

z/OS
3.2

*Cryptographic Services
Integrated Cryptographic Service Facility
System Programmer's Guide*



Note

Before using this information and the product it supports, read the information in [“Notices” on page 515.](#)

This edition applies to IBM® z/OS® 3.2 (5655-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2025-09-30

© **Copyright International Business Machines Corporation 2007, 2025.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures.....	ix
Tables.....	xi
About this information.....	xix
Who should use this information.....	xix
How to use this information.....	xix
Where to find more information.....	xx
IBM Crypto education.....	xxi
How to provide feedback to IBM.....	xxiii
Summary of changes.....	xxv
Summary of changes for z/OS 3.2.....	xxv
Changes made in Cryptographic Support for z/OS 3.1 (FMID HCR77E0).....	xxv
Chapter 1. Introduction to z/OS ICSF.....	1
Features.....	1
Cryptographic hardware features.....	1
Server hardware.....	2
z/OS ICSF FMIDs.....	3
ICSF features.....	4
The Cryptographic Key Data Set (CKDS).....	5
The Public Key Data Set (PKDS).....	5
The Token Data Set (TKDS).....	6
Additional background information.....	6
Running PCF applications on z/OS ICSF.....	7
Using RMF and SMF to monitor z/OS ICSF events.....	7
Controlling access to ICSF.....	8
Steps prior to starting installation.....	8
Chapter 2. Installation, initialization, and customization.....	9
Steps for installation and initialization.....	9
Steps to customize SYS1.PARMLIB.....	10
Creating the CKDS.....	11
Creating the PKDS.....	16
Creating the TKDS.....	20
ICSF system resource planning for random number generation.....	24
Steps to create the installation options data set.....	24
Creating an ICSF CTRACE configuration data set	26
Steps to create the ICSF startup procedure.....	28
Steps to provide access to the ICSF panels.....	30
Requiring signature verification for ICSF module CSFINPV2.....	31
Steps to start ICSF for the first time.....	32
Customizing ICSF after the first start.....	34
Parameters in the installation options data set.....	35
Dispatching priority of ICSF.....	55
Creating ICSF exits and generic services.....	55

Chapter 3. Migration.....	57
Terminology.....	57
Migrating from earlier software releases.....	58
Actions to perform before installing ICSF FMID HCR77F0.....	58
Actions to perform before the first start of ICSF FMID HCR77F0.....	61
Actions to perform after the first start of ICSF FMID HCR77F0.....	63
Callable services.....	66
CCA access control.....	81
X9.143 (TR-31) key block support.....	98
Ensure the expected P11 master key support is available.....	98
KGUP.....	98
Key store policy.....	99
DES keys.....	100
ICSF key data sets.....	101
Migrating to 24-byte DES master key.....	103
Installation options data set.....	103
Function restrictions.....	103
CICS attachment facility.....	104
Dynamic LPA load.....	104
Dynamic service update.....	104
Special secure mode.....	104
Resource Measurement Facility (RMF).....	104
System abend codes.....	105
SMF records.....	105
TKE workstation.....	105
Migrating to PCI-HSM 2016 compliance mode.....	107
Compliance warnings.....	108
Migration process.....	108
 Chapter 4. Operating ICSF.....	 113
Manually starting and stopping ICSF.....	113
ARM policy.....	114
Starting ICSF during IPL-time.....	114
Retrieving a protected key early in IPL.....	116
Modifying ICSF.....	116
Command syntax notation.....	116
How to read syntax diagrams.....	116
ICSF operator commands.....	118
Display ICSF.....	119
SETICSF.....	130
Using different configurations.....	137
Adding and removing cryptographic coprocessors.....	138
Adding cryptographic coprocessors.....	138
Steps for activating/deactivating cryptographic coprocessors.....	139
Steps to configure on/off cryptographic coprocessors.....	139
Steps for enabling/disabling cryptographic coprocessors.....	139
Dynamic service update.....	140
Considerations when using dynamic service update.....	141
Steps to initiate dynamic service update.....	141
Verifying dynamic service update.....	143
Deactivating dynamic service update.....	144
Performance considerations for using installation options.....	144
Dispatching priority of ICSF.....	144
VTAM session-level encryption.....	144
System SSL encryption.....	145
Access method services cryptographic option.....	145

Remote key loading.....	145
Event recording.....	145
System Management Facilities (SMF) recording.....	146
Message recording.....	154
Security considerations.....	154
Controlling the program environment.....	154
Controlling access to KGUP.....	155
Controlling access to CSFDUTIL.....	155
Controlling access to the callable services.....	155
Controlling access to cryptographic keys.....	155
Controlling access to secure key tokens.....	156
Scheduling changes for cryptographic keys.....	156
Controlling access to administrative panel functions.....	156
Obtaining RACF SMF log records.....	156
Debugging aids.....	156
Component trace.....	157
Abnormal endings	158
IPCS formatting routine.....	158
VERBX.....	159
Detecting ICSF serialization contention conditions.....	160
ENF signals.....	162

Chapter 5. Installation exits.....165

Types of exits.....	165
Mainline exits.....	165
Exits for the services.....	166
The PCF CKDS conversion program exit.....	166
The single-record, read-write exit.....	166
The cryptographic key data set entry retrieval exit.....	166
Security exits.....	166
The KGUP exit.....	167
Entry and return specifications.....	167
Registers at entry.....	167
Registers at return.....	168
Exits environment.....	169
Mainline exits.....	169
Service exits.....	169
CKDS entry retrieval exit.....	169
KGUP, Conversion Programs, and Single-record, Read-write exits.....	169
Security exits.....	169
Exit recovery.....	169
Mainline installation exits.....	170
Purpose and use of the exits.....	170
Environment of the exits.....	170
Installing the exits.....	171
Input.....	171
Return Codes.....	177
Services installation exits.....	178
Purpose and use of the exits.....	178
Environment of the exits.....	178
Installing the exits.....	179
Input.....	180
Return Codes.....	185
CSF_SERVICE_EXIT - ICSF callable services exit.....	186
Cryptographic key data set entry retrieval installation exit.....	188
Purpose and use of the exit.....	188
Environment of the exit.....	188

Installing the exit.....	189
Input.....	189
Return codes.....	190
PCF conversion program installation exit.....	190
Purpose and use of the exit.....	191
Environment of the exit.....	191
Installing the exit.....	191
Input.....	192
Return codes.....	193
Single-record, Read-write installation exit.....	193
Purpose and use of the exit.....	194
Environment of the exit.....	194
Installing the exit.....	194
Input.....	195
Return codes.....	196
Exit points for security installation exits.....	197
Security installation exits.....	197
Purpose and use of the exits.....	197
Environment of the exits.....	198
Installing the exits.....	198
Input.....	199
Return codes.....	200
Key generator utility program installation exit.....	201
Purpose and use of the exit.....	201
Environment of the exit.....	202
Installing the exit.....	202
Input.....	203
The SET statement.....	211
Return codes.....	211
Chapter 6. Installation-defined Callable Services.....	213
Writing a callable service.....	213
Contents of registers.....	214
Security access control checking.....	215
Checking the parameters.....	215
Link-editing the callable service.....	215
Defining a callable service.....	216
Writing a service stub.....	216
Example of a service stub.....	217
Chapter 7. Converting a CKDS from fixed length to variable length record format	223
Chapter 8. Migration from PCF to z/OS ICSF.....	227
Running PCF and z/OS ICSF on the same system.....	227
Running in compatibility mode.....	227
Running in coexistence mode.....	228
Changing the DES master key in compatibility or coexistence mode.....	229
Running in noncompatibility mode.....	229
Specifying compatibility modes during migration.....	229
Converting a PCF CKDS to ICSF format.....	230
How the PCF conversion program runs.....	230
Using the conversion program override file.....	231
Running the conversion program.....	237
Appendix A. Diagnosis reference information.....	243
Cryptographic Key Data Set (CKDS) formats.....	243
Public Key Data Set (PKDS) format.....	247

Format of the PKDS header record.....	248
Format of the PKDS record.....	249
Token data set (TKDS) format.....	250
Format of the header record of the token data set.....	250
Format of the token and object records.....	251
Common record format (KDSR).....	279
AES key token format.....	282
AES internal fixed-length key token.....	282
Token validation value.....	283
DES key token formats.....	284
DES fixed-length key token.....	284
External RKX DES key token.....	288
DES null key token.....	289
Variable-length symmetric key token formats.....	290
Variable-length symmetric key token.....	290
Variable-length symmetric null key token.....	319
X9.143 (TR-31) key block header and optional block data.....	320
PKA key token formats.....	328
Internal PKA tokens.....	329
PKA null key token.....	329
RSA key token formats.....	329
ECC key token format.....	359
Trusted blocks.....	386
Data areas.....	400
The Cryptographic Communication Vector Table (CCVT).....	400
The Cryptographic Communication Vector Table Extension (CCVE).....	402
Generic Service Table (CSFMGST).....	403
RMF measurements table.....	404

Appendix B. ICSF SMF records.....411

Record type 82 (X'52') - ICSF record.....	411
Record environment.....	412
Record mapping.....	412
Subtype 1 - Initialization/Options Refresh section.....	417
Subtype 7 - Operational key load section.....	418
Subtype 8 - Cryptographic key data set refresh section.....	419
Subtype 9 - Dynamic CKDS update.....	420
Subtype 13 - Dynamic PKDS update.....	420
Subtype 14 - Cryptographic coprocessor master key entry.....	421
Subtype 15 - PCI Cryptographic coprocessor retained key create/delete.....	422
Subtype 16 - Cryptographic Processor Interface.....	423
Subtype 18 - Cryptographic processor configuration.....	425
Subtype 20 - Cryptographic processor processing times.....	427
Subtype 21 - ICSF sysplex group change section.....	428
Subtype 22 - Trusted block create callable services section.....	429
Subtype 23 - Token data set update.....	430
Subtype 24 - Duplicate tokens found.....	430
Subtype 25 - Key store policy for key token authorization checking.....	430
Subtype 26 - Public key data set refresh.....	431
Subtype 27 - PKA key management extensions.....	432
Subtype 28 - High performance encrypted key.....	433
Subtype 29 - TKE workstation audit record.....	434
Subtype 30 - Key store policy archived and inactive KDS records.....	434
Subtype 31 - Cryptographic usage statistics.....	435
Subtype 40 - CCA symmetric key lifecycle event.....	440
Subtype 41 - CCA asymmetric key lifecycle event.....	443
Subtype 42 - PKCS#11 object lifecycle event.....	446

Subtype 44 - CCA symmetric key usage event.....	452
Subtype 45 - CCA asymmetric key usage event.....	455
Subtype 46 - PKCS#11 key usage event.....	458
Subtype 47 - PKCS#11 no key usage event.....	464
Subtype 48 - Compliance warning event.....	466
Subtype 49 - Master key event resulted in a new master key value being promoted to current....	473
Record type 1154 (X'482') Subtype 49 – ICSF Compliance Evidence.....	477
Appendix C. CICS-ICSF Attachment Facility.....	485
Installing the CICS-ICSF Attachment Facility.....	485
Steps for installing the CICS-ICSF attachment facility.....	485
Appendix D. Helpful hints for ICSF first time startup.....	489
Checklist for first-time startup of ICSF.....	489
Step 1. Hardware setup.....	489
Step 2. LPAR activation profiles.....	489
Step 3. ICSF setup.....	490
Step 4. TKE setup.....	490
Step 5. ICSF startup.....	491
Step 6. Loading master keys and initializing the CKDS through ICSF panels.....	491
Step 7. Customizing TKE and loading master keys.....	493
Step 8. CICS-ICSF Attachment Facility setup.....	494
Step 9. Complete ICSF initialization.....	494
Commonly encountered ICSF first time setup/initialization messages.....	495
Appendix E. Using AMS REPRO encryption.....	497
Steps for setting up ICSF	497
Appendix F. Systems without Cryptographic features.....	499
Applications and programs.....	499
Callable services.....	499
ICSF setup and initialization.....	500
Secure Sockets Layer (SSL).....	501
TKE workstation.....	501
Migrating from no active coprocessors to an active coprocessor environment.....	501
Appendix G. Resource names for CCA and ICSF entry points.....	503
Appendix H. Accessibility.....	513
Notices.....	515
Terms and conditions for product documentation.....	516
IBM Online Privacy Statement.....	517
Policy for unsupported hardware.....	517
Minimum supported hardware.....	517
Trademarks.....	518
Index.....	519

Figures

1. Multiple Crypto coprocessors on a complex.....	138
2. ICSF coprocessor managementpanelsICSF Coprocessor Management.....	139
3. EXPB control block for mainline exits.....	172
4. EXPB control block in the service exits.....	180
5. Example of a service entry and exit.....	215
6. Example of a service stub (1 of 5)service stubexample.....	218
7. Example of a service stub (2 of 5).....	219
8. Example of a service stub (3 of 5).....	220
9. Example of a service stub (4 of 5).....	221
10. Example of a service stub (5 of 5).....	222
11. Example of a Conversion Initial Activity Report.....	239
12. Example of a Conversion Update Activity Report.....	240

Tables

1. z/OS ICSF FMIDs.....	3
2. FMID and Hardware.....	4
3. Information about this migration action	58
4. Information about this migration action	60
5. Information about this migration action	61
6. Information about this migration action	62
7. Information about this migration action	63
8. Information about this migration action	65
9. Summary of new and changed ICSF callable services.....	66
10. Summary of new and changed CCA access controls.....	81
11. Mapping of Enterprise PKCS #11 ACPs to firmware levels.....	107
12. Syntax examples.....	117
13. DISPLAY GRS command syntax ICSF key data set ENQ resources.....	161
14. ICSF ENF codes.....	163
15. EXPB Control Block format for Mainline Exits.....	172
16. CSFEXIT1 parameters.....	173
17. CSFEXIT2 and CSFEXIT3 parameters.....	174
18. CSFEXIT4 and CSFEXIT5 parameters.....	174
19. Format of the Exit Name table.....	175
20. Compatibility services and their ICSF names.....	180
21. EXPB Control Block Format for Services.....	181
22. SPB Control Block Format.....	183
23. IXIB control block format.....	187

24. The CKDS Entry Retrieval Exit Parameters.....	190
25. CVXP Control Block Format.....	192
26. RWXP Control Block Format.....	195
27. Parameters received by the Security Service Exit.....	200
28. Parameters received by the Security Key Exit.....	200
29. KGXP Control Block Format.....	203
30. Format of Records in the Override File.....	232
31. Cryptographic Key Data Set Header Record Format.....	243
32. Cryptographic Key Data Set Record Format.....	245
33. Variable-Length Cryptographic Key Data Set Record Format.....	247
34. Public Key Data Set Header Record Format.....	248
35. Public Key Data Set Record Format.....	249
36. Format of the header record of the token data set.....	250
37. Format of the common section of the token and object records.....	252
38. Format of the unique section of the token record.....	252
39. Format of the token object flags.....	253
40. Format of the token certificate object.....	255
41. Format of the token public key object (Version 0).....	256
42. Format of the token public key object (Version 1).....	257
43. Format of the token public key object (Version 2).....	259
44. Format of the token public key object (Version 3).....	261
45. Format of the token private key object (Version 0).....	264
46. Format of the token private key object (Version 1).....	265
47. Format of the token private key object (Version 2).....	268
48. Format of the token private key object (Version 3).....	270

49. Format of the token secret key object (Version 0).....	273
50. Format of the token secret key object (Version 1).....	274
51. Format of the token secret key object (Version 3).....	275
52. Format of the token domain parameters object (Version 1).....	276
53. Format of the token domain parameters object (Version 2).....	277
54. Format of the token data object.....	278
55. Format of the KDSR record fixed data area.....	280
56. Format of KDSR metadata area.....	281
57. Format of KDSR variable-length metadata block.....	282
58. AES internal fixed-length key token format.....	282
59. DES internal fixed-length key token format.....	284
60. DES external fixed-length key token format.....	286
61. External RKX DES key-token format, version X'10'.....	288
62. Format of Null Key Tokens.....	289
63. Variable-length symmetric key token.....	290
64. DESUSECV key-usage fields.....	295
65. HMAC algorithm key-usage fields.....	295
66. AES algorithm MAC key associated data.....	297
67. AES algorithm PINCALC key associated data.....	298
68. AES algorithm PINPROT key associated data.....	299
69. AES algorithm PINPRW key associated data.....	301
70. AES algorithm DKYGENKY key associated data.....	303
71. AES algorithm SECMSG key associated data.....	308
72. AES algorithm KEK key-usage fields.....	309
73. AES algorithm CIPHER key associated data.....	311

74. AES and HMAC algorithm key-management fields.....	313
75. DESUSECV key-management fields.....	318
76. AES algorithm KDKGENKY key-usage fields.....	318
77. Variable-length symmetric null token.....	319
78. Format and supported values of the required header for a X9.143 key block.....	320
79. Key usage values and meanings.....	323
80. IBM optional block data in a TR-31 key block, control vector (ID "10").....	326
81. IBM internal X9-SWKB controls (TLV ID '02') after conversion to binary.....	328
82. Format of PKA Null Key Tokens.....	329
83. RSA Public Key Token.....	329
84. RSA Private External Key Token Basic Record Format.....	330
85. RSA Private Key Token, 1024-bit Modulus-Exponent external format.....	332
86. RSA Private Key Token, 4096-bit Modulus-Exponent external format.....	333
87. RSA Private Key Token, 4096-bit Chinese Remainder Theorem external format.....	334
88. RSA private key, 8192-bit Modulus-Exponent format with AES encrypted OPK section (X'30') external form.....	336
89. RSA private key, 8192-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') external form.....	340
90. RSA Private Internal Key Token Basic Record Format.....	345
91. RSA Private Internal Key Token, 1024-bit X'02' ME Form.....	346
92. RSA Private Internal Key Token, 1024-bit X'06' ME Form.....	347
93. RSA private key, 8192-bit Modulus-Exponent format with AES encrypted OPK section (X'30') internal form.....	349
94. RSA private key, 8192-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') internal form.....	353
95. RSA Private Internal Key Token, 4096-bit Chinese Remainder Theorem Internal Format.....	357
96. ECC Key Token Format.....	359
97. Associated Data Format for ECC Private Key Token.....	365

98. AESKW Wrapped Payload Format for ECC Private Key Token.....	365
99. QSA Private Key section with OPK (X'50').....	366
100. QSA Public Key section (X'51').....	376
101. AESKW external format structure.....	380
102. Trusted block sections.....	387
103. Trusted block header.....	388
104. Trusted block trusted RSA public-key section (X'11').....	389
105. Trusted block rule section (X'12').....	390
106. Summary of trusted block rule subsection.....	391
107. Transport key variant subsection (X'0001' of trusted block rule section (X'12')).....	392
108. Transport key rule reference subsection (X'0002') of trusted block rule section (X'12').....	393
109. Common export key parameters subsection (X'0003') of trusted block rule section (X'12').....	393
110. Source key rule reference subsection (X'0004' of trusted block rule section (X'12')).....	394
111. Export key CCA token parameters subsection (X'0005') of trusted block rule section (X'12').....	395
112. Trusted block key label (name) section X'13'.....	397
113. Trusted block information section X'14'.....	397
114. Summary of trusted block information subsections.....	397
115. Protection information subsection (X'0001') of trusted block information section (X'14').....	398
116. Activation and expiration dates subsection (X'0002') of trusted block information section (X'14').....	399
117. Trusted block application-defined data section X'15'.....	399
118. Cryptographic communication vector table.....	401
119. Cryptographic Communication Vector Table Extension.....	403
120. Generic Service Table Block Format.....	403
121. RMF measurements record format.....	404
122. RMF measurements extension.....	410

123. Format of an SMF Type 82 record for subtypes smaller than 40.....	412
124. Format of an SMF Type 82 record for subtypes 40 and higher.....	413
125. SMF record header.....	413
126. ICSF header (for all subtypes 40 or greater).....	413
127. SMF type 82 server user or end user audit section.....	414
128. Audit header.....	414
129. Tag-Length-Value (TLV) triplet structure (SMF82_triplet).....	415
130. TLV triplet tag values.....	415
131. Tag-Length-Value triplets.....	416
132. Subtype 1 Initialization/Options Refresh.....	417
133. Subtype 7 operational key entry.....	418
134. Subtype 8 Cryptographic key data set refresh.....	419
135. Subtype 9 Dynamic CKDS update.....	420
136. Subtype 13 Dynamic PKDS update.....	420
137. Subtype 14 Cryptographic coprocessor master key entry.....	421
138. Subtype 15 PCI Cryptographic coprocessor retained key create/delete.....	422
139. Subtype 16 Cryptographic Processor Interface.....	423
140. Subtype 16 Cryptographic Processor Interface audit data.....	424
141. Subtype 18 Cryptographic Processor Configuration.....	425
142. Subtype 20 Cryptographic Processor Processing Times.....	427
143. Subtype 21 ICSF Sysplex Group Change.....	428
144. Subtype 22 Trusted Block Create Callable Services.....	429
145. Subtype 23 Token Data Set Update.....	430
146. Subtype 24 Duplicate Tokens Found.....	430
147. Subtype 25 Key Store Policy Key Token Authorization Checking.....	430

148. Subtype 26 Public Key Data Set Refresh.....	431
149. Subtype 27 PKA Key Management Extensions.....	432
150. Subtype 28 High Performance Encrypted Key.....	433
151. Subtype 29 TKE Workstation Audit Record.....	434
152. Subtype 30 Archived and inactive KDS records.....	434
153. Subtype 31 Cryptographic usage statistics.....	435
154. Subtype 31 SMF82_TRIPL.....	436
155. Subtype 31 tag values.....	437
156. SMF82t31val_ALG algorithm names.....	437
157. Subtype 40 CCA symmetric key lifecycle event.....	440
158. Subtype 41 CCA asymmetric key lifecycle event.....	443
159. Subtype 42 PKCS#11 object lifecycle event.....	447
160. Subtype 44 CCA symmetric key usage event.....	452
161. Subtype 45 CCA asymmetric key usage event.....	455
162. Subtype 46 PKCS#11 key usage event.....	458
163. Subtype 47 PKCS#11 no key usage event.....	464
164. Subtype 48 Compliance warning event.....	466
165. Subtype 49 Master key event resulted in a new master key value being promoted to current.....	473
166. Record type 1154 Subtype 49 header.....	477
167. Record type 1154 Subtype 49 Data section 1.....	477
168. SMF1154_49_CLASS profile access.....	481
169. SMF1154_49_DL_CLASS KDS default label access controls.....	482
170. SMF1154_49_KDS KDS access controls.....	483
171. Record type 1154 Subtype 49 Data section 2.....	484
172. Smf1154_49_2_Alg algorithm count information.....	484

173. Resource names for CCA and ICSF entry points.....	503
--	-----

About this information

This information describes how to initialize, customize, operate, and diagnose the z/OS Integrated Cryptographic Service Facility (ICSF). The z/OS Cryptographic Services includes these components:

- z/OS Integrated Cryptographic Service Facility (ICSF)
- z/OS System Secure Socket Level Programming (SSL)
- z/OS Public Key Infrastructure Services (PKI)

ICSF is a software element of z/OS that works with the hardware cryptographic feature and the Security Server RACF (or an equivalent product) to provide secure, high-speed cryptographic services. ICSF provides the application programming interfaces by which applications request the cryptographic services.

Who should use this information

This information is intended for the system programmer. It describes the tasks that a system programmer might perform:

- Programming installation options, installation-defined callable services, and installation exits
- Creating the data sets that ICSF uses
- Migrating the system from the Cryptographic Unit Support Program (CUSP) and Programmed Cryptographic Facility (PCF) to ICSF
- Migrating to z/OS ICSF
- Starting and stopping ICSF
- Checking event recording
- Planning for security and performance considerations
- Debugging and recovering from problems

Defining and writing installation-defined callable services and installation exit routines is intended to be accomplished primarily by experienced system programmers. This information assumes that the reader has an advanced knowledge of z/OS.

How to use this information

This information is divided into descriptions of these tasks:

- Introducing ICSF
 - Chapter 1, “Introduction to z/OS ICSF,” on page 1 introduces the cryptographic key data set (CKDS), the public key data set (PKDS), and the token data set (TKDS) and provides basic information about running PCF applications on ICSF and preparing for installation.
- Initializing ICSF
 - Chapter 2, “Installation, initialization, and customization,” on page 9 describes how to customize SYS1.PARMLIB, create the CKDS, the PKDS, and the TKDS, the installations options data set, the startup procedure, and provide access to the ICSF panels. It also explains how to change the parameters in the installation options data set after the first start and introduces installation exits.
- Migration and coexistence issues
 - Chapter 3, “Migration,” on page 57 describes migration to this release of ICSF from previous releases of ICSF.

Chapter 8, “Migration from PCF to z/OS ICSF,” on page 227 describes how to migrate application programs and cryptographic key data set information to z/OS ICSF from the IBM cryptographic products CUSP/PCF.

- Customizing ICSF
 - [Chapter 6, “Installation-defined Callable Services,” on page 213](#) gives information that an experienced system programmer can use to write installation-defined callable services. It also explains how to define these callable services to ICSF and how to write service stubs to access them.
 - [Chapter 5, “Installation exits,” on page 165](#) describes the ICSF installation exits you can use to customize ICSF.
- Operating ICSF
 - [Chapter 4, “Operating ICSF,” on page 113](#) describes how to add and remove cryptographic coprocessors and to start, modify, and stop ICSF and other operating considerations.
 - [“ICSF operator commands” on page 118](#) describes the console commands available for ICSF.
 - [“Event recording” on page 145](#) describes ICSF event recording on the Security Console and SMF.
 - [Appendix B, “ICSF SMF records,” on page 411](#) describes SMF Record type 82, which is used to record information about the events and operations of ICSF. Record type 82 is written to the SMF data set at the completion of certain cryptographic functions.
 - [Appendix C, “CICS-ICSF Attachment Facility,” on page 485](#) defines steps to install the CICS-ICSF Attachment Facility.
 - [Appendix E, “Using AMS REPRO encryption,” on page 497](#) provides information on using IDCAMS REPRO ENCRYPT and DECRYPT options with ICSF.
 - [Appendix G, “Resource names for CCA and ICSF entry points,” on page 503](#) provides information on ICSF resource and entry points names.
 - [Appendix F, “Systems without Cryptographic features,” on page 499](#) describes processing and functionality support for this environment.
- Planning ICSF
 - [“Security considerations” on page 154](#) describes methods you can use to protect ICSF resources.
- Diagnosing ICSF
 - [“Debugging aids” on page 156](#) describes the use of component trace and Interactive Problem Control System (IPCS) to debug ICSF.
 - [Appendix A, “Diagnosis reference information,” on page 243](#) maps the cryptographic key data set and the cryptographic communication vector tables as reference information for use in debugging. This appendix also maps CCA key tokens (DES, AES, RSA, and ECC) and trusted blocks.
 - [Appendix D, “Helpful hints for ICSF first time startup,” on page 489](#) defines helpful hints and that you may encounter when starting ICSF for the first time.
- Miscellaneous
 - [Appendix H, “Accessibility,” on page 513](#) contains information on accessibility features in z/OS.
 - [“Notices” on page 515](#) contains information on notices, programming interface information and trademarks.

Where to find more information

The publications in the z/OS ICSF library include:

- [*z/OS Cryptographic Services ICSF Overview*](#)
- [*z/OS Cryptographic Services ICSF Administrator's Guide*](#)
- [*z/OS Cryptographic Services ICSF System Programmer's Guide*](#)
- [*z/OS Cryptographic Services ICSF Application Programmer's Guide*](#)

- [z/OS Cryptographic Services ICSF Messages](#)
- [z/OS Cryptographic Services ICSF Writing PKCS #11 Applications](#)
- [z/OS Cryptographic Services ICSF TKE Workstation User's Guide](#)

IBM Crypto education

Detailed explanations and samples pertaining to IBM cryptographic technology are provided in IBM Crypto Education (community.ibm.com/community/user/ibmz-and-linuxone/groups/community-home?CommunityKey=6593e27b-caf6-4f6c-a8a8-10b62a02509c).

How to provide feedback to IBM

We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information. For more information, see [How to send feedback to IBM](#).

Summary of changes

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line to the left of the change.

Note: IBM z/OS policy for the integration of service information into the z/OS product documentation library is documented on the z/OS Internet Library under [IBM z/OS Product Documentation Update Policy](http://www.ibm.com/docs/en/zos/latest?topic=zos-product-documentation-update-policy) (www.ibm.com/docs/en/zos/latest?topic=zos-product-documentation-update-policy).

Summary of changes for z/OS 3.2

The following content is new, changed, or no longer included in z/OS 3.2.

New

The following content is new.

September 2025 release

- None.

Changed

The following content is changed.

September 2025 release

- [“ICSF system resource planning for random number generation” on page 24](#)
- [“Parameters in the installation options data set” on page 35](#)
- [“SETICSF” on page 130](#)
- [“Format of the object-specific sections of the token object records” on page 253](#)
- [“The Cryptographic Communication Vector Table \(CCVT\)” on page 400](#)
- [“Subtype 31 - Cryptographic usage statistics” on page 435](#)
- [“Subtype 42 - PKCS#11 object lifecycle event” on page 446](#)
- [“Subtype 46 - PKCS#11 key usage event” on page 458](#)
- [“Subtype 47 - PKCS#11 no key usage event” on page 464](#)

Deleted

The following content is deleted.

September 2025 release

- None.

Changes made in Cryptographic Support for z/OS 3.1 (FMID HCR77E0)

The following changes are made for z/OS 3.1. The most recent updates are listed at the top of each section.

New

The following content is new.

June 2025 refresh

- Information about IBM z17.

January 2024 refresh

- [“Migrating from no active coprocessors to an active coprocessor environment” on page 501](#) is new.

Changed

The following content is changed.

June 2025 refresh

- Updated the following for APAR OA66797, which also applies to z/OS V2R5 (ICSF FMID HCR77D2):
 - [“Audit header and audit section” on page 414.](#)
 - [“Subtype 31 - Cryptographic usage statistics” on page 435.](#)
 - [“Subtype 40 - CCA symmetric key lifecycle event” on page 440.](#)
 - [“Subtype 44 - CCA symmetric key usage event” on page 452.](#)
 - [“Subtype 45 - CCA asymmetric key usage event” on page 455.](#)
 - [“Subtype 46 - PKCS#11 key usage event” on page 458.](#)
 - [“Subtype 47 - PKCS#11 no key usage event” on page 464.](#)
 - [“Subtype 48 - Compliance warning event” on page 466.](#)
- Updated the following for APAR OA66395, which also applies to z/OS V2R5 (ICSF FMID HCR77D2):
 - [“Callable services” on page 66.](#)
 - [“CCA access control” on page 81.](#)
 - [“PKA key token formats” on page 328.](#)
 - [“RSA private external key token” on page 330.](#)
 - [“RSA private internal key token” on page 344.](#)
 - [“QSA key token format” on page 365.](#)
 - [“AESKW external format” on page 380.](#)
 - [“RMF measurements table” on page 404.](#)
 - [“Subtype 31 - Cryptographic usage statistics” on page 435.](#)
 - [“Subtype 41 - CCA asymmetric key lifecycle event” on page 443.](#)
 - [“Subtype 42 - PKCS#11 object lifecycle event” on page 446.](#)
 - [“Subtype 45 - CCA asymmetric key usage event” on page 455.](#)
 - [“Subtype 46 - PKCS#11 key usage event” on page 458.](#)
 - [“Subtype 48 - Compliance warning event” on page 466.](#)
- Updated the following:
 - [“Manually starting and stopping ICSF” on page 113.](#)
 - [“Starting ICSF during IPL-time” on page 114.](#)

March 2024 refresh

Updated the following for APAR OA65205, which also applies to z/OS V2R5 (ICSF FMID HCR77D2) and ICSF FMID HCR77D1:

- [“Callable services” on page 66.](#)
- [“Display ICSF” on page 119.](#)

January 2024 refresh

Updated the following for APAR OA64883, which also applies to z/OS V2R5 (ICSF FMID HCR77D2) and ICSF FMID HCR77D1:

- [“Callable services” on page 66.](#)
- [“QSA key token format” on page 365.](#)
- [“AESKW external format” on page 380.](#)
- [“Subtype 30 - Key store policy archived and inactive KDS records” on page 434.](#)
- [“Subtype 31 - Cryptographic usage statistics” on page 435.](#)
- [“Subtype 41 - CCA asymmetric key lifecycle event” on page 443.](#)
- [“Subtype 42 - PKCS#11 object lifecycle event” on page 446.](#)
- [“Subtype 45 - CCA asymmetric key usage event” on page 455.](#)
- [“Subtype 46 - PKCS#11 key usage event” on page 458.](#)
- [“Subtype 48 - Compliance warning event” on page 466.](#)
- [Appendix G, “Resource names for CCA and ICSF entry points,” on page 503.](#)

September 2023 release

- [“RMF measurements table” on page 404](#) was updated.
- [Appendix B, “ICSF SMF records,” on page 411](#) was updated for CSFZSM82, the SMF record mapping macro for ICSF type 82 record, which resides in SYS1.MODGEN.
- [“Subtype 31 - Cryptographic usage statistics” on page 435](#) was updated for the BCrypt hashing algorithm.

Deleted

The following content was deleted.

June 2025 release

- Preventive service planning (PSP) buckets for many IBM products, including z/OS 2.5 and 3.1, are no longer updated. For more information, see the following IBM Support document: [PSP bucket information for IBM Z products \(www.ibm.com/support/pages/node/7127792\)](https://www.ibm.com/support/pages/node/7127792).

Chapter 1. Introduction to z/OS ICSF

ICSF is a software element of z/OS. ICSF works with the hardware cryptographic features and the Security Server (RACF element) to provide secure, high-speed cryptographic services in the z/OS environment. ICSF provides the application programming interfaces by which applications request the cryptographic services. ICSF is also the means by which the secure cryptographic features are loaded with master key values, allowing the hardware features to be used by applications. The cryptographic feature is secure, high-speed hardware that performs the actual cryptographic functions. Your processor hardware determines the cryptographic feature available to your applications.

Features

Cryptographic hardware features

This topic describes the cryptographic hardware features available. Information on adding and removing cryptographic coprocessors can be found in [*z/OS Cryptographic Services ICSF Administrator's Guide*](#).

Crypto Express8 adapter (CEX8C, CEX8P, or CEX8A)

The Crypto Express8 adapter is an asynchronous cryptographic coprocessor or accelerator. The adapter contains one cryptographic engine that can be configured as a coprocessor (CEX8C for CCA and CEX8P for PKCS #11) or as an accelerator (CEX8A). It is available on IBM z16 and IBM z17.

Crypto Express7 adapter (CEX7C, CEX7P, or CEX7A)

The Crypto Express7 adapter is an asynchronous cryptographic coprocessor or accelerator. The adapter contains one cryptographic engine that can be configured as a coprocessor (CEX7C for CCA and CEX7P for PKCS #11) or as an accelerator (CEX7A). One feature may include one or two adapters, depending on the feature code. It is available on IBM z15, IBM z16, and IBM z17.

Crypto Express6 adapter (CEX6C, CEX6P, or CEX6A)

The Crypto Express6 adapter is an asynchronous cryptographic coprocessor or accelerator. The adapter contains one cryptographic engine that can be configured as a coprocessor (CEX6C for CCA and CEX6P for PKCS #11) or as an accelerator (CEX6A). It is available on IBM z14, IBM z15, and IBM z16.

Crypto Express5 adapter (CEX5C, CEX5P, or CEX5A)

The Crypto Express5 adapter is an asynchronous cryptographic coprocessor or accelerator. The adapter contains one cryptographic engine that can be configured as a coprocessor (CEX5C for CCA and CEX5P for PKCS #11) or as an accelerator (CEX5A). It is available on IBM z14 and IBM z15.

CP Assist for Cryptographic Functions (CPACF)

CPACF is a set of cryptographic instructions available on all CPs. Use of the CPACF instructions provides improved performance. The SHA-1, SHA-2, and SHA-3 algorithms are always available on the servers where the algorithm is supported. Additional algorithms are available with the appropriate enablement. For more information, see [“Server hardware”](#) on page 2.

CP Assist for Cryptographic Functions (CPACF) DES/TDES Enablement, feature 3863, provides for clear key AES, DES, and TDES instructions. On IBM z15 and later systems, this feature also includes ECC algorithm for P-256, P-384, P-521, Ed25519, and Ed448.

Server hardware

This topic describes the servers on which the cryptographic hardware features are available.

IBM z17

The IBM z17 provides constraint relief and addresses various customer demands. It has several cryptographic features.

- CP Assist for Cryptographic Functions is implemented on every processor. SHA-1, SHA-2, and SHA-3 secure hashing and SHAKE extendable output functions are directly available to application programs.
- Feature code 3863, CP Assist for Cryptographic Functions (CPACF) DES/TDES Enablement - enables DES, TDES, and AES instructions on all CPs. This feature also includes clear key ECC algorithm for NIST and Edwards curves.
- Feature code 0909, Crypto Express8 adapter - optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. Each feature code has one hardware adapters and each can be configured as a CCA coprocessor, a PKCS #11 coprocessor, or an accelerator.
- Feature code 0908, Crypto Express8 adapter - optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. Each feature code has two hardware adapters and each can be configured as a CCA coprocessor, a PKCS #11 coprocessor, or an accelerator.
- Feature code 0899, Crypto Express7 adapter - optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. Each feature code has one hardware adapter which can be configured as a CCA coprocessor, a PKCS #11 coprocessor, or an accelerator.
- Feature code 0898, Crypto Express7 adapter - optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. Each feature code has two hardware adapters and each can be configured as a CCA coprocessor, a PKCS #11 coprocessor, or an accelerator.

Note: The IBM z17 can have at most 60 Crypto Express adapters installed.

IBM z16

The IBM z16 provides constraint relief and addresses various customer demands. It has several cryptographic features.

- CP Assist for Cryptographic Functions is implemented on every processor. SHA-1, SHA-2, and SHA-3 secure hashing and SHAKE extendable output functions are directly available to application programs.
- Feature code 3863, CP Assist for Cryptographic Functions (CPACF) DES/TDES Enablement - enables DES, TDES, and AES instructions on all CPs. This feature also includes clear key ECC algorithm for NIST and Edwards curves.
- Feature code 0909, Crypto Express8 adapter - optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The IBM z16 can support a maximum of 60 adapters and the IBM z16 Model A02 can support a maximum of 40 adapters. Each feature code has one hardware adapters and each can be configured as a CCA coprocessor, a PKCS #11 coprocessor, or an accelerator.
- Feature code 0908, Crypto Express8 adapter - optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The IBM z16 can support a maximum of 60 adapters and the IBM z16 Model A02 can support a maximum of 40 adapters. Each feature code has two hardware adapters and each can be configured as a CCA coprocessor, a PKCS #11 coprocessor, or an accelerator.
- Feature code 0899, Crypto Express7 adapter - optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The IBM z16 can support a maximum of 60 adapters. Each feature code has one hardware adapter which can be configured as a CCA coprocessor, a PKCS #11 coprocessor, or an accelerator.
- Feature code 0898, Crypto Express7 adapter - optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The IBM z16 can support a maximum of 60 adapters. Each feature code has two hardware adapters and each can be configured as a CCA coprocessor, a PKCS #11 coprocessor, or an accelerator.

- Feature code 0893, Crypto Express6 adapter - optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The IBM z16 can support a maximum of 16 adapters. Each adapter code has one hardware adapter which can be configured as a CCA coprocessor, a PKCS #11 coprocessor, or an accelerator.

Note: The IBM z16 can have at most 60 Crypto Express adapters installed.

IBM z15

The IBM z15 provides constraint relief and addresses various customer demands. It has several cryptographic features.

- CP Assist for Cryptographic Functions is implemented on every processor. SHA-1, SHA-2, and SHA-3 secure hashing and SHAKE extendable output functions are directly available to application programs.
- Feature code 3863, CP Assist for Cryptographic Functions (CPACF) DES/TDES Enablement - enables DES, TDES, and AES instructions on all CPs. This feature also includes clear key ECC algorithm for NIST and Edwards curves.
- Feature code 0899, Crypto Express7 adapter - optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The IBM z15 can support a maximum of 60 adapters and the IBM z15 Model T02 can support a maximum of 40 adapters. Each feature code has one hardware adapter which can be configured as a CCA coprocessor, a PKCS #11 coprocessor, or an accelerator.
- Feature code 0898, Crypto Express7 adapter - optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The IBM z15 can support a maximum of 60 adapters and the IBM z15 Model T02 can support a maximum of 40 adapters. Each feature code has two hardware adapters and each can be configured as a CCA coprocessor, a PKCS #11 coprocessor, or an accelerator.
- Feature code 0893, Crypto Express6 adapter - optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The IBM z15 can support a maximum of 16 adapters. Each adapter code has one hardware adapter which can be configured as a CCA coprocessor, a PKCS #11 coprocessor, or an accelerator.
- Feature code 0890, Crypto Express5 adapter - optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The IBM z15 can support a maximum of 16 adapters. Each feature code has one hardware feature which can be configured as a CCA coprocessor, a PKCS #11 coprocessor, or an accelerator.

Note: The IBM z15 can have at most 60 Crypto Express adapters installed.

z/OS ICSF FMIDs

These tables explain the relationships of z/OS releases, ICSF FMIDs and servers.

Table 1. z/OS ICSF FMIDs		
z/OS	ICSF FMID	Name
V2R3	HCR77C0	Cryptographic Support for z/OS V2R1 - z/OS V2R2.
	HCR77C1	Cryptographic Support for z/OS V2R1 - z/OS V2R3.
	HCR77D0	Cryptographic Support for z/OS V2R2 - z/OS V2R3.
	HCR77D1	Cryptographic Support for z/OS V2R2 - z/OS V2R4.
V2R4	HCR77D0	Cryptographic Support for z/OS V2R2 - z/OS V2R3.
	HCR77D1	Cryptographic Support for z/OS V2R2 - z/OS V2R4.
V2R5	HCR77D2	Cryptographic Support for z/OS V2R5.
3.1	HCR77E0	Cryptographic Support for z/OS 3.1.
3.2	HCR77F0	Cryptographic Support for z/OS 3.2.

Refer to this chart to determine what release is associated with each ICSF FMID and what server it will run on.

<i>Table 2. FMID and Hardware</i>		
ICSF FMID	Applicable z/OS Releases	Servers where FMID will run
HCR77D0 (Base of z/OS 2.4)	2.2 and 2.3	IBM z9 EC, IBM z9 BC, IBM z10 EC, IBM z10 BC, IBM z114, IBM z196, IBM zBC12, IBM zEC12, IBM z13, IBM z13s, IBM z14, IBM z14 ZR1, IBM z15, and IBM z16.
HCR77D1	2.2, 2.3, and 2.4	IBM z9 EC, IBM z9 BC, IBM z10 EC, IBM z10 BC, IBM z114, IBM z196, IBM zBC12, IBM zEC12, IBM z13, IBM z13s, IBM z14, IBM z14 ZR1, IBM z15, and IBM z16.
HCR77D2 (Base of z/OS 2.5)	2.5	IBM z13, IBM z13s, IBM z14, IBM z14 ZR1, IBM z15, IBM z15 TO2, IBM z16, and IBM z17.
HCR77E0 (Base of z/OS 3.1)	3.1	IBM z14, IBM z14 ZR1, IBM z15, IBM z15 TO2, IBM z16, IBM z16 AO2, and IBM z17.
HCR77F0 (Base of z/OS 3.2)	3.2	IBM z15, IBM z15 TO2, IBM z16, IBM z16 AO2, and IBM z17.

ICSF features

ICSF protects data from unauthorized disclosure or modification. It protects data that is stored within a system, stored in a file on magnetic tape off a system, and sent between systems. It can also be used to authenticate identities of senders and receivers and to ensure the integrity of messages transmitted over a network. It uses cryptography to accomplish these functions.

Cryptography enciphers data, using an algorithm and a cryptographic key, so the data is in an unintelligible form. Deciphering data involves reproducing the intelligible data from the unintelligible data. To encipher and decipher data, ICSF uses either the U.S. National Institute of Science and Technology Data Encryption Standard (DES) algorithm, Advanced Encryption Standard (AES), Elliptic Curve Cryptography (ECC) or the RSA algorithm.

ICSF supports several Public Key Algorithms (PKA), which do not require exchanging a secret key. You can use these algorithms to exchange AES or DES secret keys securely and to compute digital signatures for authenticating messages and users. For digital signatures, you use a pair of keys: a private (secret) key to sign a message and a corresponding public key to verify the signature. ICSF supports the RSA, and ECC algorithms.

You can call an ICSF callable service from an application program to perform a cryptographic function. ICSF uses keys in cryptographic functions to:

- Protect data
- Protect other keys
- Verify that messages were not altered between sender and receiver
- Generate, protect, and verify personal identification numbers (PINs)
- Distribute AES and DES keys
- Generate and verify digital signatures

You use ICSF callable services and programs to generate, maintain, and manage keys that are used in the cryptographic functions. A unique key performs each type of cryptographic function on ICSF. All

secret keys are encrypted under another key, a master key or a wrapping key. There are up to four CCA master keys depending on your cryptographic coprocessors: DES, RSA, AES and ECC. All master keys are physically secure within the boundary of the cryptographic coprocessors. Operational secret keys are encrypted under their respective master key.

The P11 master key is used to protect secure PKCS #11 keys. Secure PKCS #11 keys are supported only on features configured for PKCS #11. The P11 master key is physically secure within the boundary of the coprocessors.

The Cryptographic Key Data Set (CKDS)

Cryptographic keys that are protected under the DES or AES master key are stored in a VSAM data set that is called the cryptographic key data set (CKDS). ICSF provides sample CKDS allocation jobs (members CSFCKDS and CSFCKD2) in SYS1.SAMPLIB. An installation is not required to define a CKDS. However, when a CKDS is not defined, secure CCA symmetric key functions are not available and ICSF cannot be used to manage CCA symmetric key tokens. The CKDS contains individual entries for each key that is added to it. You can store all types of operational symmetric keys in the CKDS. Each record in the data set contains the key value encrypted under the master key and other information about the key. ICSF maintains two copies of the CKDS: a disk copy and an in-storage copy.

Callable services use the in-storage copy of the CKDS to perform cryptographic functions. For information on managing and sharing the CKDS in a sysplex environment, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

Applications can use the dynamic CKDS update callable services to create, write, read, and delete CKDS records.

There are four formats of the CKDS:

- Large common record format (KDSRL) that is common to all key data sets with LRECL=32756. This format supports all operational CCA symmetric key tokens and X9.143 (TR-31) key blocks along with metadata and allows ICSF to track key usage if so configured. Sample is CSFCKDS.
- Common record format (KDSR) that is common to all key data sets with LRECL=2048. This format supports all CCA symmetric key tokens along with metadata and allows ICSF to track key usage if so configured. Sample is CSFCKDS.
- Variable-length record format with LRECL=1024. This format supports all CCA symmetric key tokens.
- Fixed-length record format with LRECL=252. This format only supports fixed-length CCA symmetric key tokens.

You should use the most current format, the large common record format (KDSRL), for all your key data sets because KDSRL format supports all available key tokens and key block and supports additional function to manage cryptographic keys. For information on converting your existing CKDS to KDSR format, see [“Migrating to the common record format \(KDSR\) key data set”](#) on page 102.

If variable-length AES and HMAC keys are to be stored in the CKDS, you must use the variable-length or KDSR format of the CKDS. These formats can store all symmetric key tokens, both fixed-length and variable-length tokens. The KDSR format allows ICSF to track key usage if so configured.

When X9.143 (TR-31) key blocks are to be stored in the CKDS, you must use the large common record (KDSRL) format of the CKDS.

Notes:

- Support for the KDSRL format requires z/OS V2R5 ICSF (FMID HCR77D2) or later.
- Sample CSFCKD2 is used to create a KDSR/KDSRL CKDS for use in GDPS active-active environment.

The Public Key Data Set (PKDS)

RSA, ECC, and QSA public and private keys and trusted blocks can be stored in a VSAM data set that is called the public key data set (PKDS). ICSF provides sample PKDS allocation jobs (members CSFPKDS, CSFPKD2, and CSFPKD3) in SYS1.SAMPLIB. An installation is not required to define a PKDS. However,

when a PKDS is not defined, secure CCA asymmetric key functions are not available and ICSF cannot be used to manage CCA asymmetric key tokens. The PKDS contains individual entries for each key that is added to it. You can store public key tokens, both external and internal private key tokens, and trusted blocks in the PKDS. ICSF maintains two copies of the PKDS: a disk copy and an in-storage copy.

Callable services use the in-storage copy of the PKDS to perform cryptographic functions. For information on managing and sharing the PKDS in a sysplex environment, see [*z/OS Cryptographic Services ICSF Administrator's Guide*](#).

Applications can use the dynamic PKDS update callable services to create, write, read, and delete PKDS records.

There are three formats of the PKDS:

- Large common record format (KDSRL) that is common to all key data sets with LRECL=32756. This format supports all asymmetric key tokens and metadata and allows ICSF to track key usage if so configured. Sample is CSF PKDS.
- Common record format (KDSR) that is common to all key data sets with LRECL=3800. This format supports all asymmetric key tokens (except QSA) and metadata and allows ICSF to track key usage if so configured. Sample is CSF PKD2.
- Base record format with LRECL=3800. This format supports all asymmetric key tokens (except QSA).

You should use the most current format, the large common record format (KDSRL), for all your key data sets because the KDSRL format supports all available key tokens and key block and supports additional function to manage cryptographic keys. For information on converting your existing PKDS to KDSR format, see [“Migrating to the common record format \(KDSR\) key data set”](#) on page 102.

The Token Data Set (TKDS)

PKCS #11 tokens and objects are stored in a VSAM data set called the token data set (TKDS). ICSF provides sample TKDS allocation jobs (members CSFTKDS, CSFTKD2, and CSFTKD3) in SYS1.SAMPLIB. The TKDS contains individual entries for each token and object that is added to it. ICSF maintains two copies of the TKDS: a disk copy and an in-storage copy. Only token objects are stored in the TKDS. Session objects (which are not persistent) are stored in memory only.

The TKDS must be a key-sequenced data set with spanned variable length records and must be allocated on a permanently resident volume. It is recommended that the TKDS is cataloged in the master catalog. For information on managing and sharing the TKDS in a sysplex environment, see [*z/OS Cryptographic Services ICSF Administrator's Guide*](#).

The TKDS is optional for installations that do not use PKCS #11 services or for installations that use only clear session (non-persistent) PKCS #11 keys.

There are two formats of the TKDS: the TKDS record format (supported by all releases of ICSF), and the common record format (KDSR) that is common to all KDS types (supported by ICSF FMID HCR77A1 and later). KDSR allows ICSF to track key usage if so configured.

You should use the most current format, the common record format (KDSR), for all your key data sets because KDSR format supports additional function to manage cryptographic keys. For information on converting your existing TKDS to KDSR format, see [“Migrating to the common record format \(KDSR\) key data set”](#) on page 102.

Additional background information

These topics provide some additional background information about using ICSF with other products, such as the Programmed Cryptographic Facility (PCF).

Running PCF applications on z/OS ICSF

If your installation uses PCF, you can run PCF applications on ICSF. You can use an installation option to specify whether a PCF application runs on ICSF. If you are migrating from PCF, ICSF provides a conversion program that converts a PCF CKDS to ICSF format.

You can use your own installation services and exits to customize ICSF. You can write, define, and call your own installation-defined callable service. You can also write and define exits that ICSF calls during the processing of:

- ICSF mainline
- A callable service
- The PCF CKDS conversion program
- The key generator utility program
- CKDS access

For example, most callable services in ICSF call an exit before and after processing. Such an exit can alter return codes in a service.

ICSF System SVC 143

SVC 143 (0A8F) is an ICSF system SVC that is used by CUSP and PCF macros (GENKEY, RETKEY, CIPHER, and EMK) for SVC entry into ICSF. The SVC allows you to run a CUSP or PCF application on ICSF. See [“Running PCF and z/OS ICSF on the same system” on page 227](#) for more information about running CUSP and PCF applications on ICSF.

SVC 143 is a type 4 SVC and does not get a lock. The General Trace Facility data is:

R15 and R0

No applicable data.

R1

Address of the parameter list. The macro that is called determines the parameter list.

Using RMF and SMF to monitor z/OS ICSF events

You can run ICSF in different configurations and use installation options to affect ICSF performance. While ICSF is running, you can use Resource Measurement Facility (RMF) and System Management Facility (SMF) to monitor certain events. For example, ICSF records information in the SMF data set when ICSF changes the status of a cryptographic processor or when you enter master key parts or take an action that causes ICSF to promote a new master key to current. ICSF also sends information and diagnostic messages to data sets and consoles.

With the availability of cryptographic hardware on an LPAR basis, RMF provides performance monitoring in the Postprocessor Crypto Hardware Activity report. This report is based on SMF record type 70, subtype 2. The Monitor I gathering options on the REPORTS control statement are CRYPTO and NOCRYPTO. Specify CRYPTO to measure cryptographic hardware activity and NOCRYPTO to suppress the gathering. In addition, overview criteria is shown for the Postprocessor in the Postprocessor Workload Activity Report - Goal Mode (WLMGL) report. For more information, see [z/OS Resource Measurement Facility Report Analysis](#).

ICSF also supports enabling RMF to provide performance measurements on ICSF services (Decipher, Digital Signature Generate, Digital Signature Verify, Encipher, FPE Decipher, FPE Encipher, FPE Translate, MAC Generate, MAC Verify, One Way Hash, PIN Translate, and PIN Verify). These measurements are of the Crypto Express adapters.

For diagnosis monitoring, use Interactive Problem Control System (IPCS) to access the trace buffer and to format control blocks.

Controlling access to ICSF

For security, you should control access to ICSF resources and services. Use a security product like the Security Server (RACF) to protect cryptographic programs, keys, and services. You should also change the value of your master keys periodically.

Steps prior to starting installation

You use either ServerPac or CBPDO to install ICSF as part of the z/OS installation process.

When beginning installation:

1. Refer to *z/OS Planning for Installation* for installation planning information.
2. Check with your IBM center or search the IBM problem database to find any service information for IBM Z products.
3. Make sure that you have all needed programs and their corequisites:
 - If you want access control and auditing services for ICSF, you need the Security Server (RACF), an optional feature of z/OS, or an equivalent product.
 - If you are a Resource Measurement Facility (RMF) user, you need the Resource Measurement Facility option available with z/OS.
4. Collect all required information. The Program Directory lists publications useful during installation.
5. Confirm you have adequate DASD storage and create SMP/E DDDEF entries for each data set. See the Program Directory for details.

Chapter 2. Installation, initialization, and customization

For this topic, you need to understand these terms:

Installation options

You create an installation options data set that specifies these options. They become active when you start ICSF, customizing how ICSF runs on your system.

Startup procedure

You create an ICSF startup procedure. Along with other information, this specifies the name of the installation options data set.

SYS1.SAMPLIB

Contains samples, including an installation options data set, a CKDS allocation job, a PKDS allocation job, a startup procedure, a CICS Wait List data set, and sample JCL for SMP/E Delivery to load keys by using a pass phrase. You can update this code as necessary and generally store the updated code in SYS1.PARMLIB and SYS1.PROCLIB.

SYS1.PARMLIB

Generally contains the installation options data set. The installation options data set can alternately be a member of a partitioned or sequential data set.

SYS1.PROCLIB

Contains the startup procedure.

Steps for installation and initialization

Refer to the *z/OS Program Directory* for installation instructions. Several of the installation steps in the *z/OS Program Directory* refer you to this publication for details. This publication explains these installation steps.

Note: Because it is possible for ICSF control blocks like the DACC and CCVT to persist in storage across an ICSF restart, an IPL is required when installing a new release of ICSF.

1. Customize SYS1.PARMLIB. [“Steps to customize SYS1.PARMLIB” on page 10](#) describes this task.
2. • If the installation will define a CKDS, see [“Creating the CKDS” on page 11](#) to create the Cryptographic Key Data Set (CKDS). [“Steps to create the CKDS” on page 12](#) describes the steps.
 - If the installation will define a PKDS, see [“Creating the PKDS” on page 16](#) to create the Public Key Data Set (PKDS). [“Steps to create the PKDS” on page 16](#) describes the steps.
 - If the installation will define a TKDS, see [“Creating the TKDS” on page 20](#) to create the Token Data Set (TKDS). [“Steps to create the TKDS” on page 20](#) describes the steps.
3. If PKCS #11 support is desired, create the TKDS. [“Steps to create the TKDS” on page 20](#) describes this task.
4. Create the installation options data set. [“Steps to create the installation options data set” on page 24](#) describes this task.
5. Create the startup procedure. [“Steps to create the ICSF startup procedure” on page 28](#) describes this task.
6. Provide access to the ICSF panels. [“Steps to provide access to the ICSF panels” on page 30](#) describes this task.

Note: You only need to perform the first six steps once.

7. Start ICSF for the first time. See [“Steps to start ICSF for the first time” on page 32](#). Once ICSF has been started, Master Keys can be entered.

For additional information on ICSF first time startup, refer to [“Checklist for first-time startup of ICSF” on page 489](#). See [z/OS Cryptographic Services ICSF Administrator's Guide](#) for directions on entering Master Keys.

8. Enter Master Keys.

Other topics in this publication and [z/OS Cryptographic Services ICSF Administrator's Guide](#) provide additional installation information.

For information on installing the CICS-ICSF Attachment Facility, refer to [Appendix C, “CICS-ICSF Attachment Facility,” on page 485](#).

Steps to customize SYS1.PARMLIB

The installation options data set you will create is generally stored in SYS1.PARMLIB. If your administrator does not have access to SYS1.PARMLIB, you need to use another data set instead.

Update the data set you are using as follows:

1. Add CEE.SCEERUN, CSF.SCSFMODE, and CSF.SCSFSTUB to the LNKST concatenation. This adds the ICSF library to the z/OS library search. This is an example of an ICSF entry to the LNKST concatenation.

```
CSF.SCSFMODE
```

2. APF authorize CSF.SCSFMODE and CSF.SCSFSTUB, if LNKAUTH=APFTAB. This is an example of an ICSF entry for APF authorization.

```
APF ADD DSNAME(CSF.SCSFMODE) VOLUME(*****)
```

3. In the IKJTSOxx parameter, add CSFDAUTH and CSFDPKDS as a value in the AUTHPGM parameter list and in the AUTHTSF parameter list. This is an example of an ICSF entry in the IKJTSOxx member.

```
AUTHPGM NAMES(          /* AUTHORIZED PROGRAMS          */ +
....
....
CSFDAUTH              /* ICSF              */ +
CSFDPKDS              /* ICSF              */ +
....

AUTHTSF NAMES(          /* PROGRAMS TO BE AUTHORIZED WHEN */ +
/* WHEN CALLED THROUGH THE TSO    */ +
/* SERVICE FACILITY                */ +
....
....
CSFDAUTH              /* ICSF              */ +
CSFDPKDS              /* ICSF              */ +
```

4. If your application programmers intend to use PKCS #11 token key objects for AES Galois/Counter Mode (GCM) encryption or GMAC generation and have ICSF generate the initialization vectors, then you need to ensure that the first four bytes of the sysplex names (from parmlib member COUPLExx) and the first four bytes of the system name (from the SYSNAME parameter in the IEASYSxx parmlib member) are unique within the scope of the systems that will be sharing these tokens. z/OS currently does not impose any restrictions on uniqueness between sysplex names and system names. Eight character system names have to be unique within a sysplex. ICSF requires that the first four bytes are unique, but this is not enforced by the z/OS operating system.

This needs to be done because, for AES GCM encryption or GMAC generation, the security of the algorithm is dependent on never repeating a key, initialization vector combination for two or more distinct sets of data. In PKCS #11, applications can request that ICSF generate a new (unique) initialization vector each time AES GCM or GMAC is initiated. In fact, this is the only permitted way to perform AES GCM or GMAC when PKCS #11 is operating in FIPS mode. When ICSF generates initialization vectors, it uses the ECVTSPLX (sysplex mode) or CVTSNAME (non-sysplex mode) field as the cryptographic module name. The name ensures uniqueness if such keys are distributed to multiple systems, but only if each system is set with a unique name.

When setting ECVTSPLX or CVTSNAME to unique values, be aware that ICSF uses only the first (left most) 4 characters of these fields. For this reason, these 4 characters must be set to uniquely identify the system.

For example, suppose AES key value 123 is created on the current single-image system (known as System A) and is distributed to another system residing in a Sysplex (known as Sysplex B). Both systems will be performing GCM encryption where ICSF generates the initialization vectors. To ensure that unique initialization vectors are generated, set CVTSNAME=SYSA on System A and ECVTSPLX=PLXB on Sysplex B.

CVTSNAME is normally set from the SYSNAME=value statement in the IEASYSxx member of "SYS1.PARMLIB". For more information, see [z/OS MVS Initialization and Tuning Reference](#).

ECVTSPLX is normally set from the COUPLE SYSPLEX(value) in the COUPLExx member of "SYS1.PARMLIB". For more information, see [z/OS MVS Setting Up a Sysplex](#).

Notes:

1. If you will be using the TKE workstation on this host, you should also add CSFTTKE as a value in the AUTHCMD parameter list.
2. To change the active IKJTSOxx member of SYS.PARMLIB without an IPL, use the PARMLIB UPDATE command.

For more information, see [z/OS MVS Initialization and Tuning Guide](#) and [z/OS MVS Initialization and Tuning Reference](#).

Creating the CKDS

Installations need to understand and plan for the system resources required for managing the CKDS copy in virtual storage, particularly when the installation is deploying a very large CKDS. Refer to “[ICSF system resource planning for the CKDS](#)” on page 11 for guidelines. Once you understand these guidelines, refer to “[Steps to create the CKDS](#)” on page 12 for step-by-step instructions.

ICSF system resource planning for the CKDS

Like the PKDS and TKDS, ICSF manages a mirror copy of the CKDS data set in protected, private virtual storage to optimize cryptographic workload access to symmetric keys in the normal course of workload operation. This copy is kept current as keys are dynamically added to, and removed from, the active CKDS key store. Like any set of control information that is maintained in virtual storage, the in-storage CKDS copy must be accommodated with sufficient system central storage and auxiliary paging space resources.

Installations need to understand and plan for the system resources that are required for managing the CKDS copy in virtual storage, particularly when the installation is deploying a large CKDS. Note that “very large” is a relative assessment depending upon the installation, and might be expressed, for example, in terms of tens or hundreds of thousands of symmetric keys in the CKDS, or even millions of keys.

An in-storage copy of a CKDS that is not experiencing significant dynamic key creation or deletion activity consumes a stable amount of virtual storage, and therefore a stable amount of system backing resource. However, certain occasional but unavoidable ICSF functions such as CKDS refresh do generate a significant spike in the amount of used virtual storage, and therefore a greater temporary demand for system resources backing that virtual storage.

Given these circumstances, it is important to calculate and plan for the system central storage and auxiliary paging space that is required to support an active in-storage copy. For a CKDS shared across a sysplex environment, every active ICSF in the sysplex has an equivalent resource requirement.

Each symmetric key in the CKDS is managed with one VSAM record. Installations need to plan for the appropriate amount of combined central storage and auxiliary paging space for each VSAM record, per active ICSF. The following formula is provided to help you calculate the required system virtual storage backing resource for an active in-storage CKDS. In this formula HI-A-RBA is the allocated relative byte address for the data component of a CKDS VSAM data set. The IDCAMS LISTCAT command output for a CKDS VSAM data set can be consulted to determine the HI-A-RBA value for the data component. The

%Free Space used in this formula represents the percentage of free space in the CKDS VSAM data set. The IDCAMS EXAMINE DATATEST command output can be consulted to determine the percentage of free space.

$$HI-A-RBA \times ((100 - \%Free\ Space) / 100) \times 6$$

For example, the central storage and auxiliary paging space requirement for a CKDS VSAM data set with a HI-A-RBA of 481,787,904 for its data component entry and 16 percent free space can be calculated as follows.

$$481,787,904 \times ((100 - 16) / 100) \times 6 = 2,428,211,036.16 \text{ bytes}$$

This CKDS VSAM data set requires 2.26 Gigabytes of combined central storage and auxiliary paging space for system backing resource.

As is the case with all virtual storage usage, central storage is the preferred medium to optimize the workload performance, and to avoid system paging overhead. Excessive system paging due to any virtual storage usage can cause degradation across the workload and system operation, and an extreme shortage of central storage and auxiliary paging space can lead to a catastrophic system failure.

Note: The output from the preceding formulas should be added to the outputs calculated from the formulas in “[ICSF system resource planning for the PKDS](#)” on page 16 and “[ICSF system resource planning for the TKDS and session object memory areas](#)” on page 20. This gives you the required system virtual storage backing resource for all of ICSF's KDS data sets. This value represents the required amount of virtual storage for a given instance of ICSF. For a set of KDS data sets shared across a sysplex environment, every active ICSF in the sysplex has an equivalent resource requirement.

Additional CKDS performance considerations

IBM recommends that installations that deploy a fixed-length format CKDS with millions of symmetric keys do not enable CKDS MAC authentication or disable it if it is already enabled. CKDS MAC authentication adds an additional coprocessor request for each VSAM data set read/write operation. There is a significant performance implication for CKDS MAC authentication that would be greatly magnified with such a large CKDS.

Steps to create the CKDS

The CKDS must be a key-sequenced data. There are four formats:

- Large common record format (KDSRL) that is common to all key data sets with LRECL=32756.
- Common record format (KDSR) that is common to all key data sets with LRECL=2048.
- Variable-length record format with LRECL=1024.
- Fixed-length record format with LRECL=252.

Note: ICSF recommends using the large common record format of the CKDS which supports metadata and key usage tracking.

Allocate the CKDS on a permanently resident volume. It is recommended that the CKDS is cataloged in the master catalog.

Attention: Ensure that this volume is not subject to data set migration. If the CKDS is migrated, message CSFM450E is issued and ICSF ends.

For detailed information about calculating space for a VSAM data set and an explanation of keyed-direct update processing and what happens when control area and control interval splits occur, see [z/OS DFSMS Access Method Services Commands](#).

1. Determine the amount of primary space you need to allocate for the CKDS. This should reflect the total number of entries you expect the data set to contain originally. At a minimum, the CKDS contains just a header record.

Common record format (both KDSR and KDSRL):

The minimum size of a record is 188 bytes. Records containing fixed-length DES and AES CCA key tokens are 244 bytes long. Records containing CCA variable-length symmetric key tokens may be up to 905 bytes long. Records containing X9.143 key blocks may be up to 10180 bytes long. In addition, installations may add metadata to any record. If you are planning to add metadata or the records contain user data from conversion, account for the size of the metadata in the length of records. Allocate space for all of the installation keys you expect to store in the CKDS.

Variable length record format:

The minimum size of a record is 276 bytes. Records containing fixed-length DES and AES keys are 332 bytes long. Records containing variable-length symmetric key tokens may be up to 993 bytes long. Allocate space for all of the installation keys you expect to store in the CKDS.

Fixed length record format:

Each record is 252 bytes long. Allocate space for all of the installation keys you expect to store in the CKDS.

2. Determine the amount of secondary space to allocate for CKDS. This should reflect the total number of entries you expect to add to the data set.

To access keys, VSAM uses the key label as the VSAM key. This means that VSAM adds keys to the data set in collating sequence. That is, if two keys named A and B are in the data set, A appears earlier in the data set than B. As a result, adding keys to the data set can cause multiple VSAM control interval splits and control area splits. For example, a split might occur if the data set contains keys A, B, and E and you add C. In this case, C must be placed between B and E. These splits can leave considerable free space in the data set and can affect KGUP performance.

The amount of secondary space you allocate must take into account the number of control interval and control area splits that might occur. If the disk copy of the CKDS uses a significant amount of secondary space, you can copy it into another disk copy that you created with more primary space. You can do this by using the Access Method Services (AMS) REPRO command or the AMS EXPORT/IMPORT commands.

The BUFFERSPACE parameter on the AMS DEFINE CLUSTER command (required by Step “3” on page 13) lets VSAM optimize space for control area and control interval splits.

3. Create an empty VSAM data set to use as the CKDS. ICSF provides sample jobs to define the CKDS in SYS1.SAMPLIB.

Use the AMS DEFINE CLUSTER command to define the data set and to allocate its space.

Note: To improve security and reliability of the data that is stored on the CKDS:

- Use the ERASE parameter on the AMS DEFINE CLUSTER command. ERASE overwrites data records with binary zeros when the CKDS cluster is deleted.
- Create a data set profile for the CKDS. Ensure that no one has access to the CKDS data set by protecting the CKDS data set name resource in the DATASET class. If a data set profile is used, as opposed to using the PROTECTALL(FAIL) option for example, the profile should have a UACC of NONE.

Common record format (either original or large LRECL):

Allocate a disk copy of the CKDS by defining a VSAM cluster as in this SYS1.SAMPLIB CSFCKDS member sample:

```
//CSFCKDS   JOB <JOB CARD PARAMETERS>
//*****
//* Licensed Materials - Property of IBM
//* 5650-Z0S
//* Copyright IBM CORP. 2021
//*
//* This JCL defines a VSAM CKDS capable of variable-length records
//* in common record format (either large or regular)
//*
//* CAUTION: This is neither a JCL procedure nor a complete JOB.
//* Before using this JOB step, you will have to make the following
//* modifications:
//*
//* 1) Add the job parameters to meet your system requirements.
//*
```

```

/** 2) Be sure to change CSF to the appropriate HLQ if you choose *
/** not to use the default. *
/** 3) Change XXXXXX to the valid where you want your CKDS to *
/** reside. The CKDS needs to be on a permanently resident *
/** volume. Do not specify * (asterisk) for the valid. *
/** 4) Change lrecl to either 32756 (KDSRL) or 2048 (KDSR). *
/** Do not specify any other value. Remove CISZ if using an *
/** LRECL of 2048 (KDSR format). *
/** *
/** NOTE: This JCL is specific for creating a CKDS capable of *
/** variable-length records in KDSR format (either LRECL) *
/** *
/** *****
//DEFINE EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    DEFINE CLUSTER (NAME(CSF.CSFCKDS)          -
                    VOLUMES(XXXXXX)           -
                    RECORDS(200 100)          -
                    RECORDSIZE(372,lrecl)     -
                    KEYS(72 0)                 -
                    FREESPACE(10,10)          -
                    SHAREOPTIONS(2,3))        -
    DATA (NAME(CSF.CSFCKDS.DATA)             -
          BUFFERSPACE(100000)                 -
          ERASE                                -
          CISZ(32768))                         -
    INDEX (NAME(CSF.CSFCKDS.INDEX))
/*

```

Variable-length record format:

```

//CSFCKDV JOB <JOB CARD PARAMETERS>
//*****
/** Licensed Materials - Property of IBM *
/** 5650-ZOS *
/** Copyright IBM CORP. 2010, 2021 *
/** *
/** This JCL defines a VSAM CKDS capable of variable-length records *
/** *
/** CAUTION: This is neither a JCL procedure nor a complete JOB. *
/** Before using this JOB step, you will have to make the following *
/** modifications: *
/** *
/** 1) Add the job parameters to meet your system requirements. *
/** 2) Be sure to change CSF to the appropriate HLQ if you choose *
/** not to use the default. *
/** 3) Change XXXXXX to the valid where you want your CKDS to *
/** reside. The CKDS needs to be on a permanently resident *
/** volume. Do not specify * (asterisk) for the valid. *
/** *
/** NOTE: This JCL is specific for creating a CKDS capable of *
/** variable-length records, in non-KDSR format. There are *
/** samples for each of the other key data sets and formats. *
/** *
/** *****
//DEFINE EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    DEFINE CLUSTER (NAME(CSF.CSFCKDS)          -
                    VOLUMES(XXXXXX)           -
                    RECORDS(100 50)           -
                    RECORDSIZE(332,1024)      -
                    KEYS(72 0)                 -
                    FREESPACE(10,10)          -
                    SHAREOPTIONS(2,3))        -
    DATA (NAME(CSF.CSFCKDS.DATA)             -
          BUFFERSPACE(100000)                 -
          ERASE                                -
          INDEX (NAME(CSF.CSFCKDS.INDEX))
/*

```

Fixed-length record format:

```

//CSFCKDF JOB <JOB CARD PARAMETERS>
//*****
/** Licensed Materials - Property of IBM *
/** 5650-ZOS *
/** Copyright IBM CORP. 2002, 2021 *
/** *

```

```

/* This JCL defines a VSAM CKDS capable only of fixed-length records*
/*
/* CAUTION: This is neither a JCL procedure nor a complete JOB. *
/* Before using this JOB step, you will have to make the following *
/* modifications: *
/*
/* 1) Add the job parameters to meet your system requirements. *
/* 2) Be sure to change CSF to the appropriate HLQ if you choose *
/* not to use the default. *
/* 3) Change XXXXXX to the volid where you want your CKDS to *
/* reside. The CKDS needs to be on a permanently resident *
/* volume. Do not specify * (asterisk) for the volid. *
/*
/* NOTE: This JCL is specific for creating a CKDS capable of only *
/* fixed-length records. There are samples for each of the *
/* other key data sets and formats. *
/*
/******
//DEFINE EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    DEFINE CLUSTER (NAME(CSF.CSFCKDS)          -
                    VOLUMES(XXXXXX)           -
                    RECORDS(100 50)           -
                    RECORDSIZE(252,252)       -
                    KEYS(72 0)                -
                    FREESPACE(10,10)          -
                    SHAREOPTIONS(2,3))        -
    DATA (NAME(CSF.CSFCKDS.DATA)             -
           BUFFERSPACE(100000)               -
           ERASE)                             -
    INDEX (NAME(CSF.CSFCKDS.INDEX))
/*

```

When running in a GDPS active-active environment, there are certain attributes that are required to allow CKDS updates to be propagated to another sysplex. If that is the case, allocate a disk copy of the CKDS by defining a VSAM cluster as in this SYS1.SAMPLIB CSFCKD2 member sample:

```

//CSFCKD2 JOB <JOB CARD PARAMETERS>
//*****
/* Licensed Materials - Property of IBM *
/* 5650-Z05 *
/* Copyright IBM CORP. 2020, 2021 *
/*
/* This JCL defines a VSAM CKDS capable of variable-length records *
/* in common record format which also supports replication in a *
/* GDPS environment. *
/*
/* CAUTION: This is neither a JCL procedure nor a complete JOB. *
/* Before using this JOB step, you will have to make the following *
/* modifications: *
/*
/* 1) Add the job parameters to meet your system requirements. *
/* 2) Be sure to change CSF to the appropriate HLQ if you choose *
/* not to use the default. *
/* 3) Change XXXXXX to the volid where you want your CKDS to *
/* reside. The CKDS needs to be on a permanently resident *
/* volume. Do not specify * (asterisk) for the volid. *
/* 4) Change lrecl to either 32756 (KDSRL) or 2048 (KDSR). *
/* Do not specify any other value. Remove CISZ if using an *
/* LRECL of 2048 (KDSR format). *
/* 5) Change lsname to match the logstream name on the logstream *
/* definition. *
/*
/* NOTE: This JCL is specific for creating a CKDS capable of *
/* variable-length records in KDSR format which also supports *
/* replication in a GDPS environment. You should not use this *
/* template if the CKDS is not being replicated as that *
/* would cause unnecessary overhead. There are samples *
/* available for all key data sets and formats in GDPS and *
/* non-GDPS environments. *
/*
/******
//DEFINE EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    DEFINE CLUSTER (NAME(CSF.CSFCKDS)          -
                    VOLUMES(XXXXXX)           -
                    RECORDS(200 100)          -
                    RECORDSIZE(372,lrecl)     -

```

```

        KEYS(72 0)          -
        FREESPACE(10,10)    -
        SHAREOPTIONS(2,3)   -
        LOGREPLICATE        -
        LOGSTREAMID(1sname) -
        FRLOG(NONE))        -
DATA  (NAME(CSF.CSFCKDS.DATA) -
        BUFFERSPACE(100000) -
        ERASE                -
        CISZ(32768))         -
INDEX (NAME(CSF.CSFCKDS.INDEX))
/*

```

If instead, an existing CKDS in KDSR format is being altered to support replication in a GDPS environment, the following attributes need to be added: LOGREPLICATE, LOGSTREAMID, and FRLOG. The following shows a sample job step which can be tailored to alter the attributes of an existing CKDS to support a GDPS environment:

```

//ALTER EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
ALTER                -
CSF.CSFCKDS          -
LOGREPLICATE         -
LOGSTREAMID(1sname) -
FRLOG(NONE)
/*

```

You can change and use the Job Control Language according to the needs of your installation. Note that the JCL to define the CKDS differs from the JCL that defines the PKDS. For more information about allocating a VSAM data set, see [z/OS DFSMS Access Method Services Commands](#).

Creating the PKDS

Installations need to understand and plan for the system resources required for managing the PKDS copy in virtual storage, particularly when the installation is deploying a very large PKDS. Refer to “[ICSF system resource planning for the PKDS](#)” on [page 16](#) for guidelines. Once you understand these guidelines, refer to “[Steps to create the PKDS](#)” on [page 16](#) for step-by-step instructions.

ICSF system resource planning for the PKDS

Like the CKDS and TKDS, ICSF manages a mirror copy of the PKDS data set in protected, private virtual storage to optimize cryptographic workload access to asymmetric keys. Again, similar to the CKDS, the in-storage PKDS copy must be accommodated with sufficient system central storage and auxiliary paging space resources. The same formula that is used in the system resource planning section for the CKDS can be used to estimate the virtual storage requirement for an existing, stable PKDS (one that is not experiencing significant dynamic asymmetric key creation or deletion activity).

$$HI-A-RBA \times ((100 - \%Free\ Space) / 100) \times 6$$

As described in “[ICSF system resource planning for the CKDS](#)” on [page 11](#), the output from running the IDCAMS LISTCAT and EXAMINE DATATEST commands against a PKDS VSAM data set can be consulted to determine the data set's data component HI-A-RBA and the percentage of free space in the data set.

Note: The output from the preceding formula should be added to the outputs calculated from the formulas in “[ICSF system resource planning for the CKDS](#)” on [page 11](#) and “[ICSF system resource planning for the TKDS and session object memory areas](#)” on [page 20](#). This gives you the required system virtual storage backing resource for all of ICSF's KDS data sets. This value represents the required amount of virtual storage for a given instance of ICSF. For a set of KDS data sets shared across a sysplex environment, every active ICSF in the sysplex has an equivalent resource requirement.

Steps to create the PKDS

The PKDS must be allocated and the PKDS data set name must be specified on the PKDSN parameter of the options data set when you first start ICSF.

The PKDS must be a key-sequenced data set with variable length records. Allocate the PKDS on a permanently resident volume. It is recommended that the PKDS is cataloged in the master catalog.

1. Determine the amount of primary space you need to allocate for the PKDS.

This should reflect the total number of entries you expect the data set to contain originally. The PKDS will contain both public and private PKA keys. The average record length depends on the key type and whether it is a private key or a public key. For an RSA or ECC private key, it is 1.4 KB. For an RSA or ECC public key, it is 0.5 KB. For a QSA private key, it is 5.84 KB. For a QSA public key, it is 1.792 KB. Allocate space for a minimum of two private keys, one for digital signatures, and another for encipherment. In addition, allocate enough space for the number of public keys you expect to store in the PKDS. The number of public keys varies from system to system. Generally, only those keys that are received from other users or systems are stored in the PKDS. The public keys are used to send messages to the owners of the corresponding private keys. In addition, installations may add metadata to any record. If you are planning to add metadata, account for the size of the metadata in the length of records.

2. Determine the amount of secondary space to allocate for the PKDS.

This should reflect the total number of entries you expect to add to the data set. For detailed information about calculating space for a VSAM data set, see [z/OS DFSMS Access Method Services Commands](#).

To access keys, VSAM uses the key label as the VSAM key. This means that VSAM adds keys to the data set in collating sequence. That is, if two keys named A and B are in the data set, A appears earlier in the data set than B. As a result, adding keys to the data set can cause multiple VSAM control interval splits and control area splits. For example, a split might occur if the data set contains keys A, B, and E and you add C. In this case, C must be placed between B and E.

The amount of secondary space you allocate must take into account the number of control interval and control area splits that might occur. If the PKDS uses a significant amount of secondary space, you can copy it into another disk copy that you created with more primary space. You can do this by using the Access Method Services (AMS) REPRO command or the AMS EXPORT/IMPORT commands.

The BUFFERSPACE parameter on the AMS DEFINE CLUSTER command (required by Step “3” on page 17) lets VSAM optimize space for control area and control interval splits. For a detailed explanation of keyed-direct update processing and what happens when control area and control interval splits occur, see [z/OS DFSMS Access Method Services Commands](#).

3. Create an empty VSAM data set to use as the PKDS. ICSF provides sample jobs to define the PKDS in SYS1.SAMPLIB.

Use the AMS DEFINE CLUSTER command to define the data set and to allocate its space.

Note: To improve security and reliability of the data that is stored on the PKDS:

- Use the ERASE parameter on the AMS DEFINE CLUSTER command. ERASE overwrites data records with binary zeros when the PKDS cluster is deleted.
- Create a data set profile for the PKDS. Ensure that no one has access to the PKDS data set by protecting the PKDS data set name resource in the DATASET class. If a data set profile is used, as opposed to using the PROTECTALL(FAIL) option for example, the profile should have a UACC of NONE.
- The CISZ coded in these samples in the DATA section is a hardcoded requirement.

Common record format (either original or large LRECL):

Allocate a disk copy of the PKDS by defining a VSAM cluster as in this SYS1.SAMPLIB CSFPKDS member sample:

```
//CSFPKDS JOB <JOB CARD PARAMETERS>
//*****
//* Licensed Materials - Property of IBM *
//* 5650-Z05 *
//* Copyright IBM CORP. 2021 *
//* *
//* This JCL defines a VSAM PKDS capable of variable-length records *
//* in common record format (either large or regular) *
```

```

/*
/* CAUTION: This is neither a JCL procedure nor a complete JOB.
/* Before using this JOB step, you will have to make the following
/* modifications:
/*
/* 1) Add the job parameters to meet your system requirements.
/* 2) Be sure to change CSF to the appropriate HLQ if you choose
/* not to use the default.
/* 3) Change XXXXXX to the valid where you want your PKDS to
/* reside. The PKDS needs to be on a permanently resident
/* volume. Do not specify * (asterisk) for the valid.
/* 4) Change lrecl to either 32756 (KDSRL) or 3800 (KDSR).
/* Change lcisz to either 32768 (KDSRL) or 8192 (KDSR).
/* Do not specify any other values.
/*
/* NOTE: This JCL is specific for creating a PKDS capable of
/* variable-length records in KDSR format (either LRECL)
/*
/******
//DEFINE EXEC PGM=IDCAMS,REGION=64M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    DEFINE CLUSTER (NAME(CSF.CSFPKDS)
                   VOLUMES(XXXXXX)
                   RECORDS(100 50)
                   RECORDSIZE(800,lrecl)
                   KEYS(72 0)
                   FREESPACE(0,0)
                   SHAREOPTIONS(2,3))
    DATA (NAME(CSF.CSFPKDS.DATA)
          BUFFERSPACE(100000)
          ERASE
          CISZ(lcisiz))
    INDEX (NAME(CSF.CSFPKDS.INDEX))
/*

```

Base record format:

```

//CSFPKDB JOB <JOB CARD PARAMETERS>
//*****
/* Licensed Materials - Property of IBM
/* 5650-ZOS
/* Copyright IBM CORP. 2002, 2021
/*
/* This JCL defines a VSAM PKDS in the base record format
/*
/* CAUTION: This is neither a JCL procedure nor a complete JOB.
/* Before using this JOB step, you will have to make the following
/* modifications:
/*
/* 1) Add the job parameters to meet your system requirements.
/* 2) Be sure to change CSF to the appropriate HLQ if you choose
/* not to use the default.
/* 3) Change XXXXXX to the valid where you want your PKDS to
/* reside. The PKDS needs to be on a permanently resident
/* volume. Do not specify * (asterisk) for the valid.
/*
/* NOTE: This JCL is for creating a PKDS in base record format.
/* See CSFPKDS for the large common record format (KDSRL).
/*
/******
//DEFINE EXEC PGM=IDCAMS,REGION=64M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    DEFINE CLUSTER (NAME(CSF.CSFPKDS)
                   VOLUMES(XXXXXX)
                   RECORDS(100 50)
                   RECORDSIZE(800,3800)
                   KEYS(72 0)
                   FREESPACE(0,0)
                   SHAREOPTIONS(2,3))
    DATA (NAME(CSF.CSFPKDS.DATA)
          BUFFERSPACE(100000)
          ERASE
          CISZ(8192))
    INDEX (NAME(CSF.CSFPKDS.INDEX))
/*

```

When running in a GDPS environment, there are certain attributes that are required to allow PKDS updates to be propagated to another sysplex. If that's the case, allocate a disk copy of the PKDS by defining a VSAM cluster as in this SYS1.SAMPLIB CSFPKD2 member sample:

```
//CSFPKD2 JOB <JOB CARD PARAMETERS>
//*****
//* Licensed Materials - Property of IBM *
//* 5650-ZOS *
//* Copyright IBM CORP. 2020, 2021 *
//* *
//* This JCL defines a VSAM PKDS capable of variable-length records *
//* in common record format which also supports replication in a *
//* GDPS environment. *
//* *
//* CAUTION: This is neither a JCL procedure nor a complete JOB. *
//* Before using this JOB step, you will have to make the following *
//* modifications: *
//* *
//* 1) Add the job parameters to meet your system requirements. *
//* 2) Be sure to change CSF to the appropriate HLQ if you choose *
//* not to use the default. *
//* 3) Change XXXXXX to the valid where you want your PKDS to *
//* reside. The PKDS needs to be on a permanently resident *
//* volume. Do not specify * (asterisk) for the valid. *
//* 4) Change lrecl to either 32756 (KDSRL) or 3800 (KDSR). *
//* Change lcisz to either 32768 (KDSRL) or 8192 (KDSR). *
//* Do not specify any other values. *
//* 5) Change lsname to match the logstream name on the logstream *
//* definition. *
//* *
//* NOTE: This JCL is specific for creating a PKDS which supports *
//* replication in a GDPS environment. You should not use this *
//* template if the PKDS is not being replicated as that would *
//* cause unnecessary overhead. There are samples available *
//* for all key datasets in GDPS and non-GDPS environments. *
//* *
//*****
//DEFINE EXEC PGM=IDCAMS,REGION=64M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    DEFINE CLUSTER (NAME(CSF.CSFPKDS)
                   VOLUMES(XXXXXX)
                   RECORDS(100 50)
                   RECORDSIZE(800,lrecl)
                   KEYS(72 0)
                   FREESPACE(0,0)
                   SHAREOPTIONS(2,3)
                   LOGREPLICATE
                   LOGSTREAMID(lsname)
                   FRLOG(NONE))
    DATA (NAME(CSF.CSFPKDS.DATA)
          BUFFERSPACE(100000)
          ERASE
          CISZ(lcisz))
    INDEX (NAME(CSF.CSFPKDS.INDEX))
/*
```

If instead, an existing PKDS is being altered to support replication in a GDPS environment, the following attributes need to be added: LOGREPLICATE, LOGSTREAMID, and FRLOG. The following shows a sample job step which can be tailored to alter the attributes of an existing PKDS to support a GDPS environment:

```
//ALTER EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    ALTER
    CSF.CSFPKDS
    LOGREPLICATE
    LOGSTREAMID(lsname)
    FRLOG(NONE)
/*
```

You can change and use the Job Control Language according to the needs of your installation. Note that the JCL to define the PKDS differs from the JCL that defines the CKDS. For more information about allocating a VSAM data set, see [z/OS DFSMS Access Method Services Commands](#).

Creating the TKDS

TKDS Installations need to understand and plan for the system resources required for managing the TKDS copy in virtual storage, particularly when the installation is deploying a very large TKDS. Refer to [“ICSF system resource planning for the TKDS and session object memory areas” on page 20](#) for guidelines. Once you understand these guidelines, refer to [“Steps to create the TKDS” on page 20](#) for step-by-step instructions.

ICSF system resource planning for the TKDS and session object memory areas

Like the CKDS and PKDS, ICSF manages a mirror copy of the TKDS data set in protected, private virtual storage to optimize cryptographic workload access to persistent PKCS #11 objects (keys, certificates, and so on). Also, like the CKDS and PKDS, the in-storage TKDS copy must be accommodated with sufficient system central storage and auxiliary paging space resources. Unfortunately, the variable length nature of PKCS #11 objects makes resource estimating for the TKDS difficult. The best way to estimate the virtual storage requirement for an existing, stable TKDS (one that is not experiencing significant dynamic PKCS #11 object creation or deletion activity) is to determine the actual size of the used DATA portion of the TKDS and multiply this by 3. The following formula is provided to help you calculate the required system virtual storage backing resource for an active in-storage TKDS. In this formula HI-A-RBA is the allocated relative byte address for the data component of a TKDS VSAM data set. The IDCAMS LISTCAT command output for a TKDS VSAM data set can be consulted to determine the HI-A-RBA value for the data component. The %Free Space used in this formula represents the percentage of free space in the TKDS VSAM data set. The IDCAMS EXAMINE DATATEST command output can be consulted to determine the percentage of free space.

$$\text{HI-A-RBA} \times ((100 - \% \text{Free Space}) / 100) \times 3$$

For example, if the DATA HI-A-RBA has the value 1622016 with 56% free space, then the virtual storage requirement estimate would be $1622016 \times (44/100) \times 3 = 4282122$ bytes or 4182 Kilobytes.

In addition to the persistent PKCS #11 objects that are stored in the TKDS, applications can also make use of temporary (session) objects. These too occupy ICSF protected, private virtual storage and should be accounted for. However, since these objects are not stored in the TKDS, it is impossible to estimate their virtual storage requirements without having some knowledge of the applications that are using PKCS #11. Fortunately, most applications that use PKCS #11 use only a few PKCS #11 session objects and their storage requirements are already factored into the preceding TKDS estimate. However, some applications, such as TCP/IP's IPsec, use session objects exclusively, and can use many of them. Estimating the virtual storage requirements for these is beyond the scope of this document. Applications that use PKCS #11 session objects have an overall upper limit of 128 Megabytes per application address space for session objects.

Note: The output from the preceding formula should be added to the outputs calculated from the formulas in [“ICSF system resource planning for the CKDS” on page 11](#) and [“ICSF system resource planning for the PKDS” on page 16](#). This gives you the required system virtual storage backing resource for all of ICSF's KDS data sets. This value represents the required amount of virtual storage for a given instance of ICSF. For a set of KDS data sets shared across a sysplex environment, every active ICSF in the sysplex has an equivalent resource requirement.

Steps to create the TKDS

To enable applications to create and use persistent PKCS #11 tokens and objects using the PKCS #11 services, the TKDS must be allocated and the TKDS data set name must be specified on the TKDSN parameter of the options data set when you first start ICSF.

The TKDS must be a key-sequenced data set with variable length records. Allocate the TKDS on a permanently resident volume. It is recommended that the TKDS is cataloged in the master catalog.

For detailed information about calculating space for a VSAM data set and an explanation of keyed-direct update processing and what happens when control area and control interval splits occur, see [z/OS DFSMS Access Method Services Commands](#).

1. Determine the amount of primary space you need to allocate for the TKDS.

This should reflect the total number of entries you expect the data set to contain originally. The TKDS will contain PKCS #11 tokens and objects. Each record has a maximum size of 32 KB. A record for a token will use 0.1 KB. The minimum size of a record for objects is: Data: 1 KB, Secret Key: 1.1 KB, Public Key: 1.5 KB, Private Key: 3.4 KB, Certificate: 1 KB, Domain Parameter: 1.5KB. Allocate enough space for the number of tokens to be supported and for the number of objects to be created. In addition, installations may add metadata to any record. If you are planning to add metadata, account for the size of the metadata in the length of records. Note that session objects are not stored in the TKDS.

2. Determine the amount of secondary space to allocate for the TKDS.

This should reflect the total number of entries you expect to add to the data set.

To access tokens and objects, VSAM uses the token handle or object handle as the VSAM key. This means that VSAM adds objects to the data set in collating sequence. That is, if two objects named A and B are in the data set, A appears earlier in the data set than B. As a result, adding objects to the data set can cause multiple VSAM control interval splits and control area splits. For example, a split might occur if the data set contains objects A, B, and E and you add C. In this case, C must be placed between B and E.

The amount of secondary space you allocate must take into account the number of control interval and control area splits that might occur. If the TKDS uses a significant amount of secondary space, you can copy it into another disk copy that you created with more primary space. You can do this by using the Access Method Services (AMS) REPRO command or the AMS EXPORT/IMPORT commands.

The BUFFERSPACE parameter on the AMS DEFINE CLUSTER command (required by Step “3” on page 21) lets VSAM optimize space for control area and control interval splits.

3. Create an empty VSAM data set to use as the TKDS. Use the AMS DEFINE CLUSTER command to define the data set and to allocate its space. ICSF provides a sample job to define the TKDS in member CSFTKDS of SYS1.SAMPLIB.

Note: To improve security and reliability of the data that is stored on the TKDS:

- Use the ERASE parameter on the AMS DEFINE CLUSTER command. ERASE overwrites data records with binary zeros when the TKDS cluster is deleted.
- Create a data set profile for the TKDS. Ensure that no one has access to the TKDS data set by protecting the TKDS data set name resource in the DATASET class. If a data set profile is used, as opposed to using the PROTECTALL(FAIL) option for example, the profile should have a UACC of NONE.

4. Allocate a disk copy of the TKDS by defining a VSAM cluster with one of the following samples:

SYS1.SAMPLIB CSFTKDS member sample is used to define a TKDS in KDSR format as the target of a reencipher or coordinated conversion:

```
//CSFTKDS JOB <JOB CARD PARAMETERS>
//*****
//* Licensed Materials - Property of IBM *
//* 5650-Z0S *
//* Copyright IBM CORP. 2007, 2021 *
//* *
//* This JCL defines a VSAM TKDS in the common record format (KDSR) *
//* *
//* NOTE: If creating a VSAM TKDS as the target of reencipher, *
//* coordinated change master key, or coordinated convert *
//* operations, you should use this sample, not CSFTKD2. *
//* *
//* CAUTION: This is neither a JCL procedure nor a complete JOB. *
//* Before using this JOB step, you will have to make the following *
//* modifications: *
//* *
//* 1) Add the job parameters to meet your system requirements. *
```

```

/** 2) Be sure to change CSF to the appropriate HLQ if you choose *
/** not to use the default. *
/** 3) Change XXXXXX to the valid where you want your TKDS to *
/** reside. The TKDS needs to be on a permanently resident *
/** volume. Do not specify * (asterisk) for the valid. *
/** *
/** NOTE: This JCL is specific for creating a TKDS which is *
/** in common record format and is to be used as the target *
/** for reencipher or a coordinated operation. *
/** *
/** *****
//DEFINE EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    DEFINE CLUSTER (NAME(CSF.CSFTKDS) -
                    VOLUMES(XXXXXX) -
                    RECORDS(100 50) -
                    RECORDSIZE(2200,32756) -
                    KEYS(72 0) -
                    FREESPACE(0,0) -
                    SHAREOPTIONS(2,3)) -
    DATA (NAME(CSF.CSFTKDS.DATA) -
          BUFFERSPACE(100000) -
          CONTROLINTERVALSIZE(32768) -
          ERASE) -
    INDEX (NAME(CSF.CSFTKDS.INDEX))
/*

```

SYS1.SAMPLIB CSFTKD2 member sample is used to define a TKDS in KDSR format (for first time initialization only):

```

//CSFTKD2 JOB <JOB CARD PARAMETERS>
//*****
/** Licensed Materials - Property of IBM *
/** 5650-ZOS *
/** Copyright IBM CORP. 2013, 2021 *
/** *
/** This JCL defines a VSAM TKDS which is initialized to use common *
/** record format for first time TKDS initialization only. *
/** *
/** NOTE: If creating a VSAM TKDS as the target of reencipher, *
/** coordinated change master key, or coordinated convert *
/** operations, you should use sample CSFTKDS, not this one. *
/** *
/** CAUTION: This is neither a JCL procedure nor a complete JOB. *
/** Before using this JOB step, you will have to make the following *
/** modifications: *
/** *
/** 1) Add the job parameters to meet your system requirements. *
/** 2) Be sure to change CSF to the appropriate HLQ if you choose *
/** not to use the default. *
/** 3) Change XXXXXX to the valid where you want your TKDS to *
/** reside. The TKDS needs to be on a permanently resident *
/** volume. Do not specify * (asterisk) for the valid. *
/** *
/** NOTE: This JCL is specific for creating a TKDS which is *
/** initialized for first time use in common record format. *
/** Use sample CSFTKDS as the target for reencipher or a *
/** coordinated operation. *
/** *
/** *****
//DEFINE EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    DEFINE CLUSTER (NAME(CSF.CSFTKDS) -
                    VOLUMES(XXXXXX) -
                    RECORDS(100 50) -
                    RECORDSIZE(2200,32756) -
                    KEYS(72 0) -
                    FREESPACE(0,0) -
                    SHAREOPTIONS(2,3)) -
    DATA (NAME(CSF.CSFTKDS.DATA) -
          BUFFERSPACE(100000) -
          CONTROLINTERVALSIZE(32768) -
          ERASE) -
    INDEX (NAME(CSF.CSFTKDS.INDEX))
/*
/**-----*
/** Repro header record into the TKDS *
/**-----*

```

```

//MKHEAD EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD *

//SYSUT2 DD DSN= &&GENTMP,UNIT=SYSDA,DISP=(,PASS),
// DCB=(RECFM=FB,LRECL=156,BLKSIZE=1560),SPACE=(TRK,(1,1))
//SYSIN DD *
GENERATE MAXFLDS=10,MAXLITS=156
RECORD FIELD=(20,X'0000000000000000000000000000000000000000',,1),
FIELD=(20,X'0000000000000000000000000000000000000000',,21),
FIELD=(20,X'E3C8C4D900000000000000000000000000000000',,41),
FIELD=(20,X'0000000000000000000000000000000000000000',,61),
FIELD=(16,X'0000000000000000000000000000000000000000',,81),
FIELD=(16,X'0000000000000000000000000000000000000000',,97),
FIELD=(4,X'0000009C',,113),
FIELD=(16,X'0000000000000000000000000000000000000000',,117),
FIELD=(20,X'0000000000000000000000000000000000000000',,133),
FIELD=(4,X'00000200',,153)

/*
//REPROKSD EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSDATA DD DSN=*.MKHEAD.SYSUT2,DISP=(OLD,DELETE)
//SYSIN DD *
REPRO INFILE(SYSDATA) -
OUTDATASET(CSF.CSFTKDS)
/*

```

When running in a GDPS environment, there are certain attributes that are required to allow TKDS updates to be propagated to another sysplex. If that's the case, allocate a disk copy of the TKDS by defining a VSAM cluster as in this SYS1.SAMPLIB CSFTKD3 member sample:

```

//CSFTKD3 JOB <JOB CARD PARAMETERS>
//*****
/* Licensed Materials - Property of IBM *
/* 5650-Z0S *
/* Copyright IBM CORP. 2020, 2021 *
/* *
/* This JCL defines a VSAM TKDS initialized to use common record *
/* format which also supports replication in a GDPS environment. *
/* *
/* NOTE: If creating a VSAM TKDS as the target of reencipher, *
/* coordinated change master key, or coordinated convert *
/* operations, you should only use the DEFINE step. *
/* *
/* CAUTION: This is neither a JCL procedure nor a complete JOB. *
/* Before using this JOB step, you will have to make the following *
/* modifications: *
/* *
/* 1) Add the job parameters to meet your system requirements. *
/* 2) Be sure to change CSF to the appropriate HLQ if you choose *
/* not to use the default. *
/* 3) Change XXXXXX to the volid where you want your TKDS to *
/* reside. The TKDS needs to be on a permanently resident *
/* volume. Do not specify * (asterisk) for the volid. *
/* 4) Change lsnme to match the logstream name on the logstream *
/* definition. *
/* *
/* NOTE: This JCL is specific for creating a TKDS initialized to *
/* use common record format which also supports replication *
/* in a GDPS environment. You should not use this template if *
/* the TKDS is not being replicated as that would cause *
/* unnecessary overhead. There are samples available for all *
/* key data sets and formats in GDPS and non-GDPS *
/* environments. *
/* *
//*****
//DEFINE EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DEFINE CLUSTER (NAME(CSF.CSFTKDS) -
VOLUMES(XXXXXX) -
RECORDS(100 50) -
RECORDSIZE(2200,32756) -
KEYS(72 0) -
FREESPACE(0,0) -
SHAREOPTIONS(2,3) -
LOGREPLICATE -
LOGSTREAMID(lsnme) -
FRLOG(NONE)) -
DATA (NAME(CSF.CSFTKDS.DATA) -

```

```

                BUFFERSPACE(100000) -
                CONTROLINTERVALSIZE(32768) -
                ERASE) -
INDEX (NAME(CSF.CSFTKDS.INDEX))

/*
//*-----*
//* Repro header record into the TKDS *
//*-----*
//MKHEAD EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD *

//SYSUT2 DD DSN=&&GENTMP,UNIT=SYSDA,DISP=(,PASS),
// DCB=(RECFM=FB,LRECL=156,BLKSIZE=1560),SPACE=(TRK,(1,1))
//SYSIN DD *
GENERATE MAXFLDS=10,MAXLITS=156
RECORD FIELD=(20,X'00000000000000000000000000000000',,1),
        FIELD=(20,X'00000000000000000000000000000000',,21),
        FIELD=(20,X'E3C8C4D9000000000000000000000000',,41),
        FIELD=(20,X'00000000000000000000000000000000',,61),
        FIELD=(16,X'00000000000000000000000000000000',,81),
        FIELD=(16,X'00000000000000000000000000000000',,97),
        FIELD=(4,X'0000009C',,113),
        FIELD=(16,X'00000000000000000000000000000000',,117),
        FIELD=(20,X'00000000000000000000000000000000',,133),
        FIELD=(4,X'00000200',,153)

/*
//REPROKSD EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSDATA DD DSN=*.MKHEAD.SYSUT2,DISP=(OLD,DELETE)
//SYSIN DD *
REPRO INFILE(SYSDATA) -
      OUTDATASET(CSF.CSFTKDS)
/*

```

If instead, an existing TKDS is being altered to support replication in a GDPS environment, the following attributes need to be added: LOGREPLICATE, LOGSTREAMID, and FRLOG. The following shows a sample job step which can be tailored to alter the attributes of an existing TKDS to support a GDPS environment:

```

//ALTER EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
ALTER
  CSF.CSFTKDS -
  LOGREPLICATE -
  LOGSTREAMID(1sname) -
  FRLOG(NONE)
/*

```

You can change and use the Job Control Language according to the needs of your installation. For more information about allocating a VSAM data set, see [z/OS DFSMS Access Method Services Commands](#).

ICSF system resource planning for random number generation

Several ICSF callable services support pseudo-random number generation on behalf of system and application requests. ICSF's random number generation implementation utilizes a minimum virtual storage utilization of 256 kilobytes. To avoid system paging overhead, installations should plan for 256 kilobytes of central storage to back this storage utilization. This should be sufficient for most workloads, but for some workloads that are excessively heavy with multitasking random number generation requests, ICSF may dynamically extend that footprint 64 kilobytes at a time to optimize random number request handling.

Steps to create the installation options data set

The *installation options data set* is a file that you create that contains installation options. It becomes active when you start ICSF.

- The installation options data set can be a member of PARMLIB or a member of a partitioned data set.
- The format of each record in the data set must be fixed length or fixed block length.

- A physical line in the data set is 80 characters long. The system ignores any characters in positions 72 to 80 of the line.
- A logical line is one or more physical lines. You can group physical lines into a logical line by placing a comma at the end of the information. Only a comment can appear after the comma. The system ignores any other information between the comma and column 71.
- Continuation causes the next physical line to append immediately following the comma. The system removes all leading blanks on the next physical line.
- You can delimit comments by /* and */ and include them anywhere within the text. A comment cannot span physical records. The system removes comments from a logical line before parsing it. It ignores physical lines that contain only comments.
- Specify only one option setting or keyword on a logical line. (If you specify more than one, the system ignores all but the last one on the line. The system reports syntax errors, but the errors do not cause it to stop interpreting the file.)

ICSF provides a sample installation options data set. The sample data set uses the recommended values for each option.

1. When you are starting ICSF for the first time:

- a. Change the name of the data set on the CKDSN and PKDSN statements to the name of the empty VSAM data sets you created previously (in Step “3” on page 13 and Step “3” on page 17).
- b. For a complete description of options you may want to change after the first start, see “Customizing ICSF after the first start” on page 34.)

2. Store the updated data set in SYS1.PARMLIB.

Note: For convenience, the installation options data set generally resides in SYS1.PARMLIB. If your cryptographic administrator does not have update access to SYS1.PARMLIB, store installation options in another data set, and SAF-protect it.

The sample installation options data set is as follows in SYS1.SAMPLIB: CSFPRM00

```

/*****/
/*      LICENSED MATERIALS - PROPERTY OF IBM      */
/*      */                                          */
/*      5650-ZOS      */                          */
/*      */                                          */
/*      COPYRIGHT IBM CORP. 1990, 2013      */      */
/*      */                                          */
/*      THIS IS A SAMPLE OF THE ICSF OPTIONS DATASET      */
/*      */                                          */
/*****/
CKDSN(CSF.CSFCKDS)
PKDSN(CSF.CSFPKDS)
COMPAT(NO)
SSM(NO)
CHECKAUTH(NO)
CTRACE(CTICSF00)
USERPARM(USERPARM)
REASONCODES(ICSF)

```

Note: See “Parameters in the installation options data set” on page 35 for descriptions of these parameters.

Use of system symbols in the options data set is supported. System symbols can be used as values for any of the parameters. System symbols must be no more than 8 characters.

Note: ICSF allows the CKDS, PKDS and TKDS data set names to be a maximum of 44 characters with up to 21 qualifiers. Also, the first character must be alphabetic.

See “Parameters in the installation options data set” on page 35 for additional information.

This example shows how system symbols could be used for the CKDS and PKDS data set names. You could use a SYS1.PARMLIB(IEASYMxx) file and modify CSFPRM00.

IEASYMxx file could contain:

```

/*-----*/
/* SYSTEM SYMBOLS FOR ICSF CRYPTO */
/*-----*/
SYSDEF
SYMDEF(&CKDSN001='CSF')
SYMDEF(&CKDSN002='CSFCKDS')
SYMDEF(&PKDSN001='CSF')
SYMDEF(&PKDSN002='CSFPKDS')

```

CSFPRM00 could be modified as follows.

```

/*****
/*      LICENSED MATERIALS - PROPERTY OF IBM      */
/*      */
/*      5650-ZOS      */
/*      */
/*      COPYRIGHT IBM CORP. 1990, 2013      */
/*      */
/*      THIS IS A SAMPLE OF THE ICSF OPTIONS DATASET      */
/*      */
/*****
CKDSN(&CKDSN001..&CKDSN002)
PKDSN(&PKDSN001..&PKDSN002)
COMPAT(NO)
SSM(NO)
CHECKAUTH(NO)
CTRACE(CTICSF00)
USERPARM(USERPARM)
REASONCODES(ICSF)

```

When the machine or partition is IPLed, specify within the load parameter the symbol file that should be used. For example, if the previous symbol file was called IEASYM01, then within the load member, the IEASYM entry might look like IEASYM(00,01); where 00 denotes the IEASYM00 file (usually the system default) and 01 denotes the IEASYM01 file.

Creating an ICSF CTRACE configuration data set

Starting with ICSF FMID HCR77A1, ICSF CTRACE support has been enhanced to support configurable ICSF CTRACE options from PARMLIB. During SMP/E install, a default CTICSF00 PARMLIB member is installed in SYS1.PARMLIB. The CTICSF00 PARMLIB member provides default component trace values for ICSF. By default, ICSF CTRACE support will trace with the KdsIO, CardIO, RdIO, and SysCall filters using a 2M buffer. Configurable options are commented out within this PARMLIB member to provide examples of how to turn them on.

Note: Beginning with FMID HCR77A1, ICSF needs to have read access to all data sets in the PARMLIB concatenation to access the CTRACE parmlib member CTICSF00.

The CTICSF00 PARMLIB member can be used to create customized ICSF CTRACE Configuration Data Sets in PARMLIB. A customized ICSF CTRACE Configuration Data Set can then be specified in the ICSF Options Data Set using the new CTRACE option.

For example, CTRACE(CTICSFxx), where xx is any 2 characters that were used when copying the default CTICSF00 parmlib member.

Component tracing is active when ICSF starts using the trace options defined in the CTICSFxx PARMLIB member, where 00 is the default. If the specified PARMLIB member is incorrect or absent, ICSF CTRACE will attempt to use the default CTICSF00 PARMLIB member. If the CTICSF00 PARMLIB member is incorrect or absent, ICSF CTRACE will perform tracing using an internal default set of trace options. The operator can specify trace options individually on the TRACE CT command, or can specify the name of a CTICSFxx PARMLIB member containing the desired trace options. Using a PARMLIB member on the TRACE CT command can help minimize operator intervention and avoid syntax or keystroke errors.

The contents of the CTICSF00 PARMLIB member, is as follows:

```

/****START OF SPECIFICATIONS*****/
/*
/* $MAC (CTICSF00) COMP(05101) PROD(CSF):
/*
/*
/*01* MACRO NAME: CTICSF00
/*

```

```

/*                                                    */
/*01* DESCRIPTIVE NAME: CTRACE Options for ICSF Startup */
/*                                                    */
/*01* COPYRIGHT:                                     */
/*                                                    */
/*    LICENSED MATERIALS - PROPERTY OF IBM          */
/*                                                    */
/*    5650-ZOS                                       */
/*                                                    */
/*    COPYRIGHT IBM CORP. 2015                       */
/*                                                    */
/*    STATUS = HCR77B1                               */
/*                                                    */
/*01* FUNCTION:                                     */
/*    Define the default ICSF CTRACE options         */
/*                                                    */
/*01* COMPONENT: 05101 (CSF)                         */
/*                                                    */
/*01* DISTRIBUTION LIBRARY: PARMLIB                  */
/*                                                    */
/****END OF SPECIFICATIONS*****
TRACEOPTS
/*-----*/
/*    ON OR OFF: PICK 1                             */
/*-----*/
/*    ON                                             */
/*    OFF                                           */
/*-----*/
/*    ASID: 1 TO 16, 2-HEXBYTE VALUES              */
/*-----*/
/*    ASID(0042,0043,0044)                         */
/*-----*/
/*    JOBNAME: 1 TO 16, 8 BYTE VALUES              */
/*    This option takes 1 to 16 comma-separated 8 byte values. Each */
/*    value specified represents a jobname that should be traced by */
/*    ICSF CTRACE support. Additionally, other jobnames that begin */
/*    with the same characters will also be traced. For example, if */
/*    a USERID is specified, all TSO jobs matching USERIDc, where */
/*    'c' is a character between A-Z will be traced, and, all Unix */
/*    processes matching USERIDn, where 'n' is a number from 0-9 */
/*    will be traced.                                */
/*-----*/
/*    JOBNAME(USERID,JOBNAME1)                      */
/*-----*/
/*    BUFSIZE: A VALUE IN RANGE 16K TO 16M          */
/*-----*/
/*    BUFSIZE(2M)                                    */
/*-----*/
/*    OPTIONS: NAMES OF FUNCTIONS TO BE TRACED, OR "ALL", OR "MIN" */
/*-----*/
/*    OPTIONS(                                     */
/*        'ALL'                                     */
/*        , 'KDSIO'                                 */
/*        , 'CARDIO'                                */
/*        , 'SYSCALL'                               */
/*        , 'DEBUG'                                 */
/*        , 'RDIO'                                  */
/*        , 'RDDATA'                                */
/*        , 'MIN'                                   */
/*    )                                             */
/*-----*/
OPTIONS('KDSIO','CARDIO','SYSCALL','RDIO')

```

TRACEOPTS - This option takes a value of either ON or OFF. Turning this option OFF reduces ICSF CTRACE to use a minimal set of tracing. Turning this option OFF disables ICSF CTRACE. When OFF is specified all other trace options within the PARMLIB options data set should be commented out

ASID - This option takes 1 TO 16 comma-separated 2-hexbyte values. Each value specified represents an address space ID that should be traced by ICSF CTRACE support

JOBNAME - This option takes 1 TO 16 comma-separated 8 byte values. Each value specified represents a jobname that should be traced by ICSF CTRACE support. Additionally, other jobnames that begin with the same characters will also be traced. For example, if a USERID is specified, all TSO jobs matching USERIDc, where 'c' is a character between A-Z will be traced, and, all Unix processes matching USERIDn, where 'n' is a number from 0-9 will be traced.

BUFSIZE - This option takes a value in the range between 16K to 16M, where K represents kilobytes and M represents megabytes. This value is used to specify the ICSF CTRACE buffer size to be allocated.

OPTIONS - This option is used to specify the ICSF CTRACE filters to use for tracing. A comma-separated list of filter names, each enclosed with single quotes, may be specified. The following filters are supported by this option:

ALL - This filter provides output for all ICSF trace records regardless of their filter specification.

CARDIO - This filter traces activity with requests to cryptographic coprocessors.

DEBUG - This filter provides granular trace output for debugging specific ICSF modules. This filter should only be turned on at the direction of IBM service professionals. Turning this level of tracing on may degrade ICSF performance.

KDSIO - This filter traces update activity to the CKDS, PKDS, and TKDS.

MIN - This filter traces a minimum set of operations that are not covered by the other filters.

RDDATA - This filter traces remote device request and response messages.

RDIO - This filter traces activity pertaining to remote device I/O events.

SYSCALL - This filter traces entry and exit from ICSF callable services.

The TRACEENTRY option in the ICSF Options Data Set has been deprecated. If this option is specified, it will be ignored and will produce a CSFO0212 message.

Steps to create the ICSF startup procedure

ICSF provides two job control language programs:

- “Member CSF in SYS1.SAMPLIB” on page 28
- “Member CSF2 in SYS1.SAMPLIB” on page 29

You can use this code as the basis for your startup procedure. It is important that the JOB statement in the ICSF startup procedure includes REGION=0M,MEMLIMIT=NOLIMIT.

Member CSF in SYS1.SAMPLIB

```
//CSF PROC
//CSF EXEC PGM=CSFINIT,REGION=0M,TIME=1440,MEMLIMIT=NOLIMIT
//* When using CSFPARM DD, the installation options data set must be
//* a partitioned data set on systems running HCR77D0 or later.
//CSFPARM DD DSN=USER.PARMLIB(CSFPRM&PRM),DISP=SHR
```

Store this startup PROC in SYS1.PROCLIB (or another suitable library).

1. Change or use the sample startup procedure according to your needs.
 - a. In the sample code, the first line is the PROC statement. You can add one or more procedure variables to the PROC statement. For example, you can allow the system operator to decide at start time which member of the installation options data set to use. This example allows the operator to enter START CSF,PRM=00, specifying an alternate set of start-up options. For systems running ICSF FMID HCR77D0 or later, the procedure variable PRM must be used to point to the xx of the CSFPRMxx member containing the installation options in order to start ICSF during IPL-time.

```
//CSF PROC PRM=00
.
.
.
//CSFPARM DD DSN=MY.ICSF.PARM(CSFPRM&PRM),DISP=SHR
```

You can use the same principle to change the name of a sequential data set, if you are not using a partitioned data set. For systems running ICSF FMID HCR77D0 or later, sequential data sets are no longer supported in the ICSF startup procedure.

- b. The last line is the CSFPARM DD statement. The sample code specifies SYS1.PARMLIB as the data set where the installation options data set is stored. If you stored the installation options data set

- elsewhere, replace SYS1.PARMLIB with the name of the data set where you stored the installation options.
- c. The CSFPARM DD statement also specifies member CSFPRM00 as the name of the installation options data set. If you used a different name when you created the installation options data set (or any time you want to use other options), change this member name.
2. Store your startup procedure in SYS1.PROCLIB (or another suitable library) with a member name of your choice. (Depending on installation standards, possible names include CSF, CSFPROD, and CRYPTO.)
 3. If you use Security Server (RACF), you may need to update the RACF Started Procedure Table if you define a new started task:
 - a. Add the new started task name
 - b. Add a RACF userid to associate with the started task. See [z/OS Security Server RACF System Programmer's Guide](#) for more information.
 - c. Optionally, you can add a RACF group name.

Notes:

- SAF uses the userid associated with the ICSF address space when accessing the CKDS and PKDS named in the installation options data set both at ICSF startup and when performing coordinated functions (Coordinated Change-MK, Coordinated Refresh, or Coordinated Convert). When you perform a non-coordinated CKDS or PKDS task (Initialize, Change MK, Refresh, Convert), SAF uses the identity associated with the invoker (TSO userid when using panels under TSO/E or the userid associated with the batch address space when using a batch job).
- CSFPARM2 DD is used internally within ICSF so do not define a CSFPARM2 DD in the ICSF startup procedure.

Member CSF2 in SYS1.SAMPLIB

```
//CSF2 PROC PRM=00
//* This procedure can only be used on systems running HCR77D0 or later.
//* On systems running HCR77D0 or later, The PARM keyword can be used to read in an
//* installation options data set that resides in the parmlib concatenation.
//* The value specified on the PARM keyword will be appended to CSFPRM to
//* form the member name.
//CSF2 EXEC PGM=CSFINIT,PARM=&PRM,REGION=0M,TIME=1440,MEMLIMIT=NOLIMIT
```

Store this startup PROC in SYS1.PROCLIB (or another suitable library).

1. Change or use the sample startup procedure according to your needs.
 - a. In the sample code, the first line is the PROC statement. You can add one or more procedure variables to the PROC statement. For example, you can allow the system operator to decide at start time which member of the installation options data set to use. This example allows the operator to enter START CSF2,PRM=00, specifying an alternate set of start-up options. For systems running ICSF FMID HCR77D0 or later, the procedure variable PRM must be used to point to the xx of the CSFPRMxx member containing the installation options in order to start ICSF during IPL-time.

```
// CSF2 PROC PRM=00
:
//CSF2 EXEC PGM=CSFINIT,PARM=&PRM,REGION=0M,TIME=1440,MEMLIMIT=NOLIMIT
```

- b. The second line is the EXEC statement. You must add the PARM= keyword to this line and set it to the procedure variable containing the xx value of the CSFPRMxx member containing the installation options dataset. The CSFPRMxx member must reside in the parmlib concatenation to be read in.

```
//CSF2 PROC PRM=00
:
//CSF2 EXEC PGM=CSFINIT,PARM=&PRM,REGION=0M,TIME=1440,MEMLIMIT=NOLIMIT
```

- c. The EXEC statement derives its member name from the PARM keyword. The 00 value specified on the PARM keyword is appended to CSFPRM to form CSFPRM00. If you used a different name when

you created the installation options data set (or any time you want to use other options), change the xx value specified on the PARM keyword.

- d. Current ICSF users that want to migrate from CSFPARM DD to using the parmlib concatenation as detailed in CSF2 must name or rename their installation options data set using the CSFPRM prefix (for example, CSFPRM00).
2. Store your startup procedure in SYS1.PARMLIB (or another suitable library) with a member name using the CSFPRM prefix (for example, CSFPRM00).
3. If you use Security Server (RACF), you may need to update the RACF Started Procedure Table if you define a new started task:
 - a. Add the new started task name
 - b. Add a RACF userid to associate with the started task. See [z/OS Security Server RACF System Programmer's Guide](#) for more information.
 - c. Optionally, you can add a RACF group name.

Notes:

- SAF uses the userid associated with the ICSF address space when accessing the CKDS and PKDS named in the installation options data set both at ICSF startup and when performing coordinated functions (Coordinated Change-MK, Coordinated Refresh, or Coordinated Convert). When you perform a non-coordinated CKDS or PKDS task (Initialize, Change MK, Refresh, Convert), SAF uses the identity associated with the invoker (TSO userid when using panels under TSO/E or the userid associated with the batch address space when using a batch job).
- CSFPARM2 DD is used internally within ICSF so do not define a CSFPARM2 DD in the ICSF startup procedure.

Steps to provide access to the ICSF panels

To provide a way for the administrator to access the ICSF panels, you can create an ICSF option on the ISPF Primary Option Menu. Access the code for the ISPF Primary Option Menu panel body and perform these steps:

1. Under the % OPTION ==> _ZCMD line, add this line:

```
% <option value> - ICSF Panels
```

You can specify either a letter or number for the option value. Do not use an option value that already exists in the menu.

2. On the &ZSEL= TRANS(&ZQ line, add this information:

```
<option value>,'PANEL(CSF@PRIM) NEWAPPL(CSF)'
```

The option value should be the same value as the option value you chose to use in the preceding step.

When you access the ISPF Primary Option Menu panel, the ICSF panels option appears on the menu. You can choose the ICSF option value to access the ICSF panels.

You must also update the logon procedure that is used by ICSF administrators who will use the ICSF panels. For example:

```
//SYSPROC DD ...  
.  
.  
.  
//      DD  DSN=CSF.SCSFCLIO,DISP=SHR  
.  
.  
.  
//ISPLIB DD ...  
.  
.  
.  
//      DD  DSN=CSF.SCSFPNL0,DISP=SHR
```

```

.
.
.
//ISPMLIB DD ...
.
.
//          DD DSN=CSF.SCSFMSG0,DISP=SHR
.
.
//ISPSLIB DD ...
.
.
//          DD DSN=CSF.SCSFSKL0,DISP=SHR
.
.
// ISPTLIB
.
.
//          DD DSN=CSF.SCSFTLIB,DISP=SHR
.
.
.

```

An alternate method to access the ICSF panels is to use ISPF LIBDEF. Here is a sample clist.

```

/* Rxx */
/* IBMs ICSF */

address ispexec

"LIBDEF ISPLIB DATASET ID('CSF.SCSFPNL0') STACK"
"LIBDEF ISPMLIB DATASET ID('CSF.SCSFMSG0') STACK"
"LIBDEF ISPSLIB DATASET ID('CSF.SCSFSKL0') STACK"
"LIBDEF ISPTLIB DATASET ID('CSF.SCSFTLIB') STACK"

address tso "ALTLIB ACTIVATE APPLICATION(CLIST)
            DATASET('CSF.SCSFCLI0')"
"SELECT PANEL(CSF@PRIM) NEWAPPL(CSF) PASSLIB"
address tso "ALTLIB DEACTIVATE APPLICATION(CLIST)"

"LIBDEF ISPSLIB"
"LIBDEF ISPLIB"
"LIBDEF ISPMLIB"
"LIBDEF ISPTLIB"

```

The *z/OS Program Directory* lists additional installation steps and some of these steps depend on the system from which you are migrating. See the *z/OS Program Directory*, other topics in this publication, and *z/OS Cryptographic Services ICSF Administrator's Guide* for details about the remaining steps.

Requiring signature verification for ICSF module CSFINPV2

If your installation needs to operate z/OS PKCS #11 in compliance with the FIPS 140 standards, the integrity of the cryptographic functions shipped by IBM must be verified at your installation during ICSF startup. The load module that contains the software cryptographic functions is SYS1.SIEALNKE(CSFINPV2), and this load module is digitally signed when it is shipped from IBM. Using RACF, you can verify that the module has remained unchanged from the time it was built and installed on your system. To do this, you create a profile in the PROGRAM class for the CSFINPV2 module, and use this profile to indicate that signature verification is required before the module can be loaded.

To require signature verification for ICSF module CSFINPV2:

1. Make sure that RACF has been prepared to verify signed programs. As described in *z/OS Security Server RACF Security Administrator's Guide*, a security administrator prepares RACF to verify signed programs by creating a key ring for signature verification, and adding the code-signing CA certificate that is supplied with RACF to the key ring. If RACF has been prepared to verify signed programs, there will be a key ring dedicated to signature verification, the code-signing CA certificate will be attached to the key ring, and the PROGRAM class will be active.

- a. If RACF has been prepared to verify signed programs, the discrete profile IRR.PROGRAM.SIGNATURE.VERIFICATION in the FACILITY class will specify the name of the signature-verification key ring. To determine if a signature key ring is already active, enter the command:

```
RLIST FACILITY IRR.PROGRAM.SIGNATURE.VERIFICATION
```

If there is no discrete profile with this name, have your security administrator prepare RACF to verify signed programs using the information in [z/OS Security Server RACF Security Administrator's Guide](#).

- b. If the signature verification key ring exists, the RLIST command will display information for the discrete profile IRR.PROGRAM.SIGNATURE.VERIFICATION in the FACILITY class. The name of the signature verification key ring and the name of the key ring owner will be included in the APPLICATION DATA field of the RLIST command output. Using this information, enter the RACDCERT LISTRING command to make sure the code-signing CA certificate is attached to the key ring:

```
RACDCERT ID(key-ring-owner) LISTRING(key-ring-name)
```

The label of the code-signing CA certificate is 'STG Code Signing CA - G2'. If this label is not shown in the RACDCERT LISTRING command output, have your security administrator prepare RACF to verify signed programs using the information in [z/OS Security Server RACF Security Administrator's Guide](#).

- c. Program control must be active in order for RACF to perform signature verification processing. To make sure the PROGRAM class is active, enter the SETROPTS LIST command.

```
SETROPTS LIST
```

The ACTIVE CLASSES field of the command output should include the PROGRAM class. If it does not, have your security administrator prepare RACF to verify signed programs using the information in [z/OS Security Server RACF Security Administrator's Guide](#).

2. Create a profile for the CSFINPV2 program module in the PROGRAM class, indicating that the program must be signed. The following command specifies that the program should fail to load if the signature cannot be verified for any reason. This command also specifies that all signature verification failures should be logged.

Note: Due to space constraints, this command example appears on two lines. However, the RDEFINE command should be entered completely on one line.

```
RDEFINE PROGRAM CSFINPV2 ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))
```

You will need to activate your profile changes in the PROGRAM class.

```
SETROPTS WHEN(PROGRAM) REFRESH
```

Steps to start ICSF for the first time

Now that you have created the key data sets, the installation data set, the started procedure, and the ICSF management panels, you can start ICSF.

For additional information on starting ICSF for the first time, see [Appendix D, "Helpful hints for ICSF first time startup,"](#) on page 489.

- Created an empty data set for use as a CKDS
- Specified the CKDS name in the installation options data set
- Created an empty data set for use as a PKDS
- Specified the PKDS name in the installation options data set
- If PKCS #11 support is desired, create the TKDS

- Created a startup procedure
- Installed ICSF

Steps for initializing ICSF

You must initialize ICSF and the cryptographic coprocessors:

1. Enter the START command and the startup procedure name. In this example, CSF is the name of the startup procedure.

```
START CSF
```

When you start ICSF, you specify the name of the ICSF startup procedure you created (see “Steps to create the ICSF startup procedure” on page 28). See “Manually starting and stopping ICSF” on page 113 for more information about starting and stopping ICSF.

Note: To reuse ASIDs, the REUSASID parameter can be added to the START comment:

```
START CSF,REUSASID=YES
```

2. Access the ICSF panels to define a master key and initialize the CKDS and PKDS. For a description of how to use the ICSF panels to define a master key and initialize the CKDS and PKDS at first-time startup, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

If you intend to use secure key PKCS #11 services, you will also need to initialize the TKDS. This step is optional and may be deferred until a later time. Initializing the TKDS requires entering the master key using a TKE workstation. For more information, see [z/OS Cryptographic Services ICSF TKE Workstation User's Guide](#).

When defining a master key by specifying master key parts, **make sure the key parts are recorded and saved in a secure location**. When you are entering the key parts for the first time, be aware that **you may need to reenter these same key values at a later date to restore master key values that have been cleared**. If defining a master key using a pass phrase, realize that the same pass phrase will always produce the same master key values, and is therefore as critical and sensitive as the master key values themselves. Make sure you save the pass phrase so that you can later reenter it if needed. Because of the sensitive nature of the pass phrase, make sure you secure it in a safe place.

3. When you start ICSF for the first time, you will see different messages depending on your system hardware. The following examples show the messages returned on a IBM zEnterprise EC12 machine with one Crypto EXPRESS6 CCA coprocessor and one Crypto EXPRESS6 EP11 cryptographic coprocessor.
- First time startup messages before master keys have been loaded and the CKDS, PKDS, and TKDS have not been initialized:

```
S CSF
CSFM608I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM608I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM654I KEY ARCHIVING USE CONTROL IS DISABLED.
CSFM015I FIPS 140 SELF CHECKS FOR PKCS11 SERVICES SUCCESSFUL.
CSFM111I CRYPTOGRAPHIC FEATURE IS ACTIVE. CRYPTO EXPRESS6 COPROCESSOR 6Cxx, SERIAL NUMBER nnnnnnnn.
CSFM111I CRYPTOGRAPHIC FEATURE IS ACTIVE. CRYPTO EXPRESS6 COPROCESSOR 6Pxx, SERIAL NUMBER nnnnnnnn.
CSFM131E CRYPTOGRAPHY - SECURE KEY PKCS11 SERVICES ARE NOT AVAILABLE.
CSFM102I TOKEN DATA SET, CSF.TKDS IS NOT INITIALIZED FOR SECURE KEY PKCS11.
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.CKDS IS NOT INITIALIZED.
CSFM101E PKA KEY DATA SET, CSF.PKDS IS NOT INITIALIZED.
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.
CSFM698I DOMAIN IN USE: 4
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
CSFM001I ICSF INITIALIZATION COMPLETE
```

- First time startup messages before master keys have been loaded and sharing an initialized CKDS, PKDS, and TKDS:

```
S CSF
CSFM608I A CKDS KEY STORE POLICY IS NOT DEFINED.
```

```

CSFM608I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM654I KEY ARCHIVING USE CONTROL IS DISABLED.
CSFM015I FIPS 140 SELF CHECKS FOR PKCS11 SERVICES SUCCESSFUL.
CSFM124I MASTER KEY P11 ON CRYPTO EXPRESS6 COPROCESSOR 6Pxx, SERIAL NUMBER nnnnnnnn, NOT
INITIALIZED.
CSFM124I MASTER KEY DES ON CRYPTO EXPRESS6 COPROCESSOR 6Cxx, SERIAL NUMBER nnnnnnnn, NOT
INITIALIZED.
CSFM124I MASTER KEY AES ON CRYPTO EXPRESS6 COPROCESSOR 6Cxx, SERIAL NUMBER nnnnnnnn, NOT
INITIALIZED.
CSFM124I MASTER KEY RSA ON CRYPTO EXPRESS6 COPROCESSOR 6Cxx, SERIAL NUMBER nnnnnnnn, NOT
INITIALIZED.
CSFM124I MASTER KEY ECC ON CRYPTO EXPRESS6 COPROCESSOR 6Cxx, SERIAL NUMBER nnnnnnnn, NOT
INITIALIZED.

CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.
CSFM698I DOMAIN IN USE: 4
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE
AVAILABLE.
CSFM001I ICSF INITIALIZATION COMPLETE

```

- Normal ICSF restart messages. Master key registers are valid and match the CKDS/PKDS/TKDS:

```

S CSF
CSFM608I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM608I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM654I KEY ARCHIVING USE CONTROL IS DISABLED.
CSFM015I FIPS 140 SELF CHECKS FOR PKCS11 SERVICES
SUCCESSFUL.
CSFM129I MASTER KEY P11 ON CRYPTO EXPRESS6 COPROCESSOR 6Pxx, SERIAL NUMBER nnnnnnnn, IS CORRECT.
CSFM129I MASTER KEY DES ON CRYPTO EXPRESS6 COPROCESSOR 6Cxx, SERIAL NUMBER nnnnnnnn, IS CORRECT.
CSFM129I MASTER KEY AES ON CRYPTO EXPRESS6 COPROCESSOR 6Cxx, SERIAL NUMBER nnnnnnnn, IS CORRECT.
CSFM129I MASTER KEY RSA ON CRYPTO EXPRESS6 COPROCESSOR 6Cxx, SERIAL NUMBER nnnnnnnn, IS CORRECT.
CSFM129I MASTER KEY ECC ON CRYPTO EXPRESS6 COPROCESSOR 6Cxx, SERIAL NUMBER nnnnnnnn, IS CORRECT.
CSFM111I CRYPTOGRAPHIC FEATURE IS ACTIVE. CRYPTO EXPRESS6 COPROCESSOR 6Cxx, SERIAL NUMBER nnnnnnnn.
CSFM111I CRYPTOGRAPHIC FEATURE IS ACTIVE. CRYPTO EXPRESS6 COPROCESSOR 6Pxx, SERIAL NUMBER nnnnnnnn.
CSFM132I SECURE KEY PKCS11 SERVICES AVAILABLE.
CSFM698I DOMAIN IN USE: 4
CSFM400I CRYPTOGRAPHY - SERVICES ARE NOW AVAILABLE.
CSFM130I CRYPTOGRAPHY - RSA SERVICES ARE AVAILABLE.
CSFM130I CRYPTOGRAPHY - DES SERVICES ARE AVAILABLE.
CSFM130I CRYPTOGRAPHY - ECC SERVICES ARE AVAILABLE.
CSFM127I CRYPTOGRAPHY - AES SERVICES ARE AVAILABLE.
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE
AVAILABLE.
CSFM001I ICSF INITIALIZATION COMPLETE

```

Notes:

1. When you are starting ICSF for the first time and loading the first master key and initializing one or more CKDS, PKDS, or TKDS, you provide the name of the empty VSAM data set you defined previously (see “Steps to create the PKDS” on page 16 step 3) to use for the CKDS, PKDS, and TKDS when starting ICSF.
2. While ICSF processes the data set, it requires exclusive use so that no one can make changes while the data set is read. ICSF releases the data set when it completes startup processing.
3. During CKDS, PKDS, and TKDS initialization or refresh, ICSF reads the CKDS, PKDS, or TKDS into extended private storage. Specify MEMLIMIT=NOLIMIT to ensure that ICSF does not run out of virtual storage.
4. You can also write application programs to call services to perform cryptographic functions. See “Exits for the services” on page 166 for details.

Customizing ICSF after the first start

The startup procedure can include a CSFPARM DD statement, which gives the name of the installation options data set. The installation options data set can alternatively be located in the parmlib concatenation if the startup procedure has been configured to read the CSFPRMxx member from that location.

After the first start, whenever you restart ICSF, the CKDS and PKDS named in the installation options data set becomes the active in-storage CKDS and PKDS.

In order for changes to the installation options dataset to take effect, **stop and restart ICSF**. A subset of option parameters in the installation options data set are refreshable starting with ICSF FMID HCR77C0. See the SETICSF command or ICSF Multi-Purpose Service (CSFMPS and CSFMPS6) for details. To change the active in-storage CKDS or PKDS, stop and restart ICSF, or use the REFRESH option of the Master Key Management panel.

Parameters in the installation options data set

The installation options data set is an intended programming interface.

When specifying parameter values within parentheses, leading and trailing blanks are ignored. Embedded blanks may cause unpredictable results.

Support is provided for the use of system symbols in the installation options data set. System symbols can be used as values for any of the parameters. System symbols are specified in the IEASYMxx member of SYS1.PARMLIB; the IEASYM statement of the LOADxx member of SYS1.PARMLIB specifies the IEASYMxx member or members to be used for the resolution of system symbols. This example shows the use of a system symbol for specifying the domain to be used for the start of ICSF:

```
DOMAIN(&PARDOM.)
```

When the Installation Options Data Set is processed during the start of ICSF, the value of the system symbol PARDOM will be substituted as the value of the DOMAIN parameter.

For the first start, you specified an empty VSAM data set name for the CKDS in the CKDSN option and an empty VSAM data set name for the PKDS in the PKDSN option. You may want to change these and other options for subsequent starts. Here is a complete list of installation options:

AUDITKEYLIFECKDS(TOKEN(YES or NO),LABEL(YES or NO))

Provides a set of options that control auditing of events related to the lifecycle of keys that can be stored in the CKDS (CCA symmetric key tokens and X9.143 key blocks). The audit logs are in the form of Type 82 SMF records.

TOKEN(YES or NO)

Controls lifecycle auditing of CCA symmetric key tokens and X9.143 key blocks when passed to an ICSF service as a key token or key block.

Value

Indication

YES

Indicates ICSF should audit lifecycle events related to CCA symmetric key tokens and X9.143 key blocks when passed to an ICSF service as a key token or key block. An SMF type 82 subtype 40 record is logged for each event.

NO

No lifecycle auditing of CCA symmetric key tokens and X9.143 key blocks occur when passed to an ICSF service as a key token or key block.

LABEL(YES or NO)

Controls lifecycle auditing of CKDS labels.

Value

Indication

YES

Indicates ICSF should audit lifecycle events related to CKDS labels. An SMF type 82 subtype 40 record is logged for each event. The subtype 40 record replaces the subtype 9 record.

NO

No lifecycle auditing of CKDS labels occurs. ICSF continues to log an SMF type 82 subtype 9 record for CKDS updates.

If the AUDITKEYLIFECKDS option is not specified, the default is AUDITKEYLIFECKDS (TOKEN(NO),LABEL(NO)).

Note:

1. An event that involves a token is considered to be any request that uses a token as opposed to a label. This is true regardless of Key Store Policy enablement.
2. If auditing of CKDS labels is enabled, the Key Generator Utility Program (KGUP) needs access to the CSFGKF profile in the CSFSERV class in order to generate the key fingerprint for keys it processes.

For more information about the events that are audited as well as the information contained in the audit record, see Appendix B in [z/OS Cryptographic Services ICSF System Programmer's Guide](#) for the description for the subtype 40 record.

The auditing of key lifecycle events can also be controlled via the SETICSF operator command. See the description of the SETICSF command in [z/OS Cryptographic Services ICSF System Programmer's Guide](#) for more information.

The value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

AUDITKEYLIFEPKDS(TOKEN(YES or NO),LABEL(YES or NO))

Provides a set of options that control auditing of events related to the lifecycle of CCA asymmetric tokens. The audit logs are in the form of Type 82 SMF records.

TOKEN(YES or NO)

Controls lifecycle auditing of PKDS tokens.

Value

Indication

YES

Indicates ICSF should audit lifecycle events related to PKDS tokens. An SMF type 82 subtype 41 record is logged for each event.

NO

No lifecycle auditing of PKDS tokens occurs.

LABEL(YES or NO)

Controls lifecycle auditing of PKDS labels.

Value

Indication

YES

Indicates ICSF should audit lifecycle events related to PKDS labels. An SMF type 82 subtype 41 record is logged for each event. The subtype 41 record replaces the subtype 13 record.

NO

No lifecycle auditing of PKDS labels occurs. ICSF continues to log an SMF type 82 subtype 13 record for PKDS updates.

If the AUDITKEYLIFEPKDS option is not specified, the default is AUDITKEYLIFEPKDS (TOKEN(NO),LABEL(NO)).

Note: An event that involves a token is considered to be any request that uses a token as opposed to a label. This is true regardless of Key Store Policy enablement.

For more information about the events that are audited as well as the information contained in the audit record, see Appendix B in [z/OS Cryptographic Services ICSF System Programmer's Guide](#) for the description for the subtype 41 record.

The auditing of key lifecycle events can also be controlled via the SETICSF operator command. See the description of the SETICSF command in [z/OS Cryptographic Services ICSF System Programmer's Guide](#) for more information.

The value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

AUDITKEYLIFETKDS(TOKENOBJ(YES or NO),SESSIONOBJ(YES or NO))

Provides a set of options that control auditing of events related to the lifecycle of PKCS #11 objects. The audit logs are in the form of Type 82 SMF records.

TOKENOBJ(YES or NO)

Controls lifecycle auditing of PKCS #11 token objects.

Value

Indication

YES

Indicates ICSF should audit lifecycle events related to PKCS #11 token objects. An SMF type 82 subtype 42 record is logged for each event. The subtype 42 record replaces the subtype 23 record.

NO

No lifecycle auditing of PKCS #11 token objects occurs. ICSF continues to log an SMF type 82 subtype 23 record for TKDS updates.

SESSIONOBJ(YES or NO)

Controls lifecycle auditing of PKCS #11 session objects.

Value

Indication

YES

Indicates ICSF should audit lifecycle events related to PKCS #11 session objects. An SMF type 82 subtype 42 record is logged for each event.

NO

No lifecycle auditing of PKCS #11 session objects occurs.

If the AUDITKEYLIFETKDS option is not specified, the default is AUDITKEYLIFETKDS (TOKENOBJ(NO),SESSIONOBJ(NO)).

For more information about the events that are audited as well as the information contained in the audit record, see Appendix B in *z/OS Cryptographic Services ICSF System Programmer's Guide* for the description for the subtype 42 record.

The auditing of key lifecycle events can also be controlled via the SETICSF operator command. See the description of the SETICSF command in *z/OS Cryptographic Services ICSF System Programmer's Guide* for more information.

The value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

AUDITKEYUSGCKDS(TOKEN(YES or NO),LABEL(YES or NO),INTERVAL(n))

Provides a set of options that control auditing of events related to the usage of keys that can be stored in the CKDS (CCA symmetric key tokens and X9.143 key blocks). The audit logs are in the form of Type 82 SMF records.

TOKEN(YES or NO)

Controls usage auditing of CCA symmetric key tokens and X9.143 key blocks when passed to an ICSF service as a key token or key block.

Value

Indication

YES

Indicates ICSF should audit usage events related to CCA symmetric key tokens and X9.143 key blocks when passed to an ICSF service as a key token or key block. An SMF type 82 subtype 44 record is logged for each event.

NO

No usage auditing of CCA symmetric key tokens and X9.143 key blocks occur when passed to an ICSF service as a key token or key block.

LABEL(YES or NO)

Controls usage auditing of CKDS labels.

Value

Indication

YES

Indicates ICSF should audit usage events related to CKDS labels. An SMF type 82 subtype 44 record is logged for each event.

NO

No usage auditing of CKDS labels occurs.

INTERVAL(n)

Defines the time interval over which the audit records are aggregated. Specify *n* as a decimal value in hours from 1 through 24. Individual usages with the same user, service, and key are aggregated over the interval into a single SMF record with a usage count. For performance reasons, ICSF may temporarily reduce the length of an interval from what was specified.

If the AUDITKEYUSGCKDS option is not specified, the default is AUDITKEYUSGCKDS(TOKEN(NO),LABEL(NO),INTERVAL(24)).

Note: An event that involves a token is considered to be any request that uses a token as opposed to a label. This is true regardless of Key Store Policy enablement.

For more information about the information contained in the audit record, see Appendix B in [z/OS Cryptographic Services ICSF System Programmer's Guide](#) for the description for the subtype 44 record.

The auditing of key usage events can also be controlled via the SETICSF operator command. See the description of the SETICSF command in [z/OS Cryptographic Services ICSF System Programmer's Guide](#) for more information.

The value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

AUDITKEYUSGPKDS(TOKEN(YES or NO),LABEL(YES or NO),INTERVAL(n))

Provides a set of options that control auditing of events related to the usage of CCA asymmetric tokens. The audit logs are in the form of Type 82 SMF records.

TOKEN(YES or NO)

Controls usage auditing of PKDS tokens.

Value

Indication

YES

Indicates ICSF should audit usage events related to PKDS tokens. An SMF type 82 subtype 45 record is logged for each event.

NO

No usage auditing of PKDS tokens occurs.

LABEL(YES or NO)

Controls usage auditing of PKDS labels.

Value

Indication

YES

Indicates ICSF should audit usage events related to PKDS labels. An SMF type 82 subtype 45 record is logged for each event.

NO

No usage auditing of PKDS labels occurs.

INTERVAL(n)

Defines the time interval over which the audit records are aggregated. Specify *n* as a decimal value in hours from 1 through 24. Individual usages with the same user, service, and key are aggregated over the interval into a single SMF record with a usage count. For performance reasons, ICSF may temporarily reduce the length of an interval from what was specified.

If the AUDITKEYUSGPKDS option is not specified, the default is AUDITKEYUSGPKDS(TOKEN(NO),LABEL(NO),INTERVAL(24)).

Note: An event that involves a token is considered to be any request that uses a token as opposed to a label. This is true regardless of Key Store Policy enablement.

For more information about the information contained in the audit record, see Appendix B in *z/OS Cryptographic Services ICSF System Programmer's Guide* for the description for the subtype 45 record.

The auditing of key usage events can also be controlled via the SETICSF operator command. See the description of the SETICSF command in *z/OS Cryptographic Services ICSF System Programmer's Guide* for more information.

The value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

AUDITPKCS11USG(TOKENOBJ(YES or NO),SESSIONOBJ(YES or NO),NOKEY(YES or NO),INTERVAL(n))

Provides a set of options that control auditing of usage events related to PKCS #11 services. The audit logs are in the form of Type 82 SMF records.

TOKEN(YES or NO)

Controls usage auditing of PKCS #11 token objects.

Value

Indication

YES

Indicates ICSF should audit usage events related to PKCS #11 token objects. An SMF type 82 subtype 46 record is logged for each event.

NO

No usage auditing of PKCS #11 token objects occurs.

SESSIONOBJ(YES or NO)

Controls usage auditing of PKCS #11 session objects.

Value

Indication

YES

Indicates ICSF should audit usage events related to PKCS #11 session objects. An SMF type 82 subtype 46 record is logged for each event.

NO

No usage auditing of PKCS #11 session objects occurs.

NOKEY(YES or NO)

Controls usage auditing of PKCS #11 services that do not involve an object.

Value

Indication

YES

Indicates ICSF should audit relevant usages that do not pertain to a PKCS #11 object. Relevant usages include use of the PKCS #11 One-way hash, sign, or verify (CSFPFRF) and PKCS #11 Pseudo-random function (CSFPOWH) services. An SMF type 82 subtype 47 record is logged for each event.

NO

No usage auditing of PKCS #11 services that do not involve an object occurs.

INTERVAL(n)

Defines the time interval over which the audit records are aggregated. Specify *n* as a decimal value in hours from 1 through 24. Individual usages with the same user, service, and key are aggregated over the interval into a single SMF record with a usage count. For performance reasons, ICSF may temporarily reduce the length of an interval from what was specified.

If the AUDITPKCS11USG option is not specified, the default is AUDITPKCS11USG(TOKENOBJ(NO),SESSIONOBJ(NO),NOKEY(NO), INTERVAL(24)).

For more information about the information contained in the audit record, see Appendix B in *z/OS Cryptographic Services ICSF System Programmer's Guide* for the description for the subtypes 46 and 47 records.

The auditing of key usage events can also be controlled via the SETICSF operator command. See the description of the SETICSF command in *z/OS Cryptographic Services ICSF System Programmer's Guide* for more information.

The value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

BEGIN(fmid)

Specifies that parameters following this BEGIN parameter are supported in release *fmid* and later. There must be an END statement to complete the current section. If not, an error message will be issued and ICSF will terminate.

There may be any number of BEGIN/END pairs in the data set, but they cannot be nested within each other. A BEGIN must have a matching END before another BEGIN can be specified.

If the release of ICSF you are running is at this release or later, the parameters will be parsed and processed. If release of ICSF you are running is an earlier release, the parameters will be ignored.

It is recommended that when your systems are all running releases that support newer parameters that the BEGIN/END pair be removed.

The following FMIDs are supported: HCR77D0, HCR77D1, HCR77D2, HCR77E0, and HCR77F0.

The value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

Here is an example of the usage of the BEGIN/END parameters.

```
parameter4      /* parameter4 is supported by all releases */
BEGIN(HCR7751)
parameter1      /* parameter1 added in HCR7751 */
parameter3      /* parameter3 added in HCR7751 */
END
BEGIN(HCR7770)
parameter2      /* parameter2 added in HCR7770 */
END
parameter5      /* parameter5 is supported by all releases */
```

CHECKAUTH(YES or NO)

Indicates whether ICSF performs security access control checking of Supervisor State or System Key callers. If you specify CHECKAUTH(YES), ICSF issues RACROUTE calls to perform the security access control checking. If you specify CHECKAUTH(NO), the authorization checks against resources in the CSFSERV, CSFKEYS, and XCSFKEY classes are not performed.

If you do not specify the CHECKAUTH option, the default is CHECKAUTH(NO).

If you configure CHECKAUTH(YES) in the ICSF options dataset, the Health Checker address space user identity must be authorized to the CSFRKL profile in class CSFSERV for the ICSFMIG7731_ICSF_RETAINED_RSAKEY migration check to successfully execute. However, you have no action to take if you choose not to run the migration check. If you configure CHECKAUTH(NO), there is no requirement to authorize the Health Checker user identity for any ICSF profiles or classes, since the check routine executes in supervisor state. This is not an implementation consideration, but rather a check deployment or activation time customer administration consideration.

The value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

CICSAUDIT(YES or NO)

Indicates whether ICSF logs CICS client identity information on SAF calls that check the CICS address space access to the CSFSERV, CSFKEYS, and XCSFKEY classes. CICSAUDIT(NO) is the default.

If you specify CICSAUDIT(YES), when a CICS transaction running on the Quasi Reentrant (QR) task calls an ICSF service, ICSF subsequently calls a CICS service to obtain the client identity information. This information is then constructed into a log string, which is passed to the security product.

The following identity information is collected:

- Userid.
- X500 certificate information:

X500_IDN

The IDN string is truncated to 255 bytes if a longer value is present.

X500_SDN

The SDN string is truncated to 255 bytes if a longer value is present.

- Distributed Identity Data (IDID):
 - IDID user name (in UTF8 format).
 - IDID user name format.
 - Distributed registry name (in UTF8 format).

CICSAUDIT(YES) should only be specified if you are collecting SMF type 80, event code 2 (resource access) records.

By processing the resulting SMF log, you can determine which CICS users are accessing which ICSF services and which keys are being used.

CKDSN(data-set-name)

Specifies the CKDS name the system uses to start ICSF. Whenever you restart ICSF, the CKDS named in the CKDSN option becomes the active in-storage CKDS. (At first-time startup, you should specify the name of an empty VSAM data set you created to use as the CKDS.)

If you do not specify this keyword, you will not be able to use secure CCA symmetric keys or use ICSF to manage CCA symmetric keys. ICSF must be restarted in order to switch between having a CKDS and not having a CKDS.

See [“Steps to create the installation options data set” on page 24](#) for the data set naming format requirements.

CKTAUTH(YES or NO)

This keyword is no longer supported, but is tolerated.

COMPAT(YES, NO, or COEXIST)

Indicates whether ICSF runs in compatibility mode, non-compatibility mode, or coexistence mode with PCF.

YES

Indicates **compatibility mode**.

In compatibility mode, you can run a PCF application on ICSF because ICSF supports the PCF macros. You do not have to reassemble PCF applications to do this. You cannot start PCF at the same time as ICSF on the same operating system.

NO

Indicates **non-compatibility mode**. In noncompatibility mode, you run PCF applications on PCF and ICSF applications on ICSF. You cannot run PCF applications on ICSF because ICSF does not support the PCF macros in this mode. PCF can be started at the same time as ICSF on the same operating system. You can start ICSF and then start PCF, or you can start PCF and then start ICSF.

You should use noncompatibility mode unless you are migrating from PCF to ICSF.

COEXIST

Indicates **coexistence mode**.

In coexistence mode, you can run a PCF application on PCF, or you can reassemble the PCF application to run on ICSF. To do this, you reassemble the application against coexistence macros that are shipped with ICSF. You can start PCF at the same time as ICSF on the same operating system.

If you do not specify the COMPAT option, the default value is COMPAT(NO). See [“Running PCF and z/OS ICSF on the same system”](#) on page 227 for a complete description of the COMPAT options.

When you initialize ICSF for the first time, noncompatibility mode must be active. Therefore, at first-time startup, you must specify COMPAT(NO)

or allow the default to be used.

COMPENC(DES or CDMF)

This keyword is no longer supported, but is tolerated.

COMPLIANCEWARN(PCIHSM2016(YES or NO or SAF))

Indicates whether ICSF should generate compliance warning events for a compliance mode.

Compliance warning events can be used to help migrate an application to a given compliance mode. Compliance warning events are written in the form of SMF type 82 subtype 48 records. If you do not specify the COMPLIANCEWARN option, the default is NO for all compliance modes.

PCIHSM2016(YES or NO or SAF)

Controls warning events for the PCI-HSM 2016 compliance mode. If you do not specify the PCIHSM2016 option, the default is NO.

Value

Indication

YES

Generate compliance warning events for all applications.

NO

No compliance warning events are generated.

SAF

Generate compliance warning events for applications which have READ access to the CSF.COMPLIANCEWARN.PCIHSM2016 profile in the XFACILIT SAF class.

For more information about the information contained in the SMF record, see [Appendix B, “ICSF SMF records,”](#) on page 411 for the description of the subtype 48 record.

For more information on when a compliance warning event is written, including how you can use compliance warning events to help migrate an application, see [Chapter 3, “Migration,”](#) on page 57.

The generation of compliance warning events can also be controlled with the SETICSF,OPT REFRESH operator command. For more information, see [“SETICSF”](#) on page 130.

The COMPLIANCEWARN option is not intended to be enabled for an extended period. It should be enabled for a specific period during which a representative sample of the relevant workload is run. At this point, it should be disabled and the results (SMF type 82 subtype 48 records) examined to see what operations are compliant or non-compliant. Make changes (if possible) to make non-compliant operations compliant, re-enable the option, and repeat the process until either there are no non-compliant results remaining or the remaining non-compliant results cannot be made compliant. See [Chapter 3, “Migration,”](#) on page 57 for more information about migrating applications to PCI-HSM mode.

CTRACE(CTICSFxx)

Specifies the CTICSFxx ICSF CTRACE configuration data set to use from PARMLIB. CTICSF00 is the default ICSF CTRACE configuration data set that is installed with ICSF FMID HCR77A1 and later releases. CTICSF00 may be copied to create new PARMLIB members using the naming convention of CTICSFxx, where xx is a unique value specified by the user.

This parameter is optional. If the specified PARMLIB member is incorrect or absent, ICSF CTRACE will attempt to use the default CTICSF00 PARMLIB member. If the CTICSF00 PARMLIB member is incorrect or absent, ICSF CTRACE will perform tracing using an internal default set of trace options. By default, ICSF CTRACE support will trace with the KdsIO, CardIO, and SysCall filters using a 2M buffer. For more information refer to [“Creating an ICSF CTRACE configuration data set”](#) on page 26s.

DEFAULTWRAP(internal_wrapping_method,external_wrapping_method)

Specifies the default key wrapping for DES keys in CCA key tokens. Any token generated or updated by a service will be wrapped using the specified method unless overridden by rule array keyword or a skeleton token. The default wrapping method for internal and external tokens is specified independently.

Valid values for *internal_wrapping_method* and *external_wrapping_method* are:

ORIGINAL

Specifies the original CCA DES token wrapping be used: ECB wrapping.

ENHANCED

Specifies the X9.24 enhanced wrapping method version 1 with SHA-1 be used. The enhanced wrapping method with SHA-1 is available on IBM zEnterprise 196, IBM zEnterprise 114, and later servers.

WRAPENH3

Specifies the enhanced wrapping method version 3 with SHA-256 and CMAC authentication code be used. The default wrapping setting for enhanced wrapping method version 3 is available on IBM z16 and later servers. If you specify WRAPENH3 on an older server, the enhanced wrapping method version 1 will be substituted.

If the DEFAULTWRAP parameter is not specified, the default wrapping method is ORIGINAL for both internal and external tokens.

Note: Triple-length DES keys (except DATA keys with a zero control vector) may be wrapped with the enhanced method version 2 or version 3. The default wrapping method is version 2. When WRAPENH3 is specified, enhanced method version 3 will be used.

During initialization, ICSF changes the setting of the default wrapping method for all CCA coprocessors to match the value that is specified by this parameter. Default wrapping for WRAPENH3 is available on CEX8C and later coprocessors. When WRAPENH3 is specified for a coprocessor that does not support WRAPENH3, ENHANCED is used. If you specify WRAPENH3 and there are no CEX8C coprocessors available, the enhanced wrapping method version 1 will be substituted.

Notes:

- The value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).
- Starting with ICSF FMID HCR77C1 on IBM z14 and later servers, the ENHANCED wrapping method should be used if you are using PCI-HSM compliant tagged keys.
- Starting with ICSF FMID HCR77D1 with the PTF for APAR OA60318 applied on IBM z13, z13s, and later servers with the May 2021 licensed internal code (LIC), the WRAPENH3 wrapping method should be used if you are using PCI-HSM compliant tagged keys. This will enable WRAPENH3 as the default wrapping method for CEX8C and ENHANCED to be the default wrapping method for previous coprocessors. To enable WRAPENH3 as the default wrapping method on previous coprocessors, see 'DES key wrapping method control' in [z/OS Cryptographic Services ICSF Administrator's Guide](#).

DOMAIN(n)

Specifies the number of the domain that you want to use for this start of ICSF. You can specify only one domain in the options data set. The domain value must match the activation profile.

DOMAIN is an optional parameter. The DOMAIN parameter is only required if more than one domain is specified as the usage domain on the PR/SM panels. If specified in the options data set, it will be used and it must be one of the usage domains for the LPAR.

If DOMAIN is not specified in the options data set, ICSF determines which domains are available in this LPAR. If only one domain is defined for the LPAR, ICSF will use it. If more than one is available, ICSF will issue error message CSFM409E.

The cryptographic processors support multiple sets of master key registers, which the specific domain values identify.

- The CCA Crypto Express adapters have master key registers for the DES-MK, AES-MK and RSA-MK master keys. Each domain has a master key register for the current, new, and old DES-MK, AES-MK and RSA-MK.
- CCA Crypto Express adapters that are CEX3C or later have master key registers for the DES-MK, AES-MK, RSA-MK, and ECC-MK master keys. Each domain has a master key for the current, new, and old DES-MK, AES-MK, RSA-MK, and ECC-MK.
- The PKCS #11 cryptographic coprocessors have master key registers for the P11-MK master key. Each domain has a master key for the current and new P11-MK.

For more information about partitions and running different configurations, see [z/OS Cryptographic Services ICSF Overview](#).

If you run ICSF in compatibility or coexistence mode, you cannot change the domain number without re-IPLing the system. A re-IPL ensures that a program does not access a cryptographic service with a key that is encrypted under a different master key. If you are certain that no cryptographic applications are still running, you can:

1. Stop CSF
2. Start CSF in COMPAT(NO) mode with a different domain number
3. Stop CSF
4. Start CSF in compatibility or coexistence mode with a different domain number.

END

Specifies the end of a section of parameters for the *fmid* from the **BEGIN(fmid)**. There must be a **BEGIN(fmid)** prior to the END. There must be an END for each **BEGIN(fmid)**. See the description for BEGIN for an example of the usage of the BEGIN and END parameters.

The value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

EXIT(ICSF-name,load-module-name,FAIL(fail-option))

Indicates information about an installation exit.

The ICSF *-name* is the identifier for each exit. [Appendix G, “Resource names for CCA and ICSF entry points,” on page 503](#) lists all the ICSF exit names and explains when ICSF calls each exit. The load module name is the name of the module that contains the exit. The name can be any valid name your installation chooses.

Using the FAIL keyword of the EXIT statement, you specify the action ICSF, the KGUP, or the PCF conversion program takes if the exit ends abnormally. The fail action that you specify applies to subsequent calls of the exit. If an exit ends abnormally, ICSF takes a system dump. The exit is protected with an ESTAE or the ICSF service functional recovery routine (FRR).

In general, you can specify one of these values for a fail option:

NONE

No action is taken. The exit can be called again and will end abnormally again.

EXIT

The exit is no longer available to be called again.

SERVICE

The service or program that called the exit is no longer available to be called again.

ICSF

ICSF or the key generator utility program or the PCF conversion program is ended, depending on the exit.

Some fail options are not valid for a specific exit. If you specify a fail option that is not valid, ICSF uses the next valid fail option. For example, if SERVICE is not a valid fail option for an exit, ICSF uses the EXIT option. EXIT is responsible for logging in SMF the results of any authorization checks that are made.

See Chapter 5, “Installation exits,” on page 165 for a detailed description of each ICSF exit, including the valid fail options.

Note: z/OS no longer ships IBM-supplied security exit routines; the security exit points remain. Users of z/OS should use Security Server (RACF) or an equivalent product to obtain access checking of services and keys. ICSF no longer needs these exit routines.

FIPSMODE(140-3,INDICATE,FAIL(fail-option) or 140-3,ENFORCE,FAIL(fail-option) or 140-3,HYBRID,FAIL(fail-option) or NO,FAIL(fail-option) or YES,FAIL(fail-option) or COMPAT,FAIL(fail-option))

Indicates the level of ICSF FIPS 140 algorithm approval checking requested for z/OS PKCS #11 clear key services. With this option, installations can configure z/OS PKCS #11 clear key services to only allow the use of cryptographic algorithms (including key sizes) that meet FIPS 140 requirements.

The FIPSMODE option applies to requests that use PKCS #11 clear keys and does not apply to ICSF PKCS #11 secure key cryptographic requests processed by the Enterprise PKCS #11 cryptographic coprocessor (EP11). Secure key EP11 requests cannot be processed in FIPS 140-3 INDICATE and ENFORCE modes. Secure key EP11 requests are allowed in all other FIPSMODEs.

The FIPSMODE option has no effect on the compliance mode setting of the Enterprise PKCS #11 cryptographic coprocessor (EP11). For more information about compliance modes and the EP11 coprocessor, see 'Standard compliance modes' under 'Enterprise PKCS #11 coprocessors' in 'The C API' of *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*.

The FIPSMODE option does not have any effect on the compliance mode of cryptographic coprocessors configured as CEXnC coprocessors. When the FIPSMODE is set to enforce PKCS #11 Services FIPS restrictions, ICSF applications using PKCS #11 services will not be directed to coprocessors except for the cases of non-PSS RSA signature services and Diffie-Hilman shared secret computation performed on coprocessors configured as a crypto express accelerator (CEXnA).

For more information, see:

- “Requiring signature verification for ICSF module CSFINPV2” on page 31.
- ‘Operating in compliance with FIPS 140’ in *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for information on the PKCS #11 clear key FIPS 140-3 approved algorithm indicator and note that secure key EP11 requests do not return this indicator.
- ‘FIPS 140-3 mode programming’, under 'Environment', in 'The C API', of *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for information on how to request FIPS 140-3 return/reason code format from ICSF’s PKCS #11 C API.

ICSF supports FIPSMODE options for FIPS 140-3 and FIPS 140-2. The default FIPSMODE value is FIPSMODE(NO,FAIL(NO)) or FIPS 140-2 on-demand mode.

FIPS 140-3 modes: FIPS 140-3 modes make use of return code 0 with non-zero reason codes for clear key requests. See ‘Operating in compliance with FIPS 140’ in *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for details on reason code values and how they are used to indicate if an approved algorithm was used. Installations should confirm that all applications using PKCS #11 services will function as required before updating the FIPSMODE to a FIPS 140-3 mode. See ‘Using SMF reporting for FIPS planning’ in *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* to see how SMF recording can be used to determine how PKCS #11 services are being used.

140-3,ENFORCE

Indicates ICSF PKCS #11 clear key services will operate in *FIPS 140-3 ENFORCE mode*. Requests that do not meet ICSF FIPS 140-3 algorithm approval checking will not be processed. Requests

that are processed successfully will return with return code 0 and a non-zero reason code indicating that FIPS 140-3 requirements were met. ENFORCE mode is an option to use when all applications meet ICSF FIPS 140-3 requirements.

140-3,INDICATE

Indicates ICSF PKCS #11 clear key services will operate in *FIPS 140-3 INDICATE mode*. Cryptographic requests will be processed when, at least, ICSF FIPS 140-2 algorithm approval checking succeeds. Requests that are processed successfully will return with return code 0 and a non-zero reason code indicating which one of FIPS 140-2 or FIPS 140-3 requirements were met. INDICATE mode is an option to use when all applications meet at least ICSF FIPS 140-2 requirements,

140-3,HYBRID

Indicates ICSF PKCS #11 clear key services will operate in *FIPS 140-3 HYBRID mode*. In this mode, by default, applications operate in FIPS 140-3 INDICATE mode. In HYBRID mode, installations can request that selected applications operate in FIPS 140-2 COMPAT mode. HYBRID mode is an option to use when not all applications can tolerate the non-zero reason code returned in ENFORCE and INDICATE modes or when some applications cannot meet ICSF FIPS 140-3 requirements.

To make an application operate in FIPS 140-2 COMPAT mode, create a resource profile in the CRYPTOZ class with the naming convention *USEFIPS140-2.token-name*, where *token-name* is the token name of the key object to be used by the application. Grant READ access to the profile that the user ID (or user ID of the task) that the request will run under.

When the user ID has READ access to the *USEFIPS140-2.token-name* profile and the key object token name is the same as *token-name*, the request is processed in FIPS 140-2 COMPAT mode.

Applications running under COMPAT mode rules must meet FIPS 140-2 requirements. An application running under COMPAT mode can then also optionally be exempt from any FIPS enforcement. See the explanation for 'COMPAT' below.

For a detailed explanation of setting up CRYPTOZ profiles for 140-3, HYBRID and COMPAT modes, see 'Requiring FIPS 140-2 algorithm checking from select z/OS PKCS #11 applications' and 'Requiring FIPS 140-3 algorithm checking from select z/OS PKCS #11 applications' in [z/OS Cryptographic Services ICSF Writing PKCS #11 Applications](#).

FIPS 140-2 modes: In FIPS 140-2 modes, all successful requests return with return code zero and reason code 0.

YES

Indicates ICSF PKCS #11 services will operate in *FIPS 140-2 ENFORCE mode*. Any application using PKCS #11 services must use only those algorithms allowed by ICSF FIPS 140-2 algorithm checking. Applications not allowed by ICSF FIPS 140-2 algorithm checking will fail.

COMPAT

Indicates ICSF PKCS #11 services will operate in *FIPS 140-2 COMPAT mode*. By default, applications running in FIPS 140-2 COMPAT mode must meet the same FIPS 140-2 algorithm restrictions as YES mode. Optionally, an application can be made exempt from FIPS 140-2 algorithm checking.

To make an application exempt from FIPS 140-2 checking, create a resource profile in the CRYPTOZ class with the naming convention *FIPSEXEMPT.token-name*, where *token-name* is the token name of the key object to be used by the application. Grant READ access to the profile that the user ID (or user ID of the task) that the request will run under. The *CKA_IBM_FIPS140* attribute must be OFF in the key object for the application to be exempt from checking.

The application will be exempt from ICSF FIPS 140-2 algorithm checking when all of the following are true:

- The *CKA_IBM_FIPS140* attribute in the key object is OFF.
- The key object token name is the same as the *token-name* in a *FIPSEXEMPT.token-name* resource profile.

- The user ID (or user ID of the task) that the application runs under has READ access to the FIPSEXEMPT.token-name resource profile.

The application will not be exempt from ICSF FIPS 140-2 algorithm checking when any of the following are true:

- The CKA_IBM_FIPS140 attribute in the key object is ON.
- No FIPSEXEMPT.token-name profile exists for key object token name or the profile exists, but the user ID (or user ID of the task) that the application runs under has an access of NONE to profile.

See 'Requiring FIPS 140-2 algorithm checking from select z/OS PKCS #11 applications' and 'Requiring FIPS 140-3 algorithm checking from select z/OS PKCS #11 applications' in [z/OS Cryptographic Services ICSF Writing PKCS #11 Applications](#) for a detailed explanation of setting up CRYPTOZ profiles for 140-3, HYBRID and COMPAT modes.

NO

Indicates ICSF PKCS #11 services will operate in *FIPS 140-2 on-demand* mode. This is the default mode. In this mode, applications are not required to meet the requirements of ICSF FIPS 140-2 algorithm checking unless the application explicitly requests that checking is done. FIPS 140-2 applications can explicitly request ICSF FIPS 140-2 checking by using a key object with the CKA_IBM_FIPS140 attribute set to ON.

Fail-option

ICSF will perform a series of cryptographic known answer tests at ICSF initialization as required by the FIPS 140 standards. If any of these initialization tests should fail, the action the ICSF initialization process takes will depend on the *fail-option* specified.

YES

Indicates ICSF is to terminate abnormally if there is a failure in any ICSF initialization known answer tests.

NO

Indicates ICSF initialization processing is to continue even if there is a failure in any of the initialization known answer tests. However, PKCS #11 support will be limited or nonexistent.

HDRDATE(YES or NO)

This keyword is no longer supported, but is tolerated.

KDSREFDAYS(n)

Specifies, in days, how often a record should be written for a reference date/time change. A key is referenced when it is used to perform a cryptographic operation. If a key is referenced ICSF will check the date and time the key was referenced previous to the current reference. If the number of days between the current date and time and the date and time the key was last referenced is greater than or equal to the number of days specified in the KDSREFDAYS installation option then the key reference date/time in the KDS will be updated to the current date and time. Otherwise the reference date/time will remain the same. Note, in this context days are 24 hour periods not necessarily beginning or ending at midnight.

For example: If KDSREFDAYS(7) was specified and a key was referenced on Monday, January 1st at 8 AM, and the reference date/time for the key was updated at that time, then any key reference before Monday, January 8th at 8 AM (7 days) will not update the reference date/time in the key record. If the key is referenced again at 7:50 AM on Monday, January 8th, the reference date/time for the key in the KDS will remain January 1st at 8 AM because fewer than seven days have passed. The reference date/time will not be updated until the next time the key is used again Monday, January 8th at 8 AM or after.

KDSREFDAYS applies to all KDS that are in the format that supports key reference tracking. In an environment of mixed KDS formats, where some support reference date tracking and some do not (for example, the CKDS supports reference date tracking, but the PKDS does not) key references will not be tracked for keys in a KDS does not support it, regardless on the value of KDSREFDAYS, until that KDS is updated to the new format. In a SYSPLEX, all systems must be started with the same value of KDSREFDAYS to ensure proper tracking of reference date/times.

KDSREFDAYS(0) means that ICSF will not keep track of key reference dates. The default is KDSREFDAYS(1). The maximum value allowed is KDSREFDAYS(30).

Note: Updates to records using the Key Generator Utility Program (KGUP) are not subject to the value specified in the KDSREFDAYS option. All updates made via KGUP will update the reference date/time if the CKDS is in a format that supports reference date tracking (KDSR).

The value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

KEYARCHMSG(YES or NO)

Controls whether a joblog message is issued when an application successfully references a key data set record that has been archived. The message is only issued for the first successful reference of a record. The results of the service request is not affected by this control. The default is NO.

Value

Indication

YES

ICSF issues a message the first time an archived record is referenced by an application.

NO

ICSF does not issue a message when an archived record is referenced by an application.

The value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

KEYAUTH(YES or NO)

This keyword is no longer supported, but is tolerated.

MASTERKCVLEN(2 or 3 or 4 or 5 or 6 or ALL)

Defines the number of hexadecimal digits to display on the ICSF Coprocessor Hardware Status panel (CSFCMP40) for the verification and hash patterns for the master keys. The patterns are also referred to as key check values. When an integer value is specified, that number of digits will be displayed. When ALL is specified, all the digits will be displayed.

Defines the number of hexadecimal digits recorded in the SMF82KV field for the SMF 82, subtype 7 record. MASTERKCVLEN also limits the number of hexadecimal digits displayed when the D ICSF,MKVPS command is issued. Regardless of the MASTERKCVLEN value, the maximum number of hexadecimal digits displayed when the D ICSF,MKVPS command is issued is six.

The default is ALL.

This option can be used for compliance with the ISO11568 and other standards for the display of the key check values for master keys.

The value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

MAXLEN(n)

Defines the maximum length of characters in a text string, including any necessary padding, for some callable service requests. For example, this option defines the maximum length of the text the encipher service encrypts for each call. Specify *n* as a decimal value from 1024 through 2147483647. If you do not specify the MAXLEN option, the default value is MAXLEN(65535).

The MAXLEN parameter may still be specified in the options data set, but only the maximum value limit will be enforced (2147483647). If a value greater than this is specified, an error will result and ICSF will not start.

Note: MAXLEN is no longer displayed on the Installation Option Display panel.

MAXSESSOBJECTS(n)

Defines the maximum number of PKCS #11 session objects and states an unauthorized (problem state, non-system key) application may own at any one time. Specify *n* as a decimal value from 1024 through 2147483647. If you do not specify the MAXSESSOBJECTS option, the default value is MAXSESSOBJECTS(65535).

The value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

MAXSLOWOPS(n)

Specifies the maximum number of slow operations (such as RSA key pair generation) which will be allowed to run simultaneously on each CCA coprocessor. Specify *n* as a decimal value between 1 and 7, inclusive. It is recommended to specify this option only when necessary to improve performance for these slower operations and only to be increased by one before verifying the impact on fast operation throughput.

If MAXSLOWOPS is not specified in the options data set, ICSF will default to a value of 1.

The value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

PKDSCACHE

This keyword is no longer supported, but is tolerated.

PKDSN(data-set-name)

Specifies the PKDS name the system uses to start ICSF. Whenever you restart ICSF, the PKDS named in the PKDSN option becomes the active PKDS. (At first-time startup, you should specify the name of an empty VSAM data set you created to use as the PKDS.)

If you do not specify this keyword, you will not be able to use secure CCA asymmetric keys or use ICSF to manage CCA asymmetric keys. ICSF must be restarted in order to switch between having a PKDS and not having a PKDS.

See [“Steps to create the installation options data set”](#) on page 24 for the data set naming format requirements.

REASONCODES(ICSF or TSS)

Specifies which set of reason codes are to be returned from callable services. If you do not specify the REASONCODES option, the default of REASONCODES(ICSF) is used. If you specify REASONCODES(TSS), reason codes used by the IBM 4765 PCIE, IBM 4767 PCIE, and IBM 4764 PCI-X cryptographic coprocessors will be returned. If there is a 1-to-1 mapping, the codes will be converted.

If you specified REASONCODES(ICSF) and your service was processed on a CCA coprocessor, a cryptographic coprocessor reason code may be returned if there is no corresponding ICSF reason code.

RNGCACHE(YES or NO)

Indicates whether ICSF should maintain a cache of random numbers to be used by services that require them. When YES is specified for this option, a noticeable performance improvement may be realized by workloads requesting a significant amount of random data.

If you do not specify the RNGCACHE option, the default value is RNGCACHE(YES).

Value

Indication

YES

ICSF maintains a random number cache.

NO

ICSF does not maintain a random number cache.

The value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

SERVICE(service-number,load-module-name,FAIL(fail-option))

Indicates information about an installation-defined service.

ICSF allows an installation to define its own service similar to an ICSF callable service. The *service-number* specifies a number that identifies the service to ICSF. The valid service numbers are 1 through 32767, inclusive. This set of service numbers is valid for both installation-defined services and UDX services. The service number of an installation-defined service must not be the same as the service

number of a UDX service. The *load-module-name* is the name of the module that contains the service. During ICSF startup, ICSF loads this module and binds it to the *service-number* you specified.

The *fail-option* is YES or NO, indicating the action ICSF should take if loading the service ends abnormally.

YES

Specifies that ICSF ends abnormally if your service cannot be loaded.

NO

Specifies that ICSF continues to start if your service cannot be loaded.

If the service itself ends abnormally, ICSF does not end, but takes a system dump instead. The ICSF service functional recovery routine (FRR) protects the service.

See [Chapter 6, “Installation-defined Callable Services,” on page 213](#) for a description of how to write and run an installation-defined callable service.

SERVICELIBS(YES or NO)

Indicates whether ICSF will be loaded using service data sets.

YES

Specifies that ICSF will be loaded using service data sets.

NO

Specifies that ICSF will not be loaded using service data sets and parameters SERVSCSFMOD0 and SERVIEALNKE are ignored.

If the SERVICELIBS option is not specified, the default is SERVICELIBS(NO).

For more information about utilizing service libraries, see [“Dynamic service update” on page 140](#).

SERVSCSFMOD0(dsn[,volser])

Specifies the name of the service data set to be used in a dynamic service update for SCSFMOD0. *volser* is optional. Must be specified in conjunction with SERVICELIBS(YES).

dsn

The data set name of the service data set.

volser

The volume of the service data set.

If the SERVSCSFMOD0 option is not specified, ICSF is initialized using LNKST.

SERVIEALNKE(dsn[,volser])

Specifies the name of the service data set to be used in a dynamic service update for SIEALNKE. *volser* is optional. Must be specified in conjunction with SERVICELIBS(YES).

dsn

The data set name of the service data set.

volser

The volume of the service data set.

If the SERVIEALNKE option is not specified, ICSF is initialized using LNKST.

Example:

```
SERVICELIBS(YES)
SERVSCSFMOD0(CSF.SCSFMOD0,VOL177)
SERVIEALNKE(SYS1.SIEALNKE,CSFDR1)
```

SSM(YES or NO)

Specifies whether or not an installation can enable special secure mode (SSM) while running ICSF. SSM lowers the security of your system to let you enter clear keys and generate clear PINs. You must enable SSM for KGUP to permit generation or entry of clear keys and to enable the secure key import, secure key import2, multiple secure key import, or clear pin generate callable services.

YES

Indicates that you can enable the SSM.

NO

Indicates that you cannot enable the SSM.

If you do not specify the SSM option, the default value is SSM(NO).

The SSM option can be changed from NO to YES while ICSF is running by defining the CSF.SSM.ENABLE SAF profile within the XFACILIT resource class. To revert to your startup option, delete the CSF.SSM.ENABLE profile. The XFACILIT class must be refreshed after each change for it to take effect.

Note: When using the SAF profiles to set the SSM, all ICSF instances sharing the SAF database will be affected.

The value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

When the CSF.SSM.ENABLE SAF profile is defined within the XFACILIT resource class, attempts to update the SSM option using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6) will be ignored. The SSM option value will be saved and used should the CSF.SSM.ENABLE SAF profile ever be deleted.

STATS(value1[,...,value3])

Enables usage tracking for various cryptographic statistics. Keywords may be combined to track multiple statistics.

ENG

Enables usage tracking of cryptographic engines. Supports Crypto Express cards, CPACF, and software.

SRV

Enables usage tracking of cryptographic services. Supports ICSF callable services and UDXes only.

ALG

Enables usage tracking of cryptographic algorithms. Supports cryptographic algorithms that are referenced in cryptographic operations. Limited support for key generation, key derivation, and key import.

For more information on the cryptographic utilization statistics monitoring, see [*z/OS Cryptographic Services ICSF Administrator's Guide*](#).

STATSFILTERS(value)

Filters the criteria that is used to aggregate crypto usage statistics when STATS is enabled. Excluding this option means that ICSF uses all available criteria (that is, HOME job id, HOME job name, SECONDARY job name, HOME user id, task level user id, and ASID) to aggregate the crypto usage statistics.

NOTKUSERID

Excludes the task level user id from the stats aggregation criteria. Enable this option in environments that have a high volume of operations that are running under task level user ids. This option reduces the number of SMF records written.

For more information on the cryptographic utilization statistics monitoring, see [*z/OS Cryptographic Services ICSF Administrator's Guide*](#).

SYSPLEXCKDS(YES or NO,FAIL(fail-option))**SYSPLEXCKDS(YES,FAIL(fail-option))**

ICSF will join the ICSF sysplex group SYSICSF and this system will participate in sysplex-wide consistency for CKDS data.

SYSplexCKDS(YES,FAIL(YES))

Indicates ICSF initialization will end abnormally if the ICSF cross-system services environment cannot be established during ICSF initialization due to a failure creating the CKDS latch set or a failure to join the ICSF sysplex group.

SYSplexCKDS(YES,FAIL(NO))

Indicates ICSF initialization processing will continue even if the request to join the ICSF sysplex group fails. The system will not be notified of updates to the CKDS by other members of the ICSF sysplex group. A value of either FAIL(YES) or FAIL(NO) will be ignored with SYSplexCKDS(NO,...).

SYSplexCKDS(NO,FAIL(fail-option))

CKDS update processing proceeds as it does today (i.e. no Cross-System Services task will be initialized, nor will any XCF signalling be performed when an update to a CKDS record occurs).

If you do not specify the SYSplexCKDS option, the default value is SYSplexCKDS(NO,FAIL(NO)).

SYSplexPKDS(YES or NO,FAIL(fail-option))

ICSF will join the ICSF sysplex group SYSICSF and this system will participate in sysplex-wide consistency for PKDS data.

SYSplexPKDS(YES,FAIL(fail-option))

ICSF will join the ICSF sysplex group SYSICSFP and this system will participate in sysplex-wide consistency for PKDS data.

SYSplexPKDS(YES,FAIL(YES))

Indicates ICSF initialization will fail to join the sysplex if the ICSF cross-system services environment cannot be established during ICSF initialization due to a failure creating the PKDS latch set or a failure to join the ICSF sysplex group.

SYSplexPKDS(YES,FAIL(NO))

Indicates ICSF initialization processing will continue even if the request to join the ICSF sysplex group fails. The system will not be notified of updates to the PKDS by other members of the ICSF sysplex group. A value of either FAIL(YES) or FAIL(NO) will be ignored with SYSplexPKDS(NO,...).

SYSplexPKDS(NO,FAIL(fail-option))

PKDS update processing proceeds without trying to join the ICSF sysplex group.

If you do not specify the **SYSplexPKDS** option, the default value is **SYSplexPKDS(NO,FAIL(NO))**.

SYSplexTKDS(YES or NO,FAIL(fail-option))

ICSF will join the ICSF sysplex group SYSICSF and this system will participate in sysplex-wide consistency for TKDS data.

Note: TKDSN needs to be specified for this to work. See **TKDSN(data-set-name)**.

SYSplexTKDS(NO,FAIL(fail-option))

Indicates no XCF signalling will be performed when an update to a TKDS record occurs.

SYSplexTKDS(YES,FAIL(fail-option))

Indicates the system will be notified of updates made to the TKDS by other members of the sysplex who have also specified SYSplexTKDS(YES,FAIL(fail-option)).

SYSplexTKDS(YES,FAIL(YES))

Indicates ICSF will terminate abnormally if there is a failure creating the TKDS latch set.

SYSplexTKDS(YES,FAIL(NO))

Indicates ICSF initialization processing will continue even if the request to join the ICSF sysplex group fails. This system will not be notified of updates to the TKDS by other members of the ICSF sysplex group.

If you do not specify the SYSplexTKDS option, the default value is SYSplexTKDS(NO,FAIL(NO)) is the default.

TKDSN(data-set-name)

The name of an existing TKDS or an empty VSAM data set to be used as the TKDS. To enable applications to create and use persistent PKCS #11 tokens and objects that use the PKCS #11 services, this option must be specified.

See [“Steps to create the installation options data set”](#) on page 24 for the data set naming format requirements.

TRACEENTRY(n)

This keyword is no longer supported, but is tolerated.

TRACKCLASSUSAGE(class1[,class2])

Indicates information about tracking key usage by classes of cryptographic operations. Reference date tracking must be enabled. See the KDSREFDAYS parameter description.

ICSF tracks the usage of keys in the common record format CKDS, PKDS, and TKDS. The usage is recorded in the metadata for the key record as the last date any service in a class was called. The reference period is the same as the reference date tracking. See the KDSREFDAYS parameter description.

The supported cryptographic classes are:

DATADEC

Symmetric keys data decryption operations.

When a symmetric key is referenced for these services, the date is recorded.

- Decipher (CSNBDEC, CSNEDEC, CSNBDEC1, and CSNEDEC1).
- Ciphertext Translate2 (CSNBCTT2, CSNECTT2, CSNBCTT3, and CSNECTT3):
 - Inbound key identifier.
- Symmetric Algorithm Decipher (CSNBSAD, CSNESAD, CSNBSAD1, and CSNESAD1).
- Symmetric Key Decipher (CSNBSYD, CSNESYD, CSNBSYD1, and CSNESYD1).
- Field Level Decipher (CSNBFLD and CSNEFLD).
- FPE Decipher (CSNBFPED and CSNEFPED).
- FPE Translate (CSNBFPET and CSNEFPET):
 - Inbound key identifier.
- Format Preserving Algorithms Decipher (CSNBFFXD and CSNEFFXD).
- Format Preserving Algorithms Translate (CSNBFFXT and CSNEFFXT):
 - Inbound key identifier.
- PKCS #11 Secret Key Decrypt (CSFPSKD and CSFPSKD6).
- PKCS #11 Secret Key Reencrypt (CSFPSKR and CSFPSKR6):
 - Inbound key identifier.

DATAENC

Symmetric keys data encryption operations.

When a symmetric key is referenced for these services, the date is recorded.

- Encipher (CSNBENC, CSNEENC, CSNBENC1, and CSNEENC1).
- Ciphertext Translate2 (CSNBCTT2, CSNECTT2, CSNBCTT3, and CSNECTT3):
 - Outbound key identifier.
- Symmetric Algorithm Encipher (CSNBSAE, CSNESAE, CSNBSAE1, and CSNESAE1).
- Symmetric Key Encipher (CSNBSYE, CSNESYE, CSNBSYE1, and CSNESYE1).
- Field Level Encipher (CSNBFLE and CSNEFLE).
- FPE Encipher (CSNBFPEE and CSNEFPEE).

- FPE Translate (CSNBFPET and CSNEFPET):
 - Outbound key identifier.
- Format Preserving Algorithms Encipher (CSNBFFXE and CSNEFFXE).
- Format Preserving Algorithms Translate (CSNBFFXT and CSNEFFXT):
 - Outbound key identifier.
- PKCS #11 Secret key encrypt (CSFPSKE and CSFPSKE6).
- PKCS #11 Secret Key Reencrypt (CSFPSKR and CSFPSKR6):
 - Outbound key identifier.

UDX(UDX-id,service-number,load-module-name,'comment_text',FAIL(fail-option))

ICSF allows the development of User Defined Extensions for the coprocessors. The *UDX-id* is supplied to the installation when the UDX is developed. The *service-number* specifies a number that identifies the service to ICSF. The valid service numbers are 1 to 32767, inclusive. This set of service numbers is valid for both installation-defined services and UDX services. The service number of a UDX service must not be the same as the service number of an installation-defined service. The *load-module-name* is the name of the module that contains this service. During ICSF startup, ICSF loads this module and binds it to the service-number that was specified. A *comment* may be specified. The positional parameter is required. The comment consists of up to 40 EBCDIC characters, and may include imbedded blank characters. The comment text is enclosed by single quotes. If no comment text is desired, two contiguous single quotes should be specified.

The *fail-option* is YES or NO, indicating the action ICSF should take if loading the service ends abnormally. If the service itself ends abnormally, ICSF does not end, but takes a system dump instead.

YES

Specifies that ICSF ends abnormally if your service cannot be loaded.

NO

Specifies that ICSF continues to start if your service cannot be loaded.

The User Defined Extension (UDX) is responsible for logging in SMF the results of any authorization checks that were made.

USERPARM(value)

Specifies an 8-byte field for installation use. The Installation Option Display panel displays this value, which is stored in the Cryptographic Communication Vector Table (CCVT) in the *CCVT_USERPARM* field. An application program or installation exit can examine this field and use it to set system environment information. The default is eight blanks.

The value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

WAITLIST(data_set_name)

This optional parameter can be used if you have ICSF with CICS installed. It specifies a customer modifiable data set will be used to determine names of the services to be placed into the ICSF CICS Wait List. A sample data set is provided by ICSF via member CSFWTL01 of SYS1.SAMPLIB. The sample data set contains the same entries as the default ICSF CICS Wait List (i.e., the data set contains the names of all ICSF callable services which, by default, will be driven through the CICS TRUE). Non-CICS customers will not need to specify the WAITLIST keyword. The WAITLIST option should be added to the Installation Options data set under these conditions.

- CICS customers who do not want to make use of CICS TRUE must either not enable the TRUE or must specify a WAITLIST keyword and point to an empty wait list data set (or specify WAITLIST(DUMMY)) in the Installation Options data set.
- CICS customers who wish to modify the ICSF default CICS Wait List should modify the sample Wait List data set supplied in member CSFWTL01 of SYS1.SAMPLIB. The WAITLIST keyword in the Installation Options Data Set should be set to point to this modified data set.

To ensure maximum performance, any existing CICS applications which invoke any of the ICSF services in the Wait List that were linked with ICSF stubs prior to HCR7770 should be re-linked with the current ICSF stubs. For additional information on the CICS Attachment Facility, see [Appendix C, “CICS-ICSF Attachment Facility,”](#) on page 485.

The value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

Dispatching priority of ICSF

To avoid performance problems, the dispatching priority of ICSF should be set at least as high as that of the highest task using ICSF.

Creating ICSF exits and generic services

You need not code any exits or generic services before using ICSF productively.

Developing callable service exits and generic services requires skill in assembler programming in a cross memory environment. To help with testing, the system programmer might want to use the WTO macro with the LINKAGE=BRANCH keyword to issue console messages while in cross-memory mode. (See [“Service exits”](#) on page 169 for more information.)

Chapter 3. Migration

This topic describes migration considerations.

Your plan for migrating to the new level of ICSF should include information from a variety of sources. These sources of information describe topics such as coexistence, service, hardware and software requirements, installation and migration procedures, and interface changes.



Attention: Although you are migrating to a new release, you should review the information in [Chapter 2, “Installation, initialization, and customization,” on page 9](#); especially review customization steps that may have changed since your last migration.

If this migration also includes a hardware upgrade be sure to have your Master Keys available. Once Migration is complete, the Master Keys may need to be loaded and set. Review [Chapter 2, “Installation, initialization, and customization,” on page 9](#) for information on setting Master Keys.

An IPL is required when installing a new release of ICSF (it is possible for ICSF control blocks like the DACC and CCVT to persist in storage across an ICSF restart).

Consult these documents for information on migration and installation:

z/OS Migration

This publication describes the migration tasks for z/OS at a system and element level.

This publication, which is supplied with your product order, provides information about installing your z/OS system. In addition to specific information about ICSF, this publication contains information about all of the z/OS elements. Consult the z/OS Migration publication for the release of z/OS running on your system.

z/OS Planning for Installation

This publication describes the installation requirements for z/OS at a system and element level. It includes hardware, software, and service requirements for both the driving and target systems. It also describes any coexistence considerations and actions.

Program Directory for Cryptographic Support for z/OS V2R1 - z/OS V2R3

This publication describes the program installation and maintenance requirements. It contains information about the material and procedures associated with the installation of ICSF.

ServerPac Installing Your Order

This is the order-customized, installation publication for using the ServerPac Installation method. Be sure to review 'Appendix A. Product Information', which describes data sets supplied, jobs or procedures that have been completed for you, and product status. IBM may have run jobs or made updates to PARMLIB or other system control data sets. These updates could affect your migration.

Terminology

This topic describes some terms you may need to know as you use this publication.

Migration

Activities that relate to the installation of a new version or release of a program to replace a previous level. Completion of these activities ensures that the applications and resources on your system will function correctly at the new level.

Coexistence

Two or more systems at different levels (for example, software, service or operational levels) that share resources. Coexistence includes the ability of a system to respond in these ways to a new function that was introduced on another system with which it shares resources: ignore a new function, terminate gracefully, support a new function. These are examples of multisystem configurations in which resource sharing can occur:

- A single system running multiple LPARs
- A single processor that is time-sliced to run different levels of the system (for example, during different times of the day)
- Two or more systems running separate processors
- A Parallel Sysplex configuration (also includes a basic sysplex)

Migrating from earlier software releases

These topics describe common activities and considerations that should be considered when you migrate from an earlier release of ICSF to FMID HCR77F0.

Actions to perform before installing ICSF FMID HCR77F0

This topic describes migration actions that you can perform on your current (old) system. You do not need the ICSF FMID HCR77F0 level of code to make these changes, and the changes do not require the ICSF FMID HCR77F0 level of code to run once they are made.

Note: You may have already performed these migration actions if you previously migrated to ICSF FMIDs HCR77D0, HCR77D1, HCR77D2, or HCR77E0.

ICSF: Detect any coprocessor that will not become active when ICSF FMID HCR77A1 or later is started

Description

For ICSF FMIDS HCR7780, HCR7790, and HCR77A0, the activation procedure was designed to maximize the number of active coprocessors by selecting the set of master keys that are available on the majority of coprocessors. A DES master key is no longer required in order for a coprocessor to become active. Instead, any one of four master keys – the DES master key, the AES master key, the RSA master key (which in earlier releases was called the asymmetric master key), or the ECC master key – is enough for a coprocessor to become active. However, because the goal is to select the combination of master keys that will maximize the number of active coprocessors, if a certain master key is not set on all the same coprocessors, that master key support will not be available.

Starting with FMID HCR77A1, the activation procedure now uses the master key verification patterns (MKVP) in the header record of the CKDS and PKDS to determine which coprocessors become active. If the MKVP of a master key is in the CKDS or PKDS, that master key must be loaded and the verification pattern of the current master key register must match the MKVP in the CKDS or PKDS. If all of the MKVPs in the CKDS and PKDS match the current master key registers, the coprocessor will become active. Otherwise, the status is master keys incorrect. This applies to all master keys that the coprocessor supports. When there is a MKVP in the CKDS or PKDS and the coprocessor does not support that master key, it is ignored. When a MKVP is not in the CKDS or PKDS, the master key is ignored.

If there are no MKVPs in the CKDS and PKDS, the coprocessor will be active. If the CKDS is initialized without any MKVPs, the CKDS cannot be used on a system that has cryptographic features installed.

Table 3 on page 58 provides more details about this migration action. Use this information to plan your changes to the system.

Table 3. Information about this migration action

Element or feature:	Cryptographic Services
When change was introduced:	Cryptographic Support for z/OS V1R13 - z/OS V2R1 web deliverable (FMID HCR77A1), which installs on z/OS V1R13 or z/OS V2R1.

Table 3. Information about this migration action (continued)

Applies to migration from:	z/OS V2R1 and z/OS V1R13, both without the Cryptographic Support for z/OS V1R13 - z/OS V2R1 web deliverable (FMID HCR77A1) or a later ICSF web deliverable installed.
Timing:	Before installing FMID HCR77A1 or later ICSF FMIDs.
Is the migration action required?	Yes, if migrating from an ICSF FMID older than HCR77A1 to ICSF FMID HCR77A1 or later and if you are affected by the change in the way master keys are processed to determine which coprocessors become active.
Target system hardware requirements:	None.
Target system software requirements:	None.
Other system (coexistence or fallback) requirements:	None.
Restrictions:	None.
System impacts:	None.
Related IBM Health Checker for z/OS check:	Use check ICSFMIG77A1_COPROCESSOR_ACTIVE to determine which coprocessors will not become active when Cryptographic Support for z/OS V1R13 - z/OS V2R1 Web Deliverable (FMID HCR77A1) is started. This check is delivered in APAR OA42011 available for ICSF FMIDs HCR7770, HCR7780, HCR7790 and HCR77A0.

Steps to take

Run the migration check ICSFMIG77A1_COPROCESSOR_ACTIVE to find any coprocessors that will not become active when you start ICSF FMID HCR77A1 or a later ICSF web deliverable.

Reference information

For more information, see the following reference:

- For information about IBM Health Checker, see [IBM Health Checker for z/OS User's Guide](#).

ICSF: Detect TKDS objects that are too large for the new KDSR record format in ICSF FMID HCR77A1 or later

Description

In ICSF FMID HCR77A1, ICSF added a common key data set record format for CCA key tokens and PKCS #11 tokens and objects. This new record format adds new fields for key utilization and metadata. Because of the size of the new fields, some existing PKCS #11 objects in the TKDS might cause ICSF to fail. If you do not have a Token Data Set (TKDS) with PKDS #11 objects in it, there is no need to run this check.

The problem exists for TKDS object records with large objects. The User data field in the existing record will cause the TKDS not be to loaded if the object size is greater that 32,520 bytes. The TKDSREC_LEN field in the record has the size of the object. If the User data field is not empty and the size of the object is greater than 32,520 bytes, the TKDS cannot be loaded.

Note that ICSF does not provide any interface to modify the User data field in the TKDS object record. A field can be created using IDCAMS. Check the contents of the User data field and determine if the information in the field is valuable. If you want to preserve the data, consider how the information can be stored other than in the object record. The field can only be modified by editing the record. For information about the TKDS object record, see [“Token data set \(TKDS\) format”](#) on page 250. The IBM Health Checker migration check, ICSFMIG77A1_TKDS_OBJECT detects any TKDS object that is too large to allow the TKDS is read into storage during ICSF initialization starting with ICSF FMID HCR77A1. This migration check is available for ICSF FMIDs HCR7770, HR7780, HCR7790, and HCR77A0 through APAR OA42011

Table 4 on page 60 provides more details about this migration action. Use this information to plan your changes to the system.

Table 4. Information about this migration action

Element or feature:	Cryptographic Services
When change was introduced:	Cryptographic Support for z/OS V1R13 – z/OS V2R1 web deliverable (FMID HCR77A1), which installs on z/OS V1R12, z/OS V1R13 or z/OS V2R1.
Applies to migration from:	z/OS V2R1 and z/OS V1R13, both without the Cryptographic Support for z/OS V1R13 - z/OS V2R1 web deliverable (FMID HCR77A1) installed.
Timing:	Before installing FMID HCR77A1 or later ICSF FMIDs.
Is the migration action required?	Yes, if migrating from an ICSF FMID older than HCR77A1 to ICSF FMID HCR77A1 or later and if you affected by the record format changes.
Target system hardware requirements:	None.
Target system software requirements:	None.
Other system (coexistence or fallback) requirements:	None.
Restrictions:	None.
System impacts:	None.
Related IBM Health Checker for z/OS check:	Use the IBM Health Checker migration check ICSFMIG77A1_TKDS_OBJECT to detect any TKDS object with a value in the User data field that is too large to preserve in the User data field of the new format record. This migration check is available for FMIDs HCR7770, HR7780, HCR7790, and HCR77A0 through APAR OA42011.

Steps to take

Run the migration check ICSFMIG77A1_TKDS_OBJECT to detect if TKDS objects are too large for the new record format in FMID HCR77A1.

Note: ICSF does not provide any interface to modify the User data field in the TKDS object record. A flat file can be created using IDCAMS. Check the contents of the User data field and determine if the information in the field is valuable. If you want to preserve the data, consider how the information can be stored other than in the object record. The field can only be modified by editing the record. For information about the TKDS object record, see [“Token data set \(TKDS\) format”](#) on page 250.

Reference information

For more information, see the following references:

- For information about the TKDS object record, see [“Token data set \(TKDS\) format” on page 250](#).
- For information about IBM Health Checker, see [IBM Health Checker for z/OS User's Guide](#).

Actions to perform before the first start of ICSF FMID HCR77F0

This topic describes migration actions that you can perform after you have installed ICSF FMID HCR77F0, but before the first time you start it. These actions might require the ICSF FMID HCR77F0 level of code to be installed, but does not require it to be started.

Note: You may have already performed these migration actions if you previously migrated to ICSF FMIDs HCR77D0, HCR77D1, HCR77D2, or HCR77E0.

ICSF: Deprecated parameters in installation options data set

Description

The ICSF installation options data set parameters COMPENC and PKDSCACHE were deprecated in FMID HCR7751, parameters CKTAUTH, KEYAUTH, and TRACEENTRY were deprecated in FMID HCR77A1, and parameter HDRDATE was deprecated in FMID HCR77B1.

Table 5. Information about this migration action

Element or feature:	Cryptographic Services.
When change was introduced:	ICSF FMID HCR77B1.
Applies to migration from:	All ICSF FMIDs prior to FMID HCR77B1.
Timing:	Before the first start of FMID HCR77B1 or later ICSF FMIDs.
Is the migration action required?	Yes.
Target system hardware requirements:	None.
Target system software requirements:	None.
Other system (coexistence or fallback) requirements:	None.
Restrictions:	None.
System impacts:	None.
Related IBM Health Checker for z/OS check:	None.

Steps to take

Edit the ICSF installation options data set and remove all the deprecated parameters.

Note: ICSF will start with the deprecated parameters in the ICSF installation options data set, but the parameters are ignored and message CSFO0212 is issued for each deprecated parameter.

Reference information

For more information, see [“Customizing ICSF after the first start” on page 34](#).

ICSF: Determine if applications using hash services have archived hashes of long data

Description

Due to service introduced by APAR OA43937, new Hash Method Rule keywords for the ICSF One-Way Hash Generate (CSNBOWH or CSNBOWH1 and CSNEOWH or CSNEOWH1) and PKCS11 One-Way Hash Services (CSFPOWH and CSFPOWH6) will support generation of legacy hash values for verification of archived hash values generated from pre-OA43937 releases of ICSF FMIDs HCR7770 through HCR77A1.

Note: This correction of hashing function does not apply to the case where the sum of the length of hashed text over a series of chained calls exceeds 256 megabytes (or 512, as described further in this topic), but no single invocation supplies an input *text_length* that exceeds 256 (or 512) megabytes. Correct hashes are created when no single invocation of the callable services exceeds the described limit prior to (and after) application of the PTFs for OA43937.

Applications that wish to verify archived hash values created by pre-OA43937 FMID HCR7770 through FMID HCR77A1 releases of ICSF callable services One-Way Hash Generate and PKCS11 One-Way Hash may need to invoke these callable services with new rule array keywords that support the creation of legacy hash values. The hash generated using the new rule array keywords must be used to verify the archived hash values.

The ICSF Callable Services One-Way Hash Generate and PKCS11 One-Way Hash, sign, or verify have corrected the way they create hash values when the length of the text on a single invocation of one of these services supplies an input *text_length* that equals or exceeds 256 megabytes (512 megabytes on z990/z890 or later hardware on FMID HCR7770). The hashing services are corrected with the application of the PTFs for OA43937.

Table 6 on page 62 provides more details about this migration action. Use this information to plan your changes to the system.

Table 6. Information about this migration action

Element or feature:	Cryptographic Services.
When change was introduced:	PTFs for OA43937, which are applicable to: ICSF FMIDs HCR7770 - HCR77A1 (z/OS V1R12 - z/OS V2R1).
Applies to migration from:	ICSF FMIDs HCR7770 - HCR77A1, without the PTF for OA43937.
Timing:	Before the first start of FMID HCR77A1 or later ICSF FMIDs.
Is the migration action required?	Yes, if migrating from an ICSF FMID older than HCR77A1 to ICSF FMID HCR77A1 or later and if you have archived hash values created before the installation of the PTFs for OA43937 which meet the length restrictions described here.
Target system hardware requirements:	None.
Target system software requirements:	None.
Other system (coexistence or fallback) requirements:	None.
Restrictions:	None.
System impacts:	If you do not use the legacy rule array keywords for affected applications, then the application may fail to verify the legacy hashes/signatures.

Table 6. Information about this migration action (continued)

Related IBM Health Checker for z/OS check:	None.
---	-------

Steps to take

Follow these steps:

1. Identify if your application needs to verify archived hash values created by either of the ICSF callable service One-Way Hash Generate (CSNBOWH or CSNBOWH1 and CSNEOWH or CSNEOWH1) or PKCS11 One-Way Hash (CSFPOWH and CSFPOWH6) on releases pre-OA43937 at FMID HCR7770 through FMID HCR77A1. (See the ICSF Application Programmer's Guide documentation changes in this APAR for new ICSF callable service keywords that support the creation of hashes for the verification of archived hash values and the input text length requirements.)
2. If your application has these archived hash values and intends to verify them, then invocations of ICSF callable services One-Way Hash Generate, PKCS11 One-Way Hash, sign, or verify that create hashes for verification of the archived hash values may need to be updated to use the new legacy rule array keywords (ONLY if those archived hash values were created with input text length exceeding the limits described).

Reference information

For more information, see [z/OS Cryptographic Services ICSF Application Programmer's Guide](#) .

Actions to perform after the first start of ICSF FMID HCR77F0

This topic describes migration actions that you can perform only after you have started ICSF FMID HCR77F0. You need ICSF FMID HCR77F0 started to perform these actions.

Note: You may have already performed these migration actions if you previously migrated to ICSF FMIDs HCR77D0, HCR77D1, HCR77D2, or HCR77E0.

ICSF: Enable CSFKEYS PKA ECC token private-key name checking control

Description

To enable CSFKEYS checking of the *private-key name* in ECC private key tokens, the XFACILIT profile CSF.CSFKEYS.ECC.PRIVATEKEYNAME.ENABLE must be defined. In FMID HCR77D2, ICSF has implemented CSFKEYS checking of the *private-key name* in PKA ECC private key tokens. CSFKEYS checking of the *private-key name* in PKA RSA and PKA QSA private key tokens has not changed and will continue to be done.

Table 7 on page 63 provides more details about this migration action. Use this information to plan your changes to the system.

Table 7. Information about this migration action

Element or feature:	Cryptographic Services.
When change was introduced:	z/OS V2R5 (ICSF FMID HCR77D2).
Applies to migration from:	ICSF releases before HCR77D2.
Timing:	After ICSF FMID HCR77D2 has been installed.
Is the migration action required?	No.
Target system hardware requirements:	None.
Target system software requirements:	None.

Table 7. Information about this migration action (continued)

Other system (coexistence or fallback) requirements:	None.
Restrictions:	None.
System impacts:	After CSFKEYS checking of the <i>private-key name</i> in ECC private key tokens has been enabled, the use of ECC private tokens may fail with return code 8 reason code X'3E84' and message ICH408I.
Related IBM Health Checker for z/OS check:	None.

Steps to take

To establish CSFKEYS checking of the *private-key name* in ECC private key tokens:

- Define CSFKEYS profiles as needed for PKA private ECC tokens that have a *private-key name* in the ECC key associated data section. See 'Enabling access authority checking for key tokens' and 'Setting up profiles in the CSFKEYS general resource class' in [z/OS Cryptographic Services ICSF Administrator's Guide](#).
- Enable CSFKEYS checking of the *private-key name* in ECC private key tokens. See 'Key Store Policy' on how to enable the CSF.CSFKEYS.ECC.PRIVATEKEYNAME control in [z/OS Cryptographic Services ICSF Administrator's Guide](#).
- Look for operator message CSFM726I CSFKEYS PKA ECC PRIVATE-KEY NAME CHECKING CONTROL IS ENABLED.

Once CSFKEYS checking of the *private-key name* in ECC private key tokens is established, to resolve problems with use of private ECC tokens failing with return code 8 reason code X'3E84' and message ICH408I identifying the ECC token private-key name as the problem resource:

- As a temporary solution, disable the CSF.CSFKEYS.ECC.PRIVATEKEYNAME ENABLE control with the RACF command:

```
RDELETE XFACILIT CSF.CSFKEYS.ECC.PRIVATEKEYNAME
```

Once CSFKEYS checking of the *private-key name* in ECC private key tokens has been disabled, message CSFM726I CSFKEYS PKA ECC PRIVATE-KEY NAME CHECKING CONTROL IS DISABLED is issued, or

- Define CSFKEYS profiles as needed for PKA private ECC tokens whose *private-key name* is being reported in message ICH408I.

Reference information

For more information, see the ECC private-key section (X'20') table in [z/OS Cryptographic Services ICSF Application Programmer's Guide](#).

ICSF: Accommodate the TRACEENTRY option deprecation

Description

In ICSF FMID HCR77A1 and later, option TRACEENTRY has been deprecated and ICSF CTRACE support has been enhanced to support configurable ICSF CTRACE options from PARMLIB. A default CTICSF00 PARMLIB member is installed in SYS1.PARMLIB. The CTICSF00 PARMLIB member provides default component trace values for ICSF. By default, ICSF CTRACE support will trace with the KdsIO, CardIO, and SysCall filters using a 2M buffer. Configurable options are commented out within this PARMLIB member to provide examples of how to turn them on.

Table 8 on page 65 provides more details about this migration action. Use this information to plan your changes to the system.

Table 8. Information about this migration action

Element or feature:	Cryptographic Services
When change was introduced:	Cryptographic Support for z/OS V1R13 - z/OS V2R1 web deliverable (FMID HCR77A1), which installs only on z/OS V1R13 or z/OS V2R1.
Applies to migration from:	z/OS V2R1 and z/OS V1R13 without the Cryptographic Support for z/OS V1R13 - z/OS V2R1 web deliverable (FMID HCR77A1). Note that when the Cryptographic Support for z/OS V1R13 - z/OS V2R1 Web deliverable (FMID HCR77A1) or later is not installed, this migration item is not applicable.
Timing:	After the first start of ICSF FMID HCR77B0.
Is the migration action required?	Yes, if you have installed the Cryptographic Support for z/OS V1R13 - z/OS V2R1 web deliverable (FMID HCR77A1) or later to handle TKDS with PKDS #11 objects for the new format in FMID HCR77A1 or later.
Target system hardware requirements:	None.
Target system software requirements:	None.
Other system (coexistence or fallback) requirements:	None.
Restrictions:	None.
System impacts:	If the TRACEENTRY option is specified it will be ignored and will produce message CSFO0212 at startup; processing continues.
Related IBM Health Checker for z/OS check:	None.

Steps to take

You can code the new CTRACE option within a BEGIN(HCR77A1) END option pair in a options data set shared between multiple releases of ICSF.

- If you share the installation options data set between FMID HCR77A1 and pre-FMID HCR77A1 systems, you can continue to supply the TRACEENTRY option at the lower-level systems as it is ignored, and processing will continue on the FMID HCR77A1 systems.
- If your installation cannot tolerate the CSFO0212 message that is issued at startup, you need to use different installation option data sets. Note that new CTRACE options will be in effect:
 - Review the default CTRACE options to ensure that they are satisfactory for your system.
 - Make any necessary changes. Use the CTICSF00 PARMLIB to create customized ICSF CTRACE Configuration Data Sets in PARMLIB. You can use the new CTRACE option to specify the customized ICSF CTRACE Configuration Data Set in the ICSF Options Data Set.

For example, you can specify CTRACE (CTICSFxx), where xx is any two characters that were used when copying the default CTICSF00 parmlib member.

Component tracing is active when ICSF starts using the trace options defined in the CTICSFxx PARMLIB member, where 00 is the default. If the CTICSF00 PARMLIB member is incorrect or missing, ICSF CTRACE performs tracing using an internal default set of trace options. The operator

can specify trace options individually on the TRACE CT command or specify the name of a CTICSFxx PARMLIB member containing the desired trace options. Using a PARMLIB member on the TRACE CT command can help minimize operator intervention and avoid syntax or keystroke errors

Reference information

For more information, see the following references:

- [z/OS Cryptographic Services ICSF Administrator's Guide](#)
- For IBM Health Checker, see [IBM Health Checker for z/OS User's Guide](#).

Callable services

The following table summarizes the new and changed callable services for ICSF FMID HCR77F0. For complete reference information on these callable services, see [z/OS Cryptographic Services ICSF Application Programmer's Guide](#).

Note: When an APAR number is listed in the FMID column along with an ICSF FMID, the FMID is the earliest release where the new function is supported.

Table 9. Summary of new and changed ICSF callable services

Callable service	FMID	Description
Digital Signature Generate	HCR77D2 OA66395	Changed: Support for ML-DSA keys. Support for RSA keys with up to an 8192-bit modulus size.
Digital Signature Verify	HCR77D2 OA66395	Changed: Support for ML-DSA keys. Support for RSA keys with up to an 8192-bit modulus size.
ECC Diffie-Hellman	HCR77D2 OA66395	Changed: Support for ML-KEM keys
ICSF Query Algorithm	HCR77D2 OA66395	Changed: Support for ML-DSA, ML-KEM algorithm
PKA Decrypt	HCR77D2 OA66395	Changed: Support for ML-KEM keys. Support for RSA keys with up to an 8192-bit modulus size.
PKA Encrypt	HCR77D2 OA66395	Changed: Support for ML-KEM keys. Support for RSA keys with up to an 8192-bit modulus size.
PKA Key Generate	HCR77D2 OA66395	Changed: Support for ML-DSA, ML-KEM keys. Support for RSA keys with up to an 8192-bit modulus size.
PKA Key Import	HCR77D2 OA66395	Changed: Support for RSA keys with up to an 8192-bit modulus size.
PKA Key Token Build	HCR77D2 OA66395	Changed: Support for ML-DSA, ML-KEM keys
PKA Key Token Change	HCR77D2 OA66395	Changed: Support for ML-DSA, ML-KEM keys. Support for RSA keys with up to an 8192-bit modulus size.
PKA Key Translate	HCR77D2 OA66395	Changed: Support for ML-DSA, ML-KEM keys. Support for RSA keys with up to an 8192-bit modulus size.
PKA Public Key Extract	HCR77D2 OA66395	Changed: Support for ML-DSA, ML-KEM keys
Public Infrastructure Certificate	HCR77D2 OA66395	Changed: Support for RSA keys with up to an 8192-bit modulus size.

Table 9. Summary of new and changed ICSF callable services (continued)

Callable service	FMID	Description
Symmetric Key Export	HCR77D2 OA66395	Changed: Support for RSA keys with up to an 8192-bit modulus size.
Symmetric Key Generate	HCR77D2 OA66395	Changed: Support for RSA keys with up to an 8192-bit modulus size.
Symmetric Key Import	HCR77D2 OA66395	Changed: Support for RSA keys with up to an 8192-bit modulus size.
Symmetric Key Import2	HCR77D2 OA66395	Changed: Support for RSA keys with up to an 8192-bit modulus size.
Symmetric Key Export with Data	HCR77D2 OA66395	Changed: Support for RSA keys with up to an 8192-bit modulus size.
PKCS #11 Derive Key	HCR77D1 OA65205	Changed: Support for Kyber-768 R2.
PKCS #11 Generate Key Pair	HCR77D1 OA65205	Changed: Support for Kyber-768 R2.
PKCS #11 Token Record Create	HCR77D1 OA65205	Changed: Support for Kyber-768 R2.
PKA Decrypt	HCR77D2 OA64883	Changed: Added support to specify the MGF algorithm separately from the Hash algorithm for OAEP v2.1.
PKA Encrypt	HCR77D2 OA64883	Changed: Added support to specify the MGF algorithm separately from the Hash algorithm for OAEP v2.1.
Multi-MAC Scheme	HCR77D1 OA64883	New: Complete the M of N MAC Scheme.
PKA Decrypt	HCR77D1 OA64883	Changed: Added support for CRYSTALS-Kyber 768 Round 2 and Round 3 and CRYSTALS-Kyber 1024 Round 3.
PKA Encrypt	HCR77D1 OA64883	Changed: Added support for CRYSTALS-Kyber 768 Round 2 and Round 3 and CRYSTALS-Kyber 1024 Round 3.
PKA Key Translate	HCR77D1 OA64883	Changed: Added support for CRYSTALS-Kyber 768 Round 2 and Round 3 and CRYSTALS-Kyber 1024 Round 3.
PKA Key Token Change	HCR77D1 OA64883	Changed: Added support for CRYSTALS-Kyber 768 Round 2 and Round 3 and CRYSTALS-Kyber 1024 Round 3.
PKA Key Generate	HCR77D1 OA64883	Changed: Added support for CRYSTALS-Kyber 768 Round 2 and Round 3 and CRYSTALS-Kyber 1024 Round 3.
PKA Key Import	HCR77D1 OA64883	Changed: Added support for CRYSTALS-Kyber 768 Round 2 and Round 3 and CRYSTALS-Kyber 1024 Round 3.
ECC Diffie-Hellman	HCR77D1 OA64883	Changed: Added support for CRYSTALS-Kyber 768 Round 2 and Round 3 and CRYSTALS-Kyber 1024 Round 3.
TR-31 Create	HCR77D1 OA61978	New: Generate TR-31 key blocks or build TR-31 skeleton key blocks.
Authentication Parameter Generate	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Cipher Text Translate2	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.

Table 9. Summary of new and changed ICSF callable services (continued)

Callable service	FMID	Description
CKDS Key Record Create2	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
CKDS Key Record Read2	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
CKDS Key Record Write2	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Clear PIN Encrypt	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Clear PIN Generate	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Data Key Export	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Data Key Import	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Decipher	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Digital Signature Generate	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Digital Signature Verify	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Diversified Key Generate	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Diversified Key Generate2	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
ECC Diffie-Hellman	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Encipher	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Encrypted PIN Generate	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Encrypted PIN Translate2	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Encrypted PIN Translate Enhanced	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Encrypted PIN Verify	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Encrypted PIN Verify2	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Field Level Decipher	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Field Level Encipher	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.

Table 9. Summary of new and changed ICSF callable services (continued)

Callable service	FMID	Description
Format Preserving Algorithms Decipher	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Format Preserving Algorithms Encipher	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Format Preserving Algorithms Translate	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
FPE Decipher	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
FPE Encipher	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
FPE Translate	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
HMAC Generate	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
HMAC Verify	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Key Export	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Key Generate	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Key Generate2	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Key Import	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Key Part Import2	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Key Test2	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Key Translate2	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
MAC Generate	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
MAC Generate2	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
MAC Verify	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
MAC Verify2	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Multiple Secure Key Import	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
PIN Change/Unblock	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.

Table 9. Summary of new and changed ICSF callable services (continued)

Callable service	FMID	Description
PKA Decrypt	HCR77D1 OA61978	Changed: Added support for PKOAE2.
PKA Encrypt	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks and PKOAE2.
PKA Key Generate	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
PKA Key Import	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
PKA Key Translate	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Random Number Generate Long	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Recover PIN from Offset	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Restrict Key Attribute	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Secure Key Import2	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Secure Messaging for Keys	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Secure Messaging for PINs	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Symmetric Algorithm Decipher	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Symmetric Algorithm Encipher	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Symmetric Key Decipher	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Symmetric Key Encipher	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Symmetric Key Export	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Symmetric Key Export with Data	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Symmetric Key Generate	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Symmetric Key Import2	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
TR-31 Translate	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks for transport keys.
TR-31 Import	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks for transport keys.

Table 9. Summary of new and changed ICSF callable services (continued)

Callable service	FMID	Description
TR-34 Key Distribution	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
TR-34 Key Receive	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Transaction Validation	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
Unique Key Derive	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
VISA CVV Service Generate	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
VISA CVV Service Verify	HCR77D1 OA61978	Changed: Added support for TR-31 key blocks.
TR-31 Import	HCR77D0 OA63531	Changed: Added support for import of TR-31 key blocks with Mode of use "N" with all key block version IDs.
TR-31 Translate	HCR77D0 OA63531	Changed: <ul style="list-style-type: none"> • Added support for export to TR-31 key blocks with Mode of use "N" with all key block version IDs. • Added support for export to TR-31 K0 key blocks with Mode of use "B".
Control Vector Generate	HCR77D1 OA61609	Changed: New key type DATA-CV to build control vector with DATA control vector.
ECC Diffie-Hellman	HCR77D1 OA61609	Changed: Added support for the Hybrid QSA Key Exchange Scheme.
Key Token Build	HCR77D1 OA61609	Changed: New key type DATA-CV to build key token with DATA control vector.
PKA Decrypt	HCR77D1 OA61609	Changed: Added support for CRYSTALS-Kyber algorithm.
PKA Encrypt	HCR77D1 OA61609	Changed: Added support for CRYSTALS-Kyber algorithm.
PKA Key Translate	HCR77D1 OA61609	Changed: Added support for CRYSTALS-Dilithium, CRYSTALS-Kyber, and ECC export to AESKW format.
PKCS #11 Derive Key	HCR77D1 OA61609	Changed: Added support for key derivation using CRYSTALS-Kyber keys.
PKCS #11 Generate Key Pair	HCR77D1 OA61609	Changed: Added support for generation of CRYSTALS-Kyber key pairs and Dilithium 6,5 R3 and Dilithium 8,7 R2 and R3 key objects.
PKCS #11 Get Attribute Value	HCR77D1 OA61609	Changed: Added support for CRYSTALS-Kyber key objects and Dilithium 6,5 R3 and Dilithium 8,7 R2 and R3 key objects.
PKCS #11 One-Way Hash, Sign, or Verify	HCR77D1 OA61609	Changed: Added support for Dilithium 6,5 R3 and Dilithium 8,7 R2 and R3 key objects.
PKCS #11 Set Attribute Value	HCR77D1 OA61609	Changed: Added support for CRYSTALS-Kyber key objects and Dilithium 6,5 R3 and Dilithium 8,7 R2 and R3 key objects.

Table 9. Summary of new and changed ICSF callable services (continued)

Callable service	FMID	Description
PKCS #11 Token Record Create	HCR77D1 OA61609	Changed: Added support for CRYSTALS-Kyber key objects and Dilithium 6,5 R3 and Dilithium 8,7 R2 and R3 key objects.
Digital Signature Generate	HCR77D1 OA61253	Changed: Support for Schnorr digital signature algorithm.
Digital Signature Verify	HCR77D1 OA61253	Changed: Support for Schnorr digital signature algorithm.
Diversified Key Generate	HCR77D1 OA61253	Changed: Support for AusPayNet key derivation algorithms.
Diversified Key Generate2	HCR77D1 OA61253	Changed: Changed derivation data length restrictions for KDFFM-DK. Added initialization vector support.
Diversify Directed Key	HCR77D1 OA61253	Changed: KTV changes for PINPROT and ISO-4.
DK PIN Change	HCR77D1 OA61253	Changed: Support for General ISO PIN error mode. Allow specifying script MAC length via parameter.
DK PIN Verify	HCR77D1 OA61253	Changed: Support for General ISO PIN error mode.
Encrypted PIN Translate	HCR77D1 OA61253	Changed: Support for General ISO PIN error mode.
Encrypted PIN Translate2	HCR77D1 OA61253	Changed: Support for General ISO PIN error mode.
Encrypted PIN Verify2	HCR77D1 OA61253	New: Verify a trial PIN against a reference PIN in an encrypted PIN block.
PKA Key Translate	HCR77D1 OA61253	Changed: Support for translating CCA PKA key token to Azure object format.
Random Number Generate Long	HCR77D1 OA61253	Changed: Support for encrypting the returned random number under a cipher key.
Symmetric Algorithm Decipher	HCR77D1 OA61253	Changed: Added support for X9.23 padding.
Symmetric Algorithm Encipher	HCR77D1 OA61253	Changed: Support for AusPayNet MAC generation and verification. Added support for X9.23 padding.
Symmetric Key Export	HCR77D1 OA61253	Changed: Support for translating CCA AES key token to Azure object format.
ICSF Query Facility	HCR77D2	Changed: Retrieve CSFKEYS PKA ECC token <i>private-key name</i> checking control.
Diversified Key Generate	HCR77D1 OA60318	Changed: Added rule array keyword WRAPENH3.
ECC Diffie-Hellman	HCR77D1 OA60318	Changed: Added rule array keywords WRAPENH2 and WRAPENH3.
Key Part Import	HCR77D1 OA60318	Changed: Added rule array keywords WRAPENH2 and WRAPENH3.
Key Test2	HCR77D1 OA60318	Changed: Added rule array keyword KEY-LEN.

Table 9. Summary of new and changed ICSF callable services (continued)

Callable service	FMID	Description
Key Token Build	HCR77D1 OA60318	Changed: Added rule array keyword WRAPENH3.
Key Translate2	HCR77D1 OA60318	Changed: Added rule array keywords WRAPENH2 and WRAPENH3.
MAC Generate	HCR77D1 OA60318	Changed: Added rule array keyword TDESCMAC.
MAC Verify	HCR77D1 OA60318	Changed: Added rule array keyword TDESCMAC.
Multiple Clear Key Import	HCR77D1 OA60318	Changed: Added rule array keywords WRAPENH2 and WRAPENH3.
Multiple Secure Key Import	HCR77D1 OA60318	Changed: Added rule array keywords WRAPENH2 and WRAPENH3.
Remote Key Export	HCR77D1 OA60318	Changed: Added rule array keywords WRAPENH2 and WRAPENH3.
Symmetric Key Generate	HCR77D1 OA60318	Changed: Added rule array keywords WRAPENH2 and WRAPENH3.
Symmetric Key Import	HCR77D1 OA60318	Changed: Added rule array keywords WRAPENH2 and WRAPENH3.
Symmetric Key Import2	HCR77D1 OA60318	Changed: Added rule array keywords WRAPENH2 and WRAPENH3.
TR-31 Import	HCR77D1 OA60318	Changed: Added rule array keywords WRAPENH2 and WRAPENH3.
TR-34 Key Receive	HCR77D1 OA60318	Changed: Added rule array keywords WRAPENH2 and WRAPENH3.
Unique Key Derive	HCR77D1 OA60318	Changed: Added rule array keywords WRAPENH2 and WRAPENH3.
CVV Key Combine	HCR77D1 OA60318	Changed: Added rule array keyword WRAPENH3.
HMAC Generate	HCR77D1 OA60317	Changed: Added support for clear HMAC keys and exploit CPACF instructions.
HMAC Verify	HCR77D1 OA60317	Changed: Added support for clear HMAC keys and exploit CPACF instructions.
MAC Generate2	HCR77D1 OA60317	Changed: Added support for clear HMAC keys and exploit CPACF instructions.
MAC Verify2	HCR77D1 OA60317	Changed: Added support for clear HMAC keys and exploit CPACF instructions.
PKCS #11 Generate Keyed MAC	HCR77D1 OA60317	Changed: Exploit CPACF instructions for clear keys.
PKCS #11 Verify Keyed MAC	HCR77D1 OA60317	Changed: Exploit CPACF instructions for clear keys.
Format Preserving Algorithms Decipher	HCR77D1 OA59593	New: Decrypts payment card data using FFX algorithms.

Table 9. Summary of new and changed ICSF callable services (continued)

Callable service	FMID	Description
Format Preserving Algorithms Encipher	HCR77D1 OA59593	New: Encrypts payment card data using FFX algorithms.
Format Preserving Algorithms Translate	HCR77D1 OA59593	New: Translates payment card data from encryption under one key to encryption under another key using FFX algorithms.
Clear PIN Generate Alternate	HCR77D1 OA59593	Changed: Added support for ISO-4 PIN blocks.
Encrypted PIN Generate	HCR77D1 OA59593	Changed: Added support for ISO-4 PIN blocks.
Encrypted PIN Verify	HCR77D1 OA59593	Changed: Added support for the AES-DUKPT algorithm and ISO-4 PIN blocks.
Encrypted PIN Translate Enhanced	HCR77D1 OA59593	Changed: Added support for the AES-DUKPT algorithm.
Encrypted PIN Translate2	HCR77D1 OA59593	Changed: Added support for the AES-DUKPT algorithm.
FPE Encipher	HCR77D1 OA59593	Changed: Added support for the AES-DUKPT algorithm.
FPE Decipher	HCR77D1 OA59593	Changed: Added support for the AES-DUKPT algorithm.
FPE Translate	HCR77D1 OA59593	Changed: Added support for the AES-DUKPT algorithm.
Key Token Build2	HCR77D1 OA59593	Changed: <ul style="list-style-type: none"> • Added support for the A-DUKPT bit for the AES DKYGENKY keys. • Added support for the FFX algorithm key usage for the AES CIPHER keys.
PIN Change/Unblock	HCR77D1 OA59593	Changed: Added support for ISO-4 PIN blocks.
Recover PIN from Offset	HCR77D1 OA59593	Changed: Added support for ISO-4 PIN blocks.
Secure Messaging for PINs	HCR77D1 OA59593	Changed: Added support for ISO-4 PIN blocks.
TR-31 Import	HCR77D1 OA59593	Changed: Added support for the import of TR-31 Key Block B0 to AES DKYGENKY BDK.
TR-31 Translate	HCR77D1 OA59593	Changed: Added support for the export of AES DKYGENKY BDK to TR-31 Key Block B0.
Unique Key Derive	HCR77D1 OA59593	Changed: Added support for the AES-DUKPT algorithm.
Derive ICC MK	HCR77D1 OA58880	Changed: Added support for VISA CVN18.
Derive Session Key	HCR77D1 OA58880	Changed: Added support for VISA CVN18.

Table 9. Summary of new and changed ICSF callable services (continued)

Callable service	FMID	Description
Digital Signature Generate	HCR77D1 OA58880	Changed: Added support for EdDSA signature algorithm and exploitation of CPACF EC instructions. Added support for CRYSTALS-Dilithium digital signature algorithm.
Digital Signature Verify	HCR77D1 OA58880	Changed: Added support for EdDSA signature algorithm and exploitation of CPACF EC instructions. Added support for CRYSTALS-Dilithium digital signature algorithm.
EMV Transaction (ARQC/ARPC)	HCR77D1 OA58880	Changed: Added support for VISA CVN18.
Generate Issuer MK	HCR77D1 OA58880	Changed: Added support for VISA CVN18.
Key Test2	HCR77D1 OA58880	Changed: Support calculating key check value for an HMAC key in a TR-31 key block.
Key Token Build2	HCR77D1 OA58880	Changed: Support keyword to limit key encrypting keys to wrapping HMAC keys.
PKA Key Generate	HCR77D1 OA58880	Changed: Added support for generation of CRYSTALS-Dilithium (6,5) Key Pairs.
PKA Key Token Build	HCR77D1 OA58880	Changed: Added support for EC Edwards curves in EC public and private key tokens. Added support for CRYSTALS-Dilithium public and private key tokens.
PKA Key Token Change	HCR77D1 OA58880	Changed: Added support for CRYSTALS-Dilithium key pairs.
TR-31 Translate	HCR77D1 OA58880	Changed: Support exporting CCA HMAC keys to TR-31 key blocks.
TR-31 Import	HCR77D1 OA58880	Changed: Support importing TR-31 key blocks with HMAC keys to CCA key token.
TR-31 Optional Data Read	HCR77D1 OA58880	Changed: Support reading the optional data of a TR-31 key block containing an HMAC key.
TR-31 Parse	HCR77D1 OA58880	Changed: Support parsing a TR-31 key block containing an HMAC key.
Ciphertext Translate2	HCR77D0 OA57089	Changed: Support compliant-tagged AES key tokens.
CKDS Key Record Read2	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
Clear PIN Encrypt	HCR77D0 OA57089	Changed: Support compliant-tagged AES key tokens.
Digital Signature Generate	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
Digital Signature Verify	HCR77D0 OA57089	Changed: <ul style="list-style-type: none"> • Support compliant-tagged key tokens. • Support X.509 certificates.
Diversified Key Generate2	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.

Table 9. Summary of new and changed ICSF callable services (continued)

Callable service	FMID	Description
Diversify Directed Key	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
DK Deterministic PIN Generate	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
DK Migrate PIN	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
DK PAN Modify in Transaction	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
DK PAN Translate	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
DK PIN Change	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
DK PIN Verify	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
DK PRW Card Number Update	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
DK PRW Card Number Update2	HCR77D0 OA57089	New: Enhanced processing for PIN blocks for cards.
DK PRW CMAC Generate	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
DK Random PIN Generate	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
DK Random PIN Generate2	HCR77D0 OA57089	New: Enhanced processing for PIN blocks for cards.
DK Regenerate PRW	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
Encrypted PIN Translate2	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
Field Level Decipher	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
Field Level Encipher	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
Key Generate2	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
Key Test2	HCR77D0 OA57089	Changed: Support compliant-tagged AES key tokens.
MAC Generate2	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
MAC Verify2	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
PKA Decrypt	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.

Table 9. Summary of new and changed ICSF callable services (continued)

Callable service	FMID	Description
PKA Encrypt	HCR77D0 OA57089	Changed: <ul style="list-style-type: none"> • Support compliant-tagged key tokens. • Support X.509 certificates.
PKA Key Generate	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
PKA Key Import	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
PKA Key Translate	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
Public Infrastructure Certificate	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
Random Number Generate Long	HCR77D0 OA57089	Changed: Support TR-34 protocol key distribution.
Restrict Key Attribute	HCR77D0 OA57089	Changed: Support compliant-tagged AES key tokens.
Symmetric Algorithm Decipher	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
Symmetric Algorithm Encipher	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
Symmetric Key Decipher	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
Symmetric Key Encipher	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
Symmetric Key Export	HCR77D0 OA57089	Changed: <ul style="list-style-type: none"> • Support compliant-tagged key tokens. • Support X.509 certificates.
Symmetric Key Generate	HCR77D0 OA57089	Changed: <ul style="list-style-type: none"> • Support compliant-tagged key tokens. • Support X.509 certificates.
Symmetric Key Import	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
Symmetric Key Import2	HCR77D0 OA57089	Changed: Support compliant-tagged key tokens.
TR-31 Translate	HCR77D0 OA57089	Changed: <ul style="list-style-type: none"> • Support TR-34 protocol key distribution. • Support compliant-tagged AES key tokens.
TR-31 Import	HCR77D0 OA57089	Changed: <ul style="list-style-type: none"> • Support TR-34 protocol key distribution. • Support compliant-tagged AES key tokens.

Table 9. Summary of new and changed ICSF callable services (continued)

Callable service	FMID	Description
TR-34 Bind-Begin	HCR77D0 OA57089	New: Support TR-34 protocol bind processing.
TR-34 Bind-Complete	HCR77D0 OA57089	New: Support TR-34 protocol bind processing.
TR-34 Key Distribution	HCR77D0 OA57089	New: Support TR-34 protocol key distribution.
TR-34 Key Receive	HCR77D0 OA57089	New: Support TR-34 protocol key distribution.
Diversify Directed Key	HCR77C1 OA58880	Changed: New <i>key_type_vector</i> values.
DK PIN Change	HCR77C1 OA58880	Changed: New script processing keywords.
Key Generate2	HCR77C1 OA58880	Changed: Allow generation of OPOP EPVR/OPIN Key Pair.
TR-31 Translate	HCR77C1 OA58880	Changed: Allow export of CCA IPINENC/OPINENC keys to TR-31 P0:B.
Encrypted PIN Translate2	HCR77C1 OA58306	Changed: Additional requirements for AES PINPROT input PIN encrypting key. New access control.
Key Token Build2	HCR77C1 OA58306	Changed: New key usages for AES PINPROT keys.
Encrypted PIN Translate2	HCR77C1 OA57088	Changed: New key usage for AES MAC authentication key.
Control Vector Generate	HCR77C1 OA55184	Changed: New rule array keywords in support of triple-length DES keys.
Diversify Directed Key	HCR77C1 OA55184	New: DK PIN support for diversified keys.
Key Part Import	HCR77C1 OA55184	Changed: New rule array keywords in support of triple-length DES keys.
Key Test	HCR77C1 OA55184	Changed: New rule array keyword in support of triple-length encrypted DES keys.
Key Test2	HCR77C1 OA55184	Changed: New rule array keyword in support of triple-length encrypted DES keys.
Key Test Extended	HCR77C1 OA55184	Changed: New rule array keyword in support of triple-length encrypted DES keys.
Key Token Build	HCR77C1 OA55184	Changed: New rule array keyword in support of triple-length DES keys.
Key Token Build2	HCR77C1 OA55184	Changed: Support for new AES key attributes and KDKGENKY key type.
TR-31 Translate	HCR77C1 OA55184	Changed: Support for ISO-20038 AES key blocks and AES key-encrypting keys.
TR-31 Import	HCR77C1 OA55184	Changed: Support for ISO-20038 AES key blocks and AES key-encrypting keys

Table 9. Summary of new and changed ICSF callable services (continued)

Callable service	FMID	Description
Authentication Parameter Generate	HCR77C1	Changed: Support compliant-tagged key tokens.
Ciphertext Translate2	HCR77C1	Changed: Support compliant-tagged key tokens.
Clear PIN Encrypt	HCR77C1	Changed: Support compliant-tagged key tokens.
Clear PIN Generate	HCR77C1	Changed: Support compliant-tagged key tokens.
Clear PIN Generate Alternate	HCR77C1	Changed: Support compliant-tagged key tokens.
Control Vector Generate	HCR77C1	Changed: Generate control vector with the compliant-tag bit on.
Cryptographic Usage Statistic	HCR77C1	New: Track cryptographic usage external to the ICSF address space.
Decipher	HCR77C1	Changed: Support compliant-tagged key tokens.
Derive ICC MK	HCR77C1	Changed: Derive compliant-tagged key tokens.
Derive Session Key	HCR77C1	Changed: Derive compliant-tagged key tokens.
Digital Signature Verify	HCR77C1	Changed: Allow signature verification using an X.509 digital certificate.
Diversified Key Generate	HCR77C1	Changed: Generate compliant-tagged key tokens.
EMV Scripting Service	HCR77C1	Changed: Support compliant-tagged key tokens.
EMV Transaction (ARQC/ARPC) Service	HCR77C1	Changed: Support compliant-tagged key tokens.
EMV Verification Functions	HCR77C1	Changed: Support compliant-tagged key tokens.
Encipher	HCR77C1	Changed: Support compliant-tagged key tokens.
Encrypted PIN Generate	HCR77C1	Changed: Support compliant-tagged key tokens.
Encrypted PIN Translate	HCR77C1	Changed: Support compliant-tagged key tokens.
Encrypted PIN Translate Enhanced	HCR77C1	Changed: Support compliant-tagged key tokens.
Encrypted PIN Verify	HCR77C1	Changed: Support compliant-tagged key tokens.
Field Level Decipher	HCR77C1	Changed: Version 05 AES CIPHER key tokens allowed.
Field Level Encipher	HCR77C1	Changed: Version 05 AES CIPHER key tokens allowed.
FPE Decipher	HCR77C1	Changed: Support compliant-tagged key tokens.
FPE Encipher	HCR77C1	Changed: Support compliant-tagged key tokens.
FPE Translate	HCR77C1	Changed: Support compliant-tagged key tokens.
Generate Issuer MK	HCR77C1	Changed: Generate compliant-tagged key token.
ICSF Query Facility	HCR77C1	Changed: Retrieve compliance data for a CCA coprocessor.
ICSF Query Facility2	HCR77C1	Changed: Retrieve CCA compliance information for the system.
Key Export	HCR77C1	Changed: Support compliant-tagged key tokens.
Key Generate	HCR77C1	Changed: Generate compliant-tagged key tokens.

Table 9. Summary of new and changed ICSF callable services (continued)

Callable service	FMID	Description
Key Import	HCR77C1	Changed: Support compliant-tagged key tokens.
Key Record Read2	HCR77C1	Changed: Returns protected key for version 05 AES CIPHER key tokens.
Key Test	HCR77C1	Changed: Support compliant-tagged key tokens.
Key Test2	HCR77C1	Changed: <ul style="list-style-type: none"> • Calculate a 3-byte or 5-byte CMACZERO verification pattern for DES keys. • Support compliant-tagged key tokens.
Key Test Extended	HCR77C1	Changed: Support compliant-tagged key tokens.
Key Token Build	HCR77C1	Changed: Generate compliant-tagged key token skeleton.
Key Token Build2	HCR77C1	Changed: Build version 05 key tokens that can be exported to CPACF protected key format.
Key Translate	HCR77C1	Changed: Support compliant-tagged key tokens.
Key Translate2	HCR77C1	Changed: Support compliant-tagged key tokens.
MAC Generate	HCR77C1	Changed: Support compliant-tagged key tokens.
MAC Verify	HCR77C1	Changed: Support compliant-tagged key tokens.
PIN Change/Unblock	HCR77C1	Changed: Support compliant-tagged key tokens.
Prohibit Export	HCR77C1	Changed: Support compliant-tagged key tokens.
Prohibit Export Extended	HCR77C1	Changed: Support compliant-tagged key tokens.
Public Infrastructure Certificate	HCR77C1	New: Create a certificate signing request.
Recover PIN from Offset	HCR77C1	Changed: Support compliant-tagged key tokens.
Restrict Key Attribute	HCR77C1	Changed: Support compliant-tagged key tokens.
Secure Messaging for PINs	HCR77C1	Changed: Support compliant-tagged key tokens.
Symmetric Key Decipher	HCR77C1	Changed: Version 05 AES CIPHER key tokens allowed.
Symmetric Key Encipher	HCR77C1	Changed: Version 05 AES CIPHER key tokens allowed.
TR-31 Translate	HCR77C1	Changed: Support compliant-tagged key tokens.
TR-31 Import	HCR77C1	Changed: Support compliant-tagged key tokens.
Transaction Validation	HCR77C1	Changed: Support compliant-tagged key tokens.
Unique Key Derive	HCR77C1	Changed: Derive compliant-tagged key tokens.
VISA CVV Service Generate	HCR77C1	Changed: Support compliant-tagged key tokens.
VISA CVV Service Verify	HCR77C1	Changed: Support compliant-tagged key tokens.
Digital Signature Generate	HCR77C0	Changed: Support for RSA-PSS digital signature scheme.

Table 9. Summary of new and changed ICSF callable services (continued)

Callable service	FMID	Description
Digital Signature Verify	HCR77C0	Changed: Support for RSA-PSS digital signature scheme.
Key Data Set List	HCR77C0	Changed: New option to list unsupported CCA key in CKDS and PKDS.
PKA Key Generate	HCR77C0	Changed: Support for additional RSA public exponent values.
PKA Key Token Build	HCR77C0	Changed: Support for additional RSA public exponent values. Support for RSA-PSS digital signature scheme.
PKA Key Translate	HCR77C0	Changed: Support for RSA-PSS digital signature scheme.

CCA access control

The following table summarizes the new and changed CCA access controls for ICSF FMID HCR77F0. For complete reference information on these CCA access controls, see [z/OS Cryptographic Services ICSF Application Programmer's Guide](#).

Note: When an APAR number is listed in the FMID column along with an ICSF FMID, the FMID is the earliest release where the new access control is supported.

Table 10. Summary of new and changed CCA access controls

Access control	Description	FMID or APAR number	Services affected	Offset
Permit Regeneration Data	Name change, affects more services	HCR77D2 OA66395	CSNDPKG/ CSNFPKG, CSNDDSG/ CSNFDSDG, CSNDPKE/ CSNFPKE	027D
PKA Decrypt - Allow ML-KEM, CRYSTALS-Kyber keys	Name change	HCR77D2 OA66395	CSNDPKD / CSNFPKD	0084
PKA Encrypt - Allow ML-KEM, CRYSTALS-Kyber keys	Name change	HCR77D2 OA66395	CSNDPKE / CSNFPKE	0083
PKA Key Generate – Clear ML-DSA, CRYSTALS-Dilithium keys	Name change	HCR77D2 OA66395	CSNDPKG / CSNFPKG	027F
PKA Key Generate – Clear ML-KEM, CRYSTALS-Kyber keys	Name change	HCR77D2 OA66395	CSNDPKG / CSNFPKG	020E
T31X – Permit AES PINPROT to P0:B	New	HCR77D2 OA66395	CSNBT31X / CSNET31X	050A
Key Part Import2 - Allow TR-31 key blocks	New	HCR77D1 OA61978	CSNBKPI2	03BC
PKA Decrypt - Disallow PKOAEP2	New	HCR77D1 OA61978	CSNDPKD / CSNFPKD	03F2
PKA Encrypt - Disallow PKOAEP2	New	HCR77D1 OA61978	CSNDPKE / CSNFPKE	03F1
SKY - Allow K0 for secmsg key identifier	New	HCR77D1 OA61978	CSNBSKY / CSNESKY	03F3

Table 10. Summary of new and changed CCA access controls (continued)

Access control	Description	FMID or APAR number	Services affected	Offset
SPN - Allow K0 for secmsg key identifier	New	HCR77D1 OA61978	CSNBSPN / CSNESP	03F4
T31I - Disallow Partial DES Key Import with CV in IBMC01 OB	New	HCR77D1 OA61978	CSNBT31I / CSNET31I	006F
T31I - Permit AES 10 to KDKGENKY:KDKTYPEA	New	HCR77D1 OA61978	CSNBT31I / CSNET31I	0387
T31I - Permit AES 11 to KDKGENKY:KDKTYPEB	New	HCR77D1 OA61978	CSNBT31I / CSNET31I	0388
T31I - Permit B1 to DES KEYGENKY:DUKPT	New	HCR77D1 OA61978	CSNBT31I / CSNET31I	03E8
T31I - Permit B3 to DES DKYGENKY	New	HCR77D1 OA61978	CSNBT31I / CSNET31I	03E9
T31I - Permit D3 to CIPHER:XLATE	New	HCR77D1 OA61978	CSNBT31I / CSNET31I	03EA
T31I - Permit DES 12 to DKYGENKY:DKYLO+DMPIN	New	HCR77D1 OA61978	CSNBT31I / CSNET31I	0389
T31I - Permit F0:N/X to DES DKYGENKY:DKYLO+DMAC	New	HCR77D1 OA61978	CSNBT31I / CSNET31I	03EB
T31I - Permit F0:N/X to DES DKYGENKY:DKYLO+DMV	New	HCR77D1 OA61978	CSNBT31I / CSNET31I	03EC
T31I - Permit F0:X to AES DKYGENKY:DKYLO+D-MAC+GENERATE+CMAC	New	HCR77D1 OA61978	CSNBT31I / CSNET31I	0504
T31I - Permit F1:N/E/D/B/X to DES DKYGENKY:DKYLO+DDATA	New	HCR77D1 OA61978	CSNBT31I / CSNET31I	03EE
T31I - Permit F1:N/E/D/B/X to DES DKYGENKY:DKYLO+DMPIN	New	HCR77D1 OA61978	CSNBT31I / CSNET31I	03ED
T31I - Permit F1:X to AES DKYGENKY:DKYLO+D-SECMSG+SMPIN+ANY-USE	New	HCR77D1 OA61978	CSNBT31I / CSNET31I	0505
T31I - Permit F2:N/X to DES DKYGENKY:DKYLO+DMAC	New	HCR77D1 OA61978	CSNBT31I / CSNET31I	03EF
T31I - Permit F2:X to AES DKYGENKY:DKYLO+D-MAC+GENERATE+CMAC	New	HCR77D1 OA61978	CSNBT31I / CSNET31I	0506
T31I - Permit F3:E/B to AES CIPHER:ENCRYPT/ENCRYPT+DECRYPT	New	HCR77D1 OA61978	CSNBT31I / CSNET31I	0508
T31I - Permit F3:N/E/D/B/G/X to DES ENCIPHER	New	HCR77D1 OA61978	CSNBT31I / CSNET31I	0502
T31I - Permit F3:X to AES DKYGENKY:D-CIPHER+ENCRYPT+DECRYPT+CBC	New	HCR77D1 OA61978	CSNBT31I / CSNET31I	0507
T31I - Permit F4:N/B/X to DES DKYGENKY:DKYLO+DDATA	New	HCR77D1 OA61978	CSNBT31I / CSNET31I	0503

Table 10. Summary of new and changed CCA access controls (continued)

Access control	Description	FMID or APAR number	Services affected	Offset
T31I - Permit F4:X to AES DKYGENKY:DKYLO+D-CIPHER+ENC+DEC+CBC	New	HCR77D1 OA61978	CSNBT31I / CSNET31I	0509
T31I - Permit M6 to DES MAC	New	HCR77D1 OA61978	CSNBT31I / CSNET31I	03F0
T31X - Disallow Partial DES Key Export with CV in IBMC01 OB	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	006E
T31X - Permit AES CIPHER, DKYGENKY:D-ALL/DCIPHER to F3:E/B/X	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	0500
T31X - Permit AES DKYGENKY:D-ALL/DCIPHER to F1:X	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	03FE
T31X - Permit AES DKYGENKY:D-ALL/DCIPHER to F4:X	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	0501
T31X - Permit AES DKYGENKY:D-ALL/DMAC to F0:X	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	03FD
T31X - Permit AES DKYGENKY:D-ALL/DMAC to F2:X	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	03FF
T31X - Permit CIPHER:XLATE to D3	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	03E1
T31X - Permit DES DATA/MAC/CIPHER/ENCIPHER to F3:N/G/E/X	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	03FA
T31X - Permit DES DKYGENKY, AES KDKGENKY to B3	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	03E0
T31X - Permit DES DKYGENKY:DKYLO+DALL to F0:N/X	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	03E6
T31X - Permit DES DKYGENKY:DKYLO+DALL to F1:N/X	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	03F7
T31X - Permit DES DKYGENKY:DKYLO+DALL to F2:N/X	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	03F9
T31X - Permit DES DKYGENKY:DKYLO+DALL to F4:N/X	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	03FC
T31X - Permit DES DKYGENKY:DKYLO+DDATA to F1:N/X	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	03F5
T31X - Permit DES DKYGENKY:DKYLO+DDATA to F4:N/X	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	03FB
T31X - Permit DES DKYGENKY:DKYLO+DMAC to F0:N/X	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	03E4
T31X - Permit DES DKYGENKY:DKYLO+DMAC to F2:N/X	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	03F8
T31X - Permit DES DKYGENKY:DKYLO+DMPIN to 12	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	0385

Table 10. Summary of new and changed CCA access controls (continued)

Access control	Description	FMID or APAR number	Services affected	Offset
T31X - Permit DES DKYGENKY:DKYLO+DMPIN to F1:N/X	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	03F6
T31X - Permit DES DKYGENKY:DKYLO+DMV to F0:N/X	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	03E5
T31X - Permit DES KEYGENKY:DUKPT, AES DKYGENKY:DUKPT to B1	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	03DF
T31X - Permit DES MAC to M6	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	03E7
T31X - Permit SECMSG:SMKEY to K0	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	03E3
T31X - Permit SECMSG:SMPIN to P0	New	HCR77D1 OA61978	CSNBT31X / CSNET31X	03E2
TR31 Create - Allow AES key blocks	New	HCR77D1 OA61978	CSNBT31C	03C1
TR31 Create - Allow creation with key block version ID A	New	HCR77D1 OA61978	CSNBT31C	03C7
TR31 Create - Allow creation with key block version ID B	New	HCR77D1 OA61978	CSNBT31C	03C8
TR31 Create - Allow creation with key block version ID C	New	HCR77D1 OA61978	CSNBT31C	03C9
TR31 Create - Allow creation with key block version ID D	New	HCR77D1 OA61978	CSNBT31C	03CA
TR31 Create - Allow DES key blocks	New	HCR77D1 OA61978	CSNBT31C	03C2
TR31 Create - Allow HMAC key blocks	New	HCR77D1 OA61978	CSNBT31C	03C3
TR31 Create - Key set	New	HCR77D1 OA61978	CSNBT31C	03C5
TR31 Create - Key set extended	New	HCR77D1 OA61978	CSNBT31C	03C6
TR31 Create - OP	New	HCR77D1 OA61978	CSNBT31C	03C4
T31X - Permit EXPORTER to K0:B	New	HCR77D0 OA63531	CSNBT31X / CSNET31X	02AD
T31X - Permit IMPORTER to K0:B	New	HCR77D0 OA63531	CSNBT31X / CSNET31X	02AE
ECC Diffie-Hellman - Allow Hybrid QSA Scheme	New	HCR77D1 OA61609	CSNDEDH / CSNFEDH	035D
PKA Decrypt - Allow CRYSTALS-Kyber keys	New	HCR77D1 OA61609	CSNDPKD / CSNFPKD	0084

Table 10. Summary of new and changed CCA access controls (continued)

Access control	Description	FMID or APAR number	Services affected	Offset
PKA Encrypt - Allow CRYSTALS-Kyber keys	New	HCR77D1 OA61609	CSNDPKE / CSNFPKE	0083
PKA Key Generate - Clear CRYSTALS-Kyber keys	New	HCR77D1 OA61609	CSNDPKG / CSNFPKG	020E
PKA Key Translate - Allow ECC private key export	New	HCR77D1 OA61609	CSNDPKT / CSNFPKT	00EF
PKA Key Translate - Allow QSA private key export	New	HCR77D1 OA61609	CSNDPKT / CSNFPKT	020F
Symmetric token wrapping - External enhanced method version 3	New	HCR77D1 OA61609	Services that wrap external symmetric key tokens.	0143
Symmetric token wrapping - Internal enhanced method version 3	New	HCR77D1 OA61609	Services that wrap internal symmetric key tokens.	0145
General ISO PIN Error Mode	New	HCR77D1 OA61253	CSNBDPC CSNBDPV CSNBPTR CSNBPTR2	039F
Encrypted PIN Translate - Translate PIN Check Mode	New	HCR77D1 OA61253	CSNBPTR CSNBPTR2	03A0
Encrypted PIN Verify2 - REFPIN	New	HCR77D1 OA61253	CSNBPVR2	03B0
Encrypted PIN Verify2 - TRUNCPIN	New	HCR77D1 OA61253	CSNBPVR2	03B1
Symmetric Algorithm Encipher - Allow A28MACGN and A28MACVR	New	HCR77D1 OA61253	CSNBSAE	03B2
Symmetric Algorithm Encipher - Allow A28OWFCL	New	HCR77D1 OA61253	CSNBSAE	03B3
Symmetric Algorithm Encipher - Allow A28OWFEC	New	HCR77D1 OA61253	CSNBSAE	03B4
Random Number Generate Long - TDES-CBC	New	HCR77D1 OA61253	CSNBRNGL	03B5
PKA Key Translate - From CCA RSA to CKM-RAKW format	New	HCR77D1 OA61253	CSNDPKT	03B6
PKA Key Translate - From CCA ECC to CKM-RAKW format	New	HCR77D1 OA61253	CSNDPKT	03B7
Symmetric Key Export - CKM-RAKW	New	HCR77D1 OA61253	CSNDSYX	03B8

Table 10. Summary of new and changed CCA access controls (continued)

Access control	Description	FMID or APAR number	Services affected	Offset
Diversified Key Generate - A28OWFEC	New	HCR77D1 OA61253	CSNBDKG	03B9
Diversified Key Generate - A28OWFCL	New	HCR77D1 OA61253	CSNBDKG	03BA
Diversified Key Generate - A28XOREC	New	HCR77D1 OA61253	CSNBDKG	03BB
Key Test2 - AES, KEY-LEN	New	HCR77D1 OA60318	CSNBKYT2	003B
Key Test2 - DES, KEY-LEN	New	HCR77D1 OA60318	CSNBKYT2	003C
Disable 56-bit length DES keys	New	HCR77D0 OA60165	All CCA callable services that accept or generate 56-bit length DES keys.	0026
Disable 56-bit effective length DES keys	New	HCR77D0 OA60165	All CCA callable services that accept or generate 56-bit effective length DES keys including loading master keys.	0027
Disable RSA keys with less than 1024-bit modulus length	New	HCR77D0 OA60165	All CCA callable services that accept or generate RSA keys with less than 1024-bit modulus length.	002B
Disable RSA keys with less than 2048-bit modulus length	New	HCR77D0 OA60165	All CCA callable services that accept or generate RSA keys with less than 2048-bit modulus length.	002C
Disable ECC keys weaker than 224-bit	New	HCR77D0 OA60165	All CCA callable services that accept or generate ECC keys weaker than 224-bit.	004D
ECC Diffie-Hellman - Allow Koblitz Curve 256	New	HCR77D1 OA59593	CSNDEDH	035E

Table 10. Summary of new and changed CCA access controls (continued)

Access control	Description	FMID or APAR number	Services affected	Offset
Format Preserving Algorithms Encipher/Decipher – Allow FF1	New	HCR77D1 OA59593	CSNBFFXD CSNBFFXE CSNBFFXT	0396
Format Preserving Algorithms Encipher/Decipher – Allow FF2	New	HCR77D1 OA59593	CSNBFFXD CSNBFFXE CSNBFFXT	0397
Format Preserving Algorithms Encipher/Decipher – Allow FF2.1	New	HCR77D1 OA59593	CSNBFFXD CSNBFFXE CSNBFFXT	0398
Format Preserving Algorithms Encipher	New	HCR77D1 OA59593	CSNBFFXE	0399
Format Preserving Algorithms Decipher	New	HCR77D1 OA59593	CSNBFFXD	039A
Format Preserving Algorithms Translate	New	HCR77D1 OA59593	CSNBFFXT	039B
Format Preserving Algorithms Translate - Allow weaker output key	New	HCR77D1 OA59593	CSNBFFXT	039C
T31X - Permit AES DKYGENKY: DUKPT BDk to B0:X	New	HCR77D1 OA59593	CSNBT31X	01CF
TR31I - Permit B0:X to AES DKYGENKY:DUKPT BDk	New	HCR77D1 OA59593	CSNBT31I	017E
PKA Key Generate - Clear CRYSTALS-Dilithium keys	New	HCR77D1 OA58880	CSFNDPKG	027F
T31X - Permit HMAC MAC to M7:G/V/C	New	HCR77D1 OA58880	CSNBT31X	020D
T31I - Permit M7:G/V/C to HMAC MAC: GENERATE/VERIFY	New	HCR77D1 OA58880	CSNBT31I	017D
DK Random PIN Generate2	New	HCR77D0 OA57089	CSNBDRG2	0024
DK PRW Card Number Update2	New	HCR77D0 OA57089	CSNBDCU2	0025
TR-34 Bind-Begin	New	HCR77D0 OA57089	CSNDT34B	01F0
TR-34 Bind-Begin - allow BINDCR	New	HCR77D0 OA57089	CSNDT34B	01F1
TR-34 Bind-Begin - allow UNBINDCR	New	HCR77D0 OA57089	CSNDT34B	01F2
TR-34 Bind-Begin - allow REBINDCR	New	HCR77D0 OA57089	CSNDT34B	01F3

Table 10. Summary of new and changed CCA access controls (continued)

Access control	Description	FMID or APAR number	Services affected	Offset
TR-34 Bind-Complete	New	HCR77D0 OA57089	CSNDT34C	01F4
TR-34 Bind-Complete - allow BINDKRDC	New	HCR77D0 OA57089	CSNDT34C	01F5
TR-34 Bind-Complete - allow BINDRV	New	HCR77D0 OA57089	CSNDT34C	01F6
TR-34 Bind-Complete - allow UNBINDRV	New	HCR77D0 OA57089	CSNDT34C	01F7
TR-34 Bind-Complete - allow REBINDRV	New	HCR77D0 OA57089	CSNDT34C	01F8
TR-34 Key Distribution	New	HCR77D0 OA57089	CSNDT34D	01F9
TR-34 Key Distribution – allow 2PASSCRE	New	HCR77D0 OA57089	CSNDT34D	01FA
TR-34 Key Distribution – allow 1PASSCRE	New	HCR77D0 OA57089	CSNDT34D	01FB
TR-34 Key Receive	New	HCR77D0 OA57089	CSNDT34R	01FC
TR-34 Key Receive – allow 2PASSRCV	New	HCR77D0 OA57089	CSNDT34R	01FD
TR-34 Key Receive – allow 1PASSRCV	New	HCR77D0 OA57089	CSNDT34R	01FE
Permit X.509 without PKI root validation	New	HCR77D0 OA57089	CSNDDSV CSNDPKE CSNDSYX CSNDSYG CSNDT34B CSNDT34C CSNDT34D CSNDT34R	01FF
TR-34 Key Receive – allow wrapping method override keywords	New	HCR77D0 OA57089	CSNDT34R	01DF
TR-34 Key Distribution - permit DES EXPORTER to K0 or K1	New	HCR77D0 OA57089	CSNDT34D	0242
TR-34 Key Distribution - permit DES IMPORTER to K0 or K1	New	HCR77D0 OA57089	CSNDT34D	0243
TR-34 Key Distribution - permit AES EXPORTER to K0	New	HCR77D0 OA57089	CSNDT34D	0244
TR-34 Key Distribution - permit AES EXPORTER to K1	New	HCR77D0 OA57089	CSNDT34D	0245
TR-34 Key Distribution - permit AES IMPORTER to K0	New	HCR77D0 OA57089	CSNDT34D	0246

Table 10. Summary of new and changed CCA access controls (continued)

Access control	Description	FMID or APAR number	Services affected	Offset
TR-34 Key Distribution - permit AES IMPORTER to K1	New	HCR77D0 OA57089	CSNDT34D	0247
TR-34 Key Receive – permit DES EXPORTER	New	HCR77D0 OA57089	CSNDT34R	0248
TR-34 Key Receive – permit DES IMPORTER	New	HCR77D0 OA57089	CSNDT34R	0249
TR-34 Key Receive – permit AES EXPORTER	New	HCR77D0 OA57089	CSNDT34R	024A
TR-34 Key Receive – permit AES IMPORTER	New	HCR77D0 OA57089	CSNDT34R	024B
TR-34 Key Receive – permit AES EXPORTER with EXP TT31D	New	HCR77D0 OA57089	CSNDT34R	024C
TR-34 Key Receive – permit AES IMPORTER with IMP TT31D	New	HCR77D0 OA57089	CSNDT34R	024D
PKA Key Translate – allow COMP-TAG	New	HCR77D0 OA57089	CSNDPKT	01EE
PKA Key Translate – allow COMP-CHK	New	HCR77D0 OA57089	CSNDPKT	01EF
PKA Key Translate – allow INTUSCHG	New	HCR77D0 OA57089	CSNDPKT	02EE
Key Generate2 – Allow GEN of OPOP EPVR/ OPIN Key Pair	New	HCR77C1 OA58880	CSNBKGN2	039D
T31X – Permit DES OPINENC/IPINENC to P0:B	New	HCR77C1 OA58880	CSNBT31X	039E
Disallow PIN block format ISO-1	New	HCR77C1 OA58306	CSNBCPE CSNBCPA CSNBEPG CSNBPTR CSNBPTRE CSNBPTR2 CSNBPVR CSNBPCU CSNBPFO CSNBSPN CSNBDMP CSNBDPMT CSNBDPC CSNBDPV	032F
Encrypted PIN Translate2 - Permit ISO-4 to ISO-1 RFMT4TO1	New	HCR77C1 OA58306	CSNBPTR2	0394
Encrypted PIN Translate2 - Permit ISO-4 to ISO-4 PTR2AUTH	New	HCR77C1 HCR77D0 OA57088	CSNBPTR2	0395

Table 10. Summary of new and changed CCA access controls (continued)

Access control	Description	FMID or APAR number	Services affected	Offset
Disallow translation from AES wrapping to DES wrapping	New	HCR77C1 OA55184	CSNBKTR2 CSNBPTR2 CSNDPKT	01C5
Disallow translation from AES wrapping to weaker AES wrapping	New	HCR77C1 OA55184	CSNBKTR2 CSNBPTR2 CSNDPKT	01C6
Disallow translation from DES wrapping to weaker DES wrapping	New	HCR77C1 OA55184	CSNBAPG CSNBEPG CSNBKTR CSNBKTR2 CSNBPFO CSNBPTR CSNBPTRE CSNBPTR2 CSNBSKY CSNDPKT	01C7
Diversify Directed Key	New	HCR77C1 OA55184	CSNBDDK	0080
Diversify Directed Key – Allow KDFFM DERIVE	New	HCR77C1 OA55184	CSNBDDK	0081
Diversify Directed Key – Allow KDFFM GENERATE	New	HCR77C1 OA55184	CSNBDDK	0082
T31X - Permit version A TR-31 key blocks	Name change	HCR77C1 OA55184	CSNBT31X	014D
T31X - Permit version B TR-31 key blocks	Name change	HCR77C1 OA55184	CSNBT31X	014E
T31X - Permit version C TR-31 key blocks	Name change	HCR77C1 OA55184	CSNBT31X	014F
T31I - Permit version A TR-31 key blocks	Name change	HCR77C1 OA55184	CSNBT31I	0150
T31I - Permit version B TR-31 key blocks	Name change	HCR77C1 OA55184	CSNBT31I	0151
T31I - Permit version C TR-31 key blocks	Name change	HCR77C1 OA55184	CSNBT31I	0152
T31I - Permit override of default wrapping method	Name change	HCR77C1 OA55184	CSNBT31I	0153
T31X - Permit any CCA DES key if INCL-CV is specified	Name change	HCR77C1 OA55184	CSNBT31X	0158
T31I - Permit C0:G/C/V to DES MAC/ MACVER:CVVKEY-A	Name change	HCR77C1 OA55184	CSNBT31I	015A
T31I - C0:G/C/V to DES MAC/MACVER:AMEX- CSC	Name change	HCR77C1 OA55184	CSNBT31I	015B

Table 10. Summary of new and changed CCA access controls (continued)

Access control	Description	FMID or APAR number	Services affected	Offset
T31I - K0:E to DES EXPORTER/OKEYXLAT	Name change	HCR77C1 OA55184	CSNBT31I	015C
T31I - K0:D to DES IMPORTER/IKEYXLAT	Name change	HCR77C1 OA55184	CSNBT31I	015D
T31I - K0:B to DES EXPORTER/OKEYXLAT	Name change	HCR77C1 OA55184	CSNBT31I	015E
T31I - K0:B to DES IMPORTER/IKEYXLAT	Name change	HCR77C1 OA55184	CSNBT31I	015F
T31I - Permit K1/K4:E to DES EXPORTER/OKEYXLAT	Name change	HCR77C1 OA55184	CSNBT31I	0160
T31I - Permit K1/K4:D to DES IMPORTER/IKEYXLAT	Name change	HCR77C1 OA55184	CSNBT31I	0161
T31I - Permit K1/K4:B to DES EXPORTER/OKEYXLAT	Name change	HCR77C1 OA55184	CSNBT31I	0162
T31I - Permit K1/K4:B to DES IMPORTER/IKEYXLAT	Name change	HCR77C1 OA55184	CSNBT31I	0163
T31I - Permit M0/M1/M3:G/C/V to DES MAC/MACVER:ANY-MAC	Name change	HCR77C1 OA55184	CSNBT31I	0164
T31I - Permit P0:E to DES OPINENC	Name change	HCR77C1 OA55184	CSNBT31I	0165
T31I - Permit P0:D to DES IPINENC	Name change	HCR77C1 OA55184	CSNBT31I	0166
T31I - Permit V0:N/G/C to DES PINGEN:NO-SPEC NOOFFSET	Name change	HCR77C1 OA55184	CSNBT31I	0167
T31I - Permit V0:N/V to DES PINVER:NO-SPEC NOOFFSET	Name change	HCR77C1 OA55184	CSNBT31I	0168
T31I - Permit V1:N/G/C to DES PINGEN:IBM-PIN/IBM-PINO NOOFFSET	Name change	HCR77C1 OA55184	CSNBT31I	0169
T31I - Permit V1:N/V to DES PINVER:IBM-PIN/IBM-PINO NOOFFSET	Name change	HCR77C1 OA55184	CSNBT31I	016A
T31I - Permit V2:N/G/C to DES PINGEN:VISA-PVV	Name change	HCR77C1 OA55184	CSNBT31I	016B
T31I - Permit V2:N/V to DES PINVER:VISA-PVV	Name change	HCR77C1 OA55184	CSNBT31I	016C
T31I - Permit E0:N/X to DES DKYGENKY:DKYLO+DMAC	Name change	HCR77C1 OA55184	CSNBT31I	016D
T31I - Permit E0:N/X to DES DKYGENKY:DKYLO+DMV	Name change	HCR77C1 OA55184	CSNBT31I	016E
T31I - Permit E0:N/X to DES DKYGENKY:DKYL1+DMAC	Name change	HCR77C1 OA55184	CSNBT31I	016F

Table 10. Summary of new and changed CCA access controls (continued)

Access control	Description	FMID or APAR number	Services affected	Offset
T31I - Permit E0:N/X to DES DKYGENKY:DKYL1+DMV	Name change	HCR77C1 OA55184	CSNBT31I	0170
T31I - Permit E1:N/E/D/B/X to DES DKYGENKY:DKYL0+DMPIN	Name change	HCR77C1 OA55184	CSNBT31I	0171
T31I - Permit E1:N/E/D/B/X to DES DKYGENKY:DKYL0+DDATA	Name change	HCR77C1 OA55184	CSNBT31I	0172
T31I - Permit E1:N/E/D/B/X to DES DKYGENKY:DKYL1+DMPIN	Name change	HCR77C1 OA55184	CSNBT31I	0173
T31I - Permit E1:N/E/D/B/X to DES DKYGENKY:DKYL1+DDATA	Name change	HCR77C1 OA55184	CSNBT31I	0174
T31I - Permit E2:N/X to DES DKYGENKY:DKYL0+DMAC	Name change	HCR77C1 OA55184	CSNBT31I	0175
T31I - Permit E2:N/X to DES DKYGENKY:DKYL1+DMAC	Name change	HCR77C1 OA55184	CSNBT31I	0176
T31I - Permit E3:N/E/D/B/G/X to DES ENCIPHER	Name change	HCR77C1 OA55184	CSNBT31I	0177
T31I - Permit E4:N/B/X to DES DKYGENKY:DKYL0+DDATA	Name change	HCR77C1 OA55184	CSNBT31I	0178
T31I - Permit E5:N/G/C/V/E/D/G/X to DES DKYGENKY:DKYL0+DMAC	Name change	HCR77C1 OA55184	CSNBT31I	0179
T31I - Permit E5:N/G/C/V/E/D/G/X to DES DKYGENKY:DKYL0+DDATA	Name change	HCR77C1 OA55184	CSNBT31I	017A
T31I - Permit E5:N/G/C/V/E/D/G/X to DES DKYGENKY:DKYL0+DEXP	Name change	HCR77C1 OA55184	CSNBT31I	017B
T31I - Permit V0/V1/V2:N to DES PINGEN/ PINVER	Name change	HCR77C1 OA55184	CSNBT31I	017C
T31X - Permit DES KEYGENKY: DUKPT to B0:N/X	Name change	HCR77C1 OA55184	CSNBT31I	0180
T31X - Permit DES MAC/MACVER: AMEX-CSC to C0:G/C/V	Name change	HCR77C1 OA55184	CSNBT31I	0181
T31X - Permit DES MAC/MACVER: CVV-KEYA to C0:G/C/V	Name change	HCR77C1 OA55184	CSNBT31I	0182
T31X - Permit DES MAC/MACVER: ANY-MAC to C0:G/C/V	Name change	HCR77C1 OA55184	CSNBT31I	0183
T31X - Permit DES DATA/DATAM/DATAMV to C0:G/C/V	Name change	HCR77C1 OA55184	CSNBT31I	0184
T31X - Permit DES ENCIPHER/DECIPHER/ CIPHER to D0:E/D/B	Name change	HCR77C1 OA55184	CSNBT31I	0185
T31X - Permit DES DATA to D0:E/D/B	Name change	HCR77C1 OA55184	CSNBT31I	0186

Table 10. Summary of new and changed CCA access controls (continued)

Access control	Description	FMID or APAR number	Services affected	Offset
T31X - Permit DES EXPORTER/OKEYXLAT to K0:E	Name change	HCR77C1 OA55184	CSNBT31I	0187
T31X - Permit DES IMPORTER/IKEYXLAT to K0:D	Name change	HCR77C1 OA55184	CSNBT31I	0188
T31X - Permit DES EXPORTER/OKEYXLAT to K1/K4:E	Name change	HCR77C1 OA55184	CSNBT31I	0189
T31X - Permit DES IMPORTER/IKEYXLAT to K1/K4:D	Name change	HCR77C1 OA55184	CSNBT31I	018A
T31X - Permit DES MAC/DATA/DATAM to M0:G/C	Name change	HCR77C1 OA55184	CSNBT31I	018B
T31X - Permit DES MACVER/DATA/DATAMV to M0:V	Name change	HCR77C1 OA55184	CSNBT31I	018C
T31X - Permit DES MAC/DATA/DATAM to M1:G/C	Name change	HCR77C1 OA55184	CSNBT31I	018D
T31X - Permit DES MACVER/DATA/DATAMV to M1:V	Name change	HCR77C1 OA55184	CSNBT31I	018E
T31X - Permit DES MAC/DATA/DATAM to M3:G/C	Name change	HCR77C1 OA55184	CSNBT31I	018F
T31X - Permit DES MACVER/DATA/DATAMV to M3:V	Name change	HCR77C1 OA55184	CSNBT31I	0190
T31X - Permit DES OPINENC to P0:E	Name change	HCR77C1 OA55184	CSNBT31I	0191
T31X - Permit DES IPINENC to P0:D	Name change	HCR77C1 OA55184	CSNBT31I	0192
T31X - Permit DES PINVER: NO-SPEC to V0:N/V	Name change	HCR77C1 OA55184	CSNBT31I	0193
T31X - Permit DES PINGEN: NO-SPEC to V0:N/C	Name change	HCR77C1 OA55184	CSNBT31I	0194
T31X - Permit DES PINVER: NO-SPEC/IBM-PIN/IBM-PINO to V1:N/V	Name change	HCR77C1 OA55184	CSNBT31I	0195
T31X - Permit DES PINGEN: NO-SPEC/IBM-PIN/IBM-PINO to V1:N/V	Name change	HCR77C1 OA55184	CSNBT31I	0196
T31X - Permit DES PINVER: NO-SPEC/VISA-PVV to V2:N/V	Name change	HCR77C1 OA55184	CSNBT31I	0197
T31X - Permit DES PINGEN: NO-SPEC/VISA-PVV to V2:N/C	Name change	HCR77C1 OA55184	CSNBT31I	0198
T31X - Permit DES DKYGENKY: DKYL0+DMAC to E0:N/X	Name change	HCR77C1 OA55184	CSNBT31I	0199
T31X - Permit DES DKYGENKY: DKYL0+DMV to E0:N/X	Name change	HCR77C1 OA55184	CSNBT31I	019A

Table 10. Summary of new and changed CCA access controls (continued)

Access control	Description	FMID or APAR number	Services affected	Offset
T31X - Permit DES DKYGENKY: DKYL0+DALL to E0:N/X	Name change	HCR77C1 OA55184	CSNBT31I	019B
T31X - Permit DES DKYGENKY: DKYL1+DMAC to E0:N/X	Name change	HCR77C1 OA55184	CSNBT31I	019C
T31X - Permit DES DKYGENKY: DKYL1+DMV to E0:N/X	Name change	HCR77C1 OA55184	CSNBT31I	019D
T31X - Permit DES DKYGENKY: DKYL1+DALL to E0:N/X	Name change	HCR77C1 OA55184	CSNBT31I	019E
T31X - Permit DES DKYGENKY: DKYL0+DDATA to E1:N/X	Name change	HCR77C1 OA55184	CSNBT31I	019F
T31X - Permit DES DKYGENKY: DKYL0+DMPIN to E1:N/X	Name change	HCR77C1 OA55184	CSNBT31I	01A0
T31X - Permit DES DKYGENKY: DKYL0+DALL to E1:N/X	Name change	HCR77C1 OA55184	CSNBT31I	01A1
T31X - Permit DES DKYGENKY: DKYL1+DDATA to E1:N/X	Name change	HCR77C1 OA55184	CSNBT31I	01A2
T31X - Permit DES DKYGENKY: DKYL1+DMPIN to E1:N/X	Name change	HCR77C1 OA55184	CSNBT31I	01A3
T31X - Permit DES DKYGENKY: DKYL1+DALL to E1:N/X	Name change	HCR77C1 OA55184	CSNBT31I	01A4
T31X - Permit DES DKYGENKY: DKYL0+DMAC to E2:N/X	Name change	HCR77C1 OA55184	CSNBT31I	01A5
T31X - Permit DES DKYGENKY: DKYL0+DALL to E2:N/X	Name change	HCR77C1 OA55184	CSNBT31I	01A6
T31X - Permit DES DKYGENKY: DKYL1+DMAC to E2:N/X	Name change	HCR77C1 OA55184	CSNBT31I	01A7
T31X - Permit DES DKYGENKY: DKYL1+DALL to E2:N/X	Name change	HCR77C1 OA55184	CSNBT31I	01A8
T31X - Permit DES DATA/DATAM/CIPHER/MAC/ENCIPHER to E3:N/G/E/X	Name change	HCR77C1 OA55184	CSNBT31I	01A9
T31X - Permit DES DKYGENKY: DKYL0+DDATA to E4:N/X	Name change	HCR77C1 OA55184	CSNBT31I	01AA
T31X - Permit DES DKYGENKY: DKYL0+DALL to E4:N/X	Name change	HCR77C1 OA55184	CSNBT31I	01AB
T31X - Permit DES DKYGENKY: DKYL0+DEXP to E5:N/X	Name change	HCR77C1 OA55184	CSNBT31I	01AC
T31X - Permit DES DKYGENKY: DKYL0+DMAC to E5:N/X	Name change	HCR77C1 OA55184	CSNBT31I	01AD
T31X - Permit DES DKYGENKY: DKYL0+DDATA to E5:N/X	Name change	HCR77C1 OA55184	CSNBT31I	01AE

Table 10. Summary of new and changed CCA access controls (continued)

Access control	Description	FMID or APAR number	Services affected	Offset
T31X - Permit DES DKYGENKY:DKYL0+DALL to E5:N/X	Name change	HCR77C1 OA55184	CSNBT31I	01AF
T31X - Permit DES PINGEN to V0:N and DES PINVER to V1/V2:N	Name change	HCR77C1 OA55184	CSNBT31I	01B0
T31X - Permit AES CIPHER to D0:E/D/B	New	HCR77C1 OA55184	CSNBT31X	01D0
T31X - Permit AES MAC: CMAC to M6:G/C/V	New	HCR77C1 OA55184	CSNBT31X	01D1
T31X - Permit AES PINPROT to P0:E/D	New	HCR77C1 OA55184	CSNBT31X	01D2
T31X - Permit AES EXPORTER to K0:E	New	HCR77C1 OA55184	CSNBT31X	01D3
T31X - Permit AES EXPORTER to K1:E	New	HCR77C1 OA55184	CSNBT31X	01D4
T31X - Permit AES EXPORTER to K4:E	New	HCR77C1 OA55184	CSNBT31X	01D5
T31X - Permit AES IMPORTER to K0:D	New	HCR77C1 OA55184	CSNBT31X	01D6
T31X - Permit AES IMPORTER to K1:D	New	HCR77C1 OA55184	CSNBT31X	01D7
T31X - Permit AES IMPORTER to K4:D	New	HCR77C1 OA55184	CSNBT31X	01D8
T31X - Permit AES DKYGENKY:D-ALL/DMAC to E0:X	New	HCR77C1 OA55184	CSNBT31X	01D9
T31X - Permit AES DKYGENKY:D-ALL/DCIPHER to E1:X	New	HCR77C1 OA55184	CSNBT31X	01DA
T31X - Permit AES DKYGENKY:D-ALL/D-MAC to E2:X	New	HCR77C1 OA55184	CSNBT31X	01DB
T31X - Permit AES CIPHER to E3:E/B,DKYGENKY:D-ALL/DCIP to E3:X	New	HCR77C1 OA55184	CSNBT31X	01DC
T31X - Permit AES DKYGENKY:D-ALL/D-CIPHER to E4:X	New	HCR77C1 OA55184	CSNBT31X	01DD
T31X - Permit AES DKYGENKY:D-MAC to E5:X	New	HCR77C1 OA55184	CSNBT31X	01DE
T31I - Permit D0:E/D/B to AES CIPHER:ENC/DEC/ENC+DEC	New	HCR77C1 OA55184	CSNBT31I	01E0
T31I - Permit M6:G/C/V to AES MAC:CMAC+GENONLY/GEN/VER	New	HCR77C1 OA55184	CSNBT31I	01E1
T31I - Permit P0:E/D to AES PINPROT:ENC/DEC+CBC+ISO-4	New	HCR77C1 OA55184	CSNBT31I	01E2

Table 10. Summary of new and changed CCA access controls (continued)

Access control	Description	FMID or APAR number	Services affected	Offset
T31I - Permit K0:E to AES EXPORTER	New	HCR77C1 OA55184	CSNBT31I	01E3
T31I - Permit K0:D to AES IMPORTER	New	HCR77C1 OA55184	CSNBT31I	01E4
T31I - Permit K1/K4:E to AES EXPORTER:EXPTT31D+VARDRV-D	New	HCR77C1 OA55184	CSNBT31I	01E5
T31I - Permit AES K1/K4:D to AES IMPORTER:IMPTT31D+VARDRV-D	New	HCR77C1 OA55184	CSNBT31I	01E6
T31I - Permit E0:X to AES DKYGENKY:DKYL0/L1/L2+D-MAC+GEN+CMAC	New	HCR77C1 OA55184	CSNBT31I	01E7
T31I - Permit E1:X to AES DKYGENKY:DKYL0/L1/L2+D-SECMSG+SMPIN	New	HCR77C1 OA55184	CSNBT31I	01E8
T31I - Permit E2:X to AES DKYGENKY:DKYL0/L1/L2+D-MAC+GEN+CMAC	New	HCR77C1 OA55184	CSNBT31I	01E9
T31I - Permit E3:X to AES DKYGENKY:D-CIPHER+ENC+DEC+CBC	New	HCR77C1 OA55184	CSNBT31I	01EA
T31I - Permit E3:E/B to AES CIPHER:ENCRYPT/ENC+DEC	New	HCR77C1 OA55184	CSNBT31I	01EB
T31I - Permit E4:X to AES DKYGENKY:DKYL0/L1/L2+D-CIPHER+ENC+DEC	New	HCR77C1 OA55184	CSNBT31I	01EC
T31I - Permit E5:X to AES DKYGENKY:DKYL0/L1/L2/D-MAC+GEN+CMAC	New	HCR77C1 OA55184	CSNBT31I	01ED
T31X - Permit version D TR-31 key blocks	New	HCR77C1 OA55184	CSNBT31X	0382
T31X - Permit AES KDKGENKY: KDKTYPEA to 11:X	New	HCR77C1 OA55184	CSNBT31X	0383
T31X - Permit AES KDKGENKY: KDKTYPEB to 10:X	New	HCR77C1 OA55184	CSNBT31X	0384
T31X - Permit DES DKYGENKY: DKYL0:DMPIN to 12:X	New	HCR77C1 OA55184	CSNBT31X	0385
T31I - Permit version D TR-31 key blocks	New	HCR77C1 OA55184	CSNBT31I	0386
Encrypted PIN Translate2 – Permit ISO-4 to ISO-4 Translate	New	HCR77C1 OA55184	CSNBPTR2	038A
Encrypted PIN Translate2 – Permit ISO-4 Reformat with PAN Change	New	HCR77C1 OA55184	CSNBPTR2	038B
Encrypted PIN Translate2 – Permit ISO-4 to ISO-4 Reformat	New	HCR77C1 OA55184	CSNBPTR2	038C
Encrypted PIN Translate2 – Permit ISO-1 to ISO-4 Reformat	New	HCR77C1 OA55184	CSNBPTR2	038D

Table 10. Summary of new and changed CCA access controls (continued)

Access control	Description	FMID or APAR number	Services affected	Offset
Encrypted PIN Translate2 – Permit ISO-4 to ISO-1 Reformat	New	HCR77C1 OA55184	CSNBPTR2	038E
Encrypted PIN Translate2 – Permit ISO-0 to ISO-4 Reformat	New	HCR77C1 OA55184	CSNBPTR2	038F
Encrypted PIN Translate2 – Permit ISO-4 to ISO-0 Reformat	New	HCR77C1 OA55184	CSNBPTR2	0390
Encrypted PIN Translate2 – REFORMAT	New	HCR77C1 OA55184	CSNBPTR2	0391
Encrypted PIN Translate2 – TRANSLATE	New	HCR77C1 OA55184	CSNBPTR2	0392
Encrypted PIN Translate2 – Permit ISO-1 to ISO-4 RFMT1TO4	New	HCR77C1 OA55184	CSNBPTR2	0393
Public Infrastructure Certificate	New	HCR77C1	CSNDPIC	0070
Public Infrastructure Certificate - PK10SNRQ	New	HCR77C1	CSNDPIC	007C
Allow weak wrapping of compliance-tagged keys by DES MK	New	HCR77C1	All callable services that use compliant tagged DES key tokens.	02EB
Authenticated Key Export - SETSNKEY	New	HCR77C1	CSNBSYD CSNBSYD1 CSNBSYE CSNBSYE1 CSNBFLD CSNBFLE CSNBKRR2	02F5
Authenticated Key Export - DRVTXKEY	New	HCR77C1	CSNBSYD CSNBSYD1 CSNBSYE CSNBSYE1 CSNBFLD CSNBFLE CSNBKRR2	02F6
Authenticated Key Export - EXPTSK	New	HCR77C1	CSNBSYD CSNBSYD1 CSNBSYE CSNBSYE1 CSNBFLD CSNBFLE CSNBKRR2	02F7
Key Translate2 – COMP-TAG	New	HCR77C1	CSNBKTR2	02F8
Key Translate2 - COMP-CHK	New	HCR77C1	CSNBKTR2	02F9

Table 10. Summary of new and changed CCA access controls (continued)				
Access control	Description	FMID or APAR number	Services affected	Offset
Digital Signature Verify - PKCS-PSS allow not exact salt length	New	HCR77C0	CSNDDSV	033B
Digital Signature Generate - PKCS-PSS allow small salt	New	HCR77C0	CSNDDSG	033C

X9.143 (TR-31) key block support

Starting with APAR OA61978 for z/OS V2R5 ICSF (FMID HCR77D2) and ICSF FMID HCR77D1, ICSF supports X9.143 (TR-31) key blocks as operational (internal) keys. Operational key blocks can be used as key identifiers for many callable services.

A new service is introduced to build skeleton key blocks and key blocks with a randomly generated key value: TR-31 Create (CSNBT31C and CSNET31C). TR-31 Translate (CSNBT31X and CSNET31X – formerly TR-31 Export) has been updated to export operational key blocks to external key blocks as well as translate between CCA key tokens and TR-31 key blocks. TR-31 Import (CSNBT31I and CSNET31I) has been updated to import external key blocks to operational keys.

Operational X9.143 key blocks can be stored in the CKDS. The large common record (KDSRL) format of the CKDS is required. The key blocks will be encrypted under the current master key. See “The Cryptographic Key Data Set (CKDS)” on page 4 for more information. To convert your current CKDS to the large common record format, see “Migrating to the common record format (KDSR) key data set” on page 90 for more information. Labels must be unique for all X9.143 (TR-31) key blocks.

Note: Support for the KDSRL format CKDS requires z/OS V2R5 ICSF (FMID HCR77D2) or later.

Ensure the expected P11 master key support is available

ICSF introduced support for the Enterprise PKCS #11 (EP11) coprocessor and its associated P11 master key with FMID HCR77A0. ICSF uses the master key validation pattern (MKVP) in the header record of the TKDS to determine which EP11 coprocessors to make active. In FMID HCR77A0, an EP11 coprocessor was considered "active" if the MKVP in the current master key register matched the MKVP in the header record of the TKDS. If the MKVP did not match, or if the TKDS was never initialized, the EP11 coprocessor was considered "online", usable only for a limited number of non-secure key PKCS #11 services.

Starting with FMID HCR77A1, the online status no longer exists. Coprocessors are either active or in some error state. If the TKDS has been initialized, then any EP11 coprocessor that does not have a current master key register MKVP that matches the TKDS is not made active and, thus, not usable. Note, however, if the TKDS has not been initialized, then all EP11 coprocessors will be made active even though they would only be usable for non-secure key PKCS #11 services.

KGUP

Two optional controls for KGUP control statement processing were introduced in ICSF FMID HCR77D0. The controls are enabled by XFACILIT class profiles.

The verb authority control is used to restrict the use of KGUP control statement verbs. The control is enabled by creating the CSF.KGUP.VERB.AUTHORITY.CHECK profile for the XFACILIT class.

The CSFKEYS authority control enables SAF checking of all labels referenced in KGUP control statements against the profiles in the CSFKEYS class. The control is enabled by creating the CSF.KGUP.CSFKEYS.AUTHORITY.CHECK profile in the XFACILIT class. In addition, the key store policy granular key access control setting is enforced if enabled and the SAF profile prefixing is enforced if enabled.

For additional details, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

Key store policy

An optional control for the use of archived keys in the CKDS and objects in the TKDS is introduced in z/OS V2R5 (FMID HCR77D2). ICSF administrators can allow the use of an archived key in a key data set by defining the CSF.KDS.KEY.ARCHIVE.USE XFACILIT SAF profile. If the profile does not exist, all services that attempt to use an archived key fail with return code 8 reason code X'D10' (3344).

For services that do data encryption, the CSF.KDS.KEY.ARCHIVE.DATA.DECRYPT XFACILIT SAF profile restricts the use of archived data-encryption keys to the decryption of data only.

These services fail with return code 8 reason code X'D5E' (3422) when an archived key in a key data set is referenced as the key identifier and the CSF.KDS.KEY.ARCHIVE.DATA.DECRYPT XFACILIT SAF profile exists. The actual key token in the KDS record is not examined for suitability for the service's operation.

- Encipher (CSNBENC, CSNEENC, CSNBENC1, and CSNEENC1).
- Ciphertext Translate2 (CSNBCTT2, CSNECTT2, CSNBCTT3, and CSNECTT3):
 - Outbound key identifier.
- Symmetric Algorithm Encipher (CSNBSAE, CSNESAE, CSNBSAE1, and CSNESAE1).
- Symmetric Key Encipher (CSNBSYE, CSNESYE, CSNBSYE1, and CSNESYE1).
- Field Level Encipher (CSNBFLE and CSNEFLE).
- FPE Encipher (CSNBFPEE and CSNEFPEE).
- FPE Translate (CSNBFPET and CSNEFPET):
 - Outbound key identifier.
- Format Preserving Algorithms Encipher (CSNBFFXE and CSNEFFXE).
- Format Preserving Algorithms Translate (CSNBFFXT and CSNEFFXT):
 - Outbound key identifier.
- PKCS #11 Secret Key Encrypt (CSFPSKE and CSFPSKE6).
- PKCS #11 Secret Key Reencrypt (CSFPSKR and CSFPSKR6).

These services allow the use of a data-encryption key when an archived key in a key data set is referenced as the key identifier and the CSF.KDS.KEY.ARCHIVE.DATA.DECRYPT XFACILIT SAF profile exists. The actual key token in the KDS record is not examined for suitability for the service's operation.

- Decipher (CSNBDEC, CSNEDEC, CSNBDEC1, and CSNEDEC1).
- Ciphertext Translate2 (CSNBCTT2, CSNECTT2, CSNBCTT3, and CSNECTT3):
 - Inbound key identifier.
- Symmetric Algorithm Decipher (CSNBSAD, CSNESAD, CSNBSAD1, and CSNESAD1).
- Symmetric Key Decipher (CSNBSYD, CSNESYD, CSNBSYD1, and CSNESYD1).
- Field Level Decipher (CSNBFLD and CSNEFLD).
- FPE Decipher (CSNBFPED and CSNEFPED).
- FPE Translate (CSNBFPET and CSNEFPET):
 - Inbound key identifier.
- Format Preserving Algorithms Decipher (CSNBFFXD and CSNEFFXD).
- Format Preserving Algorithms Translate (CSNBFFXT and CSNEFFXT):
 - Inbound key identifier.
- PKCS #11 Secret Key Decrypt (CSFPSKD and CSFPSKD6).
- PKCS #11 Secret Key Reencrypt (CSFPSKR and CSFPSKR6).

All other services are allowed to use an archived key when the CSF.KDS.KEY.ARCHIVE.DATA.DECRYPT XFACILIT SAF profile exists.

CKDS Key Record Read2 (CSNBKRR2 and CSNEKRR2) service returns an archived key token when the CSF.KDS.KEY.ARCHIVE.USE profile exists. When the CSF.KDS.KEY.ARCHIVE.DATA.DECRYPT profile exists, the PROTKEY rule array keyword was specified, and a data-encryption key is retrieved, return code 0 reason code X'D5F' (3423) is returned indicating that the key should be used for decryption only.

DES keys

Enhanced wrapping support

WRAPENH3

This method is based on the enhanced wrapping method using SHA-256 with the addition of an authentication code. The wrapping and MAC keys are derived using the NIST KDF with SHA-256 HMAC. A TDES-CMAC authentication code is generated over the complete key token. The authentication code is stored in the token where the right control vector was stored. There will always be three key parts encrypted and placed in the key token to obfuscate the key length. All keys with the exception of DATA keys with a zero control vector can be wrapped with this method. The method is introduced by APAR OA60318 for ICSF FMID HCR77D1 and later releases and licensed internal code for IBM z13 and later hardware.

Notes on NOCV key-encrypting keys:

- The entire 64-byte key token is used when the authentication code is generated including flag byte 1. This means that the NOCV KEK bit cannot be enabled after the key has been wrapped with the WRAPENH3 method.
- The Key Generate service can generate NOCV KEKs by using a skeleton token with the NOCV KEK flag enabled.
 - The *key_type_1* and *key_type_2* parameters are TOKEN.
 - The *key_form* parameter is OPEX.
 - The *generated_key_identifier_1* parameter must contain a skeleton internal token of an IMPORTER or EXPORTER with the NOCV KEK flag enabled. Any wrapping method can be specified.
- The Key Generator Utility Program (KGUP) can be used to import and generate WRAPENH3 NOCV key-encrypting keys.
- The Key Import services enables the NOCV KEK bit after the token is returned from the Crypto Express adapter. If the token is wrapped with the WRAPENH3 method, the NOCV KEK bit will not be enabled and the request will fail with return code 8 reason code 3E90 (16016).
- Operational Key Load utility supports NOCV KEKs with the WRAPENH3 wrapping method.
- The Multiple Secure Key Import service has a rule array keyword that allows a WRAPENH3 key to be marked NOCV KEK within the Crypto Express adapter.
- To import or generate a NOCV key-encrypting key, initially wrap the key with the WRAP-ENH or WRAPENH2 method, store the key in the CKDS, and run the CSFCNV2 utility to convert the NOCV KEK to the WRAPENH3 method.

Triple-length key support

Additional triple-length DES key support is introduced by APAR OA55184 for ICSF FMID HCR77C1 and later releases and licensed internal code for IBM z13, IBM z13s, IBM z14, and later servers. In general, any service where a double-length key can be used, a triple-length key can be used as well. The service description should be checked for any restrictions.

Attention: If you are using triple-length keys including zero control vector DATA keys, you should be using triple-length key-encrypting keys. If you are still using double-length key-encrypting keys to wrap triple-length DATA keys, you need to start using triple-length key-encrypting keys.

NOCV Key-encrypting keys

DES NOCV key-encrypting keys are used to export and import keys where the external token has no control vector (a zero control vector is used). This allows communication with non-CCA crypto providers.

Starting with APAR OA55184 for ICSF FMID HCR77C1 and later releases and licensed internal code for IBM z13, IBM z13s, IBM z14, and later hardware, any IMPORTER or EXPORTER can be a NOCV KEK when the control vector is the default control vector with these exceptions:

- The form bits may be any value other than single length key (000).
- The ENH-ONLY (bit 56) attribute may be enabled. For triple-length keys, the ENH-ONLY attribute is enabled.

ICSF key data sets

CKDS

There are four formats of the CKDS:

- Large common record format (KDSRL) that is common to all key data sets.
- Common record format (KDSR) that is common to all key data sets.
- Variable-length record format.
- Fixed-length record format.

Both common record formats for the CKDS support all symmetric key tokens and provides support for metadata for each record including tracking usage of the records. See [“Migrating to the common record format \(KDSR\) key data set” on page 102](#) for more information.

The variable length record format is only required if variable-length key tokens are to be stored in the CKDS. All fixed-length and variable-length symmetric key tokens can be stored in the variable-length record format CKDS. See [“Migrating the CKDS to support variable-length symmetric key tokens” on page 101](#) for more information.

In addition to supporting all symmetric key tokens, the KDSR format CKDS provides support for metadata for each record including tracking usage of the records. See [“Migrating to the common record format \(KDSR\) key data set” on page 102](#) for more information.

When new key types are added to the CKDS, the following consideration applies when sharing the CKDS:

- When clear DES or AES keys are added to the CKDS, protect all clear DES and AES keys by label name using CSFKEYS profiles on all systems sharing the CKDS.

If you have no coprocessor, you can initialize the CKDS for use with clear AES and DES data keys. This CKDS cannot be used on a system with cryptographic coprocessors.

Migrating the CKDS to support variable-length symmetric key tokens

If variable-length symmetric key tokens are to be stored in the CKDS, a fixed-length record CKDS must be converted to the common record format or the variable-length record format.

To convert to the KDSR format, see [“Migrating to the common record format \(KDSR\) key data set” on page 102](#) for more information.

To convert to the variable-length format, ICSF provides the conversion utility program, CSFCNV2, that converts a fixed-length CKDS to the variable length format. See [Chapter 7, “Converting a CKDS from fixed length to variable length record format,” on page 223](#) for more information.

There is no reason to migrate a variable length record CKDS if your applications are not using AES or HMAC keys in variable-length tokens. You can migrate to the common record format at any time.

To migrate to the variable-length format CKDS:

1. Allocate a new CKDS with the variable length record format. The new CKDS should be large enough to hold all key in the current CKDS.

2. Disable dynamic CKDS updates on all systems.
3. Run the CKDS Conversion2 utility to convert the existing CKDS records to the new record format
4. Refresh the new CKDS on all systems that are sharing the CKDS
5. Enable dynamic CKDS updates on all systems

PKDS

There are three formats of the PKDS:

- Large common record format (KDSRL) that is common to all key data sets.
- Common record format (KDSR) that is common to all key data sets.
- Base record format.

Both KDSR and base formats use the same LRECL and KDSRL uses a larger LRECL. The common record formats provides support for metadata for each record including tracking usage of the record. To convert the original format PKDS to common record (KDSR) format, see [“Migrating to the common record format \(KDSR\) key data set” on page 102.](#)

TKDS

There are two formats of the TKDS: common record format (KDSR) and the base record format. Both formats use the same LRECL. The KDSR format provides support for metadata for each record including tracking usage of the record. To convert the original format TKDS to common record (KDSR) format, see [“Migrating to the common record format \(KDSR\) key data set” on page 102.](#)

For secure PKCS #11 support (Enterprise PKCS #11), the TKDS must be initialized with the PKCS #11 master key (P11-MK). Support to INITIALIZE TKDS and UPDATE TKDS is available in the Master Key Management Panels.

For information on managing and sharing the TKDS in a sysplex environment, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

Access authorization of the new callable services will be determined via SAF calls. No support will be provided for invocation of an installation security exit for these new services. The CSFSERV class controls access to the ICSF PKCS #11 callable services.

Migrating to the common record format (KDSR) key data set

All key data sets can be converted to KDSR (or KDSRL) format.

The conversion is done with the active key data set, and a new key data set with the proper attributes for the KDSR or KDSRL format must be allocated. Note that the TKDS is always in the large common record format.

The conversion can be done by either calling the CSFCRC callable service or by using the ICSF panels. While the conversion is happening, all updates to the key data set being converted are suspended. At the end of the conversion, all systems in the sysplex sharing the key data set will be using the KDSR format key data set as the active key data set. All new updates are made to the KDSR format key data set.

Converting to KDSR format using the CSFCRC callable service

An application must be written to invoke the CSFCRC callable service in order to convert a key data set to KDSR format. See [z/OS Cryptographic Services ICSF Application Programmer's Guide](#) for details about the CSFCRC callable service.

Converting to KDSR format using the ICSF panels

To convert a key data set to KDSR format using the ICSF panels, do the following:

1. Use either ICSF sample CSFCKDS (KDSR/KDSRL) or CSFCKD2 (KDSR/KDSRL for use in GDPS Active-Active) to create an empty KDS for the new KDS. Additionally, if you plan on specifying a backup KDS, use the same sample to create that KDS.
2. On the ICSF Primary Menu panel, select option 2, KDS MANAGEMENT, and press ENTER.
3. When the ICSF Key Data Set Management panel appears, select the type of key data set you want to convert to KDSR format and press ENTER.
4. On the next panel, select the COORDINATED xKDS CONVERSION option and press ENTER.
5. When the ICSF Coordinated KDS conversion panel appears, fill in the required fields and press ENTER.

Migrating to 24-byte DES master key

ICSF and TKE accept a 16-byte key value for the DES master key. CCA coprocessors with the September 2012 licensed internal code (LIC) or later installed on a CEX3C or later will support both a 16- and 24-byte key value. ICSF and TKE will support loading both key value lengths.

To load a 24-byte DES master key, the **DES master key – 24-byte key** access control point must be enabled in the ICSF role in all CCA coprocessors for the domain where you wish to use a 24-byte DES master key. If the **DES master key – 24-byte key** access control point is not enabled consistently for all coprocessors available to a instance of ICSF, the DES new master key register cannot be loaded. The master key entry utility will fail. A TKE workstation is required to enable the access control point.

It is not possible to share a CKDS between systems with both 16- and 24-byte DES master keys. The master key verification pattern algorithm for the 24-byte DES master key is different from the algorithm for the 16-byte master key. The algorithms are described in the [z/OS Cryptographic Services ICSF Administrator's Guide](#).

The CKDS Reencipher and Symmetric Change Master Key utilities support both length key values. The coordinated CKDS administration functions support both length key values. The Passphrase KDS Initialization utility will load a 24-byte DES master key if the **DES master key – 24-byte key** access control point is enabled.

Warning: Due to control block changes required to support the 24-byte DES master key, after a 24-byte DES master key has been loaded, the LIC cannot be changed to an earlier version that does not support the 24-byte DES master key. If a change to an earlier LIC is required, all DES master keys must be changed back to 16-byte keys. This can be done using symmetric change master key.

Installation options data set

- MASTERKCVLEN - Control the number of hexadecimal digits displayed for the CCA master keys on the ICSF Hardware Status panel.
- RNGCACHE - Controls whether ICSF maintains a cache of random numbers to be used by services that require them.
- TRACKCLASSUSAGE - Indicates to track the use of key records in a common record format key data set for a class of cryptographic operations.
- DEFAULTWRAP - WRAPENH3 added as a wrapping method option.

Function restrictions

Retained keys are RSA private keys that are stored in a cryptographic coprocessor instead of in the public key storage data set. This change does not affect retained keys that you are currently using, that is, keys that are stored on the cryptographic coprocessor. However, the ICSF services do not allow you to store in a cryptographic coprocessor RSA keys intended for key management use. Your applications can continue to store in the cryptographic coprocessor RSA private keys intended for signature usage. The modulus length of these private keys is limited to 2048-bits.

The 2048-bit RSA keys may have a public exponent, e , in the range of $1 < e < 2^{2048}$, and e must be odd. The RSA public key exponents for 2049-bit to 4096-bit RSA keys are restricted to the values 3 and

65537. The public exponent may be 5, 17, or 257 on an IBM z13, IBM z13s, or later server with the October 2016 or later licensed internal code.

CICS attachment facility

If you have the CICS Attachment Facility installed and you specify your own CICS wait list data set, you need to modify the wait list data set to include the new callable services.

Modify and include:

For FMID HCR77D2:

CSFT31C (APAR OA61978)

For FMID HCR77D1:

- CSFT31C (APAR OA61978)
- CSFPVR2, CSFPSKR (APAR OA61253).
- CSFFFXD, CSFFFXE, CSFFFXT (APAR OA59593).

For FMID HCR77D0:

CSFDCU2, CSFDRG2, CSFT34B, CSFT34C, CSFT34D, CSFT34R (APAR OA57089).

Note: If no Wait List is specified, the default wait list will be used. See sample CSFWTL01 for the contents of the default wait list.

Dynamic LPA load

ICSF uses dynamic LPA to load the pre-PC routines, CICS related routines, and other modules which must reside in common storage into above-the-line ECSA. The dynamic LPA load will occur the first time that ICSF is started within an IPL, and the modules will persist across subsequent restarts of ICSF.

Dynamic service update

Dynamic service update allows you to apply service updates with minimal impact to ICSF availability. ICSF can activate service without a manual stop and start of ICSF. These updates include service updates as well as changes to the options data set that cannot be applied via the SETICSF OPT,REFRESH command. Additionally, dynamic service updates can be used to recycle ICSF when there are problems that are not resolving.

Before starting a dynamic service update, see [“Dynamic service update” on page 140](#).

Special secure mode

Use of some ICSF services requires that ICSF be in special secure mode: CSNBPGN, CSNBSKI, CSNBSKI2, and CSNBSKM.

Resource Measurement Facility (RMF)

Support to enable RMF to provide performance measurements on these selected ICSF services and functions. The measurements refer to these services processing on cryptographic coprocessors except for one-way hash. One-way hash is processed on CPACF.

- Decipher (CSNBDEC)
- Digital Signature Generate (CSNDDSG)
- Digital Signature Verify (CSNDDSV)
- Encipher (CSNBENC)
- Encrypted PIN Translate (CSNBPTR)
- Encrypted PIN Translate2 (CSNBPTR2)
- Encrypted PIN Translate Enhanced (CSNBPTRE)

- Encrypted PIN Verify (CSNBPVR)
- Encrypted PIN Verify2 (CSNBPVR2)
- Format Preserving Algorithms Decipher (CSNBFFXD)
- Format Preserving Algorithms Encipher (CSNBFFXE)
- Format Preserving Algorithms Translate (CSNBFFXT)
- FPE Decipher (CSNBPFED)
- FPE Encipher (CSNBPFEE)
- FPE Translate (CSNBPFET)
- MAC Generate (CSNBMGN)
- MAC Generate2 (CSNBMGN2)
- MAC Verify (CSNBMVR)
- MAC Verify2 (CSNBMVR2)
- One-Way Hash (CSNBOWH)
- PIN Verify (CSNBPVR)
- Symmetric Algorithm Decipher (CSNBSAD)
- Symmetric Algorithm Encipher (CSNBSAE)

System abend codes

A complete list of the reason codes for the ICSF abend (X'18F') is contained in [z/OS MVS System Codes](#), which is published on release boundaries. As a migration aid for FMID HCR77F0, new and changed codes for FMID HCR77F0 are listed here. Reason codes introduced in the previous deliverables, FMIDs HCR77D2, HCR77D1, HCR77D0, and HCR77E0 are also listed.

An 18F code indicates an abend from ICSF.

The following reason codes are no longer issued as of FMID HCR77D1:

- 4F (79)
- 426 (1062)

FMID HCR77D0 reason codes are as follows:

Code Hex (Dec)	Meaning
----------------	---------

48C (1164)	ASCRE failed during early ICSF processing. This abend results in a wait state (X'040').
-------------------	---

SMF records

SMF record information for ICSF is documented in [Appendix B, “ICSF SMF records,” on page 411](#). Refer there for information on SMF records.

TKE workstation

The Trusted Key Entry (TKE) workstation provided secure management of master and operational keys and management of access control points. Refer to [z/OS Cryptographic Services ICSF TKE Workstation User's Guide](#) for more information.

Access to callable services

Access to services that are executed on cryptographic coprocessors is through access control points in the Domain Role. To execute callable services on the coprocessor, access control points must be enabled for each service in the Role. For systems that do not use the optional TKE Workstation, all access

control points (current and new) are enabled in the role with the appropriate microcode level on the cryptographic coprocessor.

For TKE users who have modified the Domain Role, all new access control points must be enabled using the TKE workstation. For non-TKE users, all new access control points are enabled.

Note: Some access control points are disabled by default in the Coprocessor Role. See the ICSF Application Programmer's Guide and [z/OS Cryptographic Services ICSF Administrator's Guide](#) for these access control points. A TKE Workstation is required to enable these access control points

TKE enablement from the support element

You must enable TKE commands on each cryptographic coprocessor from the support element. This is true for new TKE users and those upgrading their level of LIC. See Support Element Operations Guide and [z/OS Cryptographic Services ICSF TKE Workstation User's Guide](#) for more information.

Enabling access control points for PKCS #11 coprocessor firmware

A new or a zeroized Enterprise PKCS #11 coprocessor (or domain) comes with an initial set of Access Control Points (ACPs) that are enabled by default. All other ACPs, representing potential future support, are left disabled. When a firmware upgrade is applied to an existing Enterprise PKCS #11 coprocessor, the upgrade may introduce new ACPs. The firmware upgrade does not retroactively enable these ACPs, so they are disabled by default. These ACPs must be enabled via the TKE (or subsequent zeroize) in order to utilize the new support they govern. See Table 28. PKCS #11 Access Control Points in *Writing PKCS #11 Applications* for a complete description of the Access Control Points.

Table 11. Mapping of Enterprise PKCS #11 ACPs to firmware levels

Enterprise PKCS #11 firmware level	ACPs supported at this level	ACPs that need to be enabled when this code level is obtained via firmware upgrade
Initial release	<p>Control Point Management</p> <p>Allow addition (activation) of Control Points(0)</p> <p>Allow removal (deactivation) of Control Points(1)</p> <p>Cryptographic Operations</p> <p>Sign with private keys(2)</p> <p>Sign with HMAC or CMAC(3)</p> <p>Verify with HMAC or CMAC(4)</p> <p>Encrypt with symmetric keys(5)</p> <p>Decrypt with private keys(6)</p> <p>Decrypt with private keys(7)</p> <p>Key export with public keys(8)</p> <p>Key export with symmetric keys(9)</p> <p>Key import with private keys(10)</p> <p>Key import with symmetric keys(11)</p> <p>Generate asymmetric key pairs(12)</p> <p>Generate symmetric keys(13)</p> <p>Cryptographic Algorithms</p> <p>RSA private-key use(30)</p> <p>DSA private-key use(31)</p> <p>EC private-key use(32)</p> <p>Brainpool (E.U.) EC curves(33)</p> <p>NIST/SECG EC curves(34)</p> <p>Allow non-BSI algorithms (as of 2009) (21)</p> <p>Allow non-FIPS-approved algorithms (as of 2011) (35)</p> <p>Allow non-BSI algorithms (as of 2011) (36)</p> <p>Key Size</p> <p>Allow 80 to 111-bit algorithms(24)</p> <p>Allow 112 to 127-bit algorithms(25)</p> <p>Allow 128 to 191-bit algorithms(26)</p> <p>Allow 192 to 255-bit algorithms(27)</p> <p>Allow 256-bit algorithms(28)</p> <p>Allow RSA public exponents below 0x10001(29)</p> <p>Miscellaneous</p> <p>Allow backend to save semi-retained keys not applicable(14)</p> <p>Allow keywrap without attribute-binding(16)</p> <p>Allow changes to key objects (usage flags only) (17)</p> <p>Allow mixing external seed to RNG not applicable(18)</p> <p>Allow non-administrators to mark key objects TRUSTED(37)</p> <p>Do not double-check sign/decrypt operations(38)</p> <p>Allow dual-function keys - key wrapping and data encryption(39)</p> <p>Allow dual-function keys - digital signature and data encryption(40)</p> <p>Allow dual-function keys - key wrapping and digital signature(41)</p> <p>Allow non-administrators to mark public key objects ATTRBOUND(42)</p> <p>Allow clear passphrases for password-based-encryption(43)</p> <p>Allow wrapping of stronger keys by weaker keys(44)</p> <p>Allow clear public keys as non-attribute bound wrapping keys(45)</p>	None - all default ACPs enabled in the initial release.
Version 2 Sept. 2013 or later licensed internal code (LIC)	<p>Set for initial release plus</p> <p>Cryptographic Operations</p> <p>Allow key derivation (47)</p> <p>Cryptographic Algorithms</p> <p>DH Private Key Use (46)</p>	<p>Cryptographic Operations</p> <p>Allow key derivation (47)</p> <p>Cryptographic Algorithms</p> <p>DH Private Key Use (46)</p>

Migrating to PCI-HSM 2016 compliance mode

Beginning with the Crypto Express6 adapter, when configured as a CCA coprocessor, the CCA coprocessor is capable of running in a compliance mode. In order for the requirements of PCI-HSM 2016 to apply to a workload, the workload must be using compliant-tagged key tokens. Therefore, migrating an application to PCI-HSM 2016 involves converting the key tokens that are used to compliant-tagged key tokens.

Compliance warnings

ICSF has support for generating warning events for operations that might need modifications to meet requirements for the request to be compliant. The compliance warning event indicates whether the request was compliant or not.

- When the request is compliant, the key tokens that are used can be converted to compliant-tagged key tokens and the operation would still be successful.
- When the request is non-compliant, the key tokens, the service, or the service and rule combination must be updated to be compliant before the key tokens used can be converted to be compliant-tagged.

Warning events are generated for successful requests where at least one of the key tokens used can become compliant-tagged and none of the key tokens are already compliant-tagged. Key tokens that may become compliant-tagged include internal, DES version 00, internal AES version 04 (DATA), or version 05 key tokens, and internal RSA private key tokens with section identifier X'08', X'30', or X'31'. For services that do not accept compliant-tagged key tokens, only internal key tokens that are used as input to the service are included in the event. For services that accept compliant-tagged key tokens, all key tokens that are used, including output key tokens, are included in the event.

Warning events are in the form of SMF type 82 subtype 48 records. The generation of warning events is controlled by the COMPLIANCEWARN keyword in the ICSF installation options data set.

Migration process

The migration process completes with existing key tokens that are converted to being compliant-tagged. Before you complete the migration, it is important that the necessary steps are taken so that upon completion of the migration, future key tokens can be created as compliant-tagged. For information about creating compliant-tagged key tokens, see [z/OS Cryptographic Services ICSF Application Programmer's Guide](#).

The migration process requires a Crypto Express6 CCA Coprocessor or later with the July 2019 or later licensed internal code (LIC).

Identifying key tokens to be converted using compliance warning events

Except for the Cipher Text Translate2 (CSNBCTT2/CSNECTT2) service, a compliant-tagged key token cannot be used in a cryptographic operation with a non-compliant-tagged key token (Note: A compliant-tagged key token can be used with a compliant X.509 certificate). Therefore, with that one exception, the key migration process needs to ensure that for each key that is to be compliant-tagged, all keys it can be used with must also be compliant-tagged (or compliant in the case of an X.509 certificate). The compliance warning event includes a few pieces of information that can help uniquely identify keys: the key label, key fingerprint, and token identification value.

One procedure for migrating keys is to put them into one of the following five categories in the following order of precedence:

Category 1

Keys included in compliance warning events with any of the following results: Non-compliant service, Compliance not evaluated, or Non-compliant service operation. These keys are being used in non-compliant ways or ways where compliance is not supported. Before these keys can be compliant-tagged, there must be a change to use them in a supported, compliant way.

Category 2

Keys that are used with a key in Category 1 (above). Because keys in this category are used with keys that should not be compliant-tagged, by extension, they should also not be compliant-tagged.

Category 3

1. Keys included in compliance warning events with the result: Non-compliant key used and one of the following is true:
 - The compliance check (COMP_CHK) of the key indicates the key is non-compliant. The reason for the non-compliance needs to be determined and fixed before the key can be compliant-tagged.

- The compliance check (COMP_CHK) of the key indicates compliance unknown and you have determined that the key is non-compliant.

X.509 certificates are reported as compliance unknown if not running a CEX7C or later coprocessor with the June 2020 or later licensed internal code (LIC). In this case, you will need to determine the compliance of the certificate by other means.

Other keys may also be reported as unknown. In these cases, there are other keys used in the request that were reported as non-compliant. Once any non-compliant keys are fixed, the keys previously reported as unknown will report an actual compliance check value.

- The compliance check (COMP_CHK) of the key indicates the key was not evaluated and the key is an HMAC or ECC key. Currently, HMAC and ECC keys cannot be compliant-tagged so any key used with them also cannot be compliant-tagged.
2. Keys included in compliance warning events with the result: Compliant, and the compliance check (COMP_CHK) of the key does not indicate a compliant key. This key does not affect the compliance of this operation. However, if the key is to be compliant-tagged, the reason for the non-compliance needs to be determined and fixed.

Category 4

Keys that are used with a key in Category 3 (above). Because keys in this category are used with keys that cannot be compliant-tagged, by extension, these keys should not be compliant-tagged.

Category 5

Keys included in compliance warning events with the result: Compliant, and the compliance check (COMP_CHK) of the key indicates the key is compliant.

For information on interpreting the compliance warning events such as which services are compliant, which services are not compliant, which services do not support compliance, which service operations are not compliant, and what constitutes a compliant and non-compliant key token, see *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

As the issues in Categories 1 thru 4 above are resolved, you are left only with keys in Category 5. At this point, you can successfully compliant-tag all your keys. You might have the case where not all keys can become compliant-tagged (for example, when a key is used in a service that does not support compliance). In this case, you have a key or keys that are remaining in Category 1 and possibly also in Category 2 that should not be compliant-tagged. The keys in Category 5 can still be compliant-tagged. A similar situation arises when a key that is used in an otherwise compliant way is itself non-compliant. In this case, you have a key or keys that are remaining in Category 3 and possibly also in Category 4 that should not be compliant-tagged. Once again, the keys in Category 5 can still be compliant-tagged. These scenarios are not mutually exclusive so you might end up with keys in Categories 1 - 5 at migration time. If the categories are strictly adhered to, you can compliant-tag the keys in Category 5. This means that the workload would continue by using one or more non-compliant-tagged key tokens. If necessary, the key usage audit records (SMF type 82 subtype 44) can be used to identify how non-compliant-tagged key tokens are used by the application.

All requests that are selected for compliance warning processing get routed by ICSF to a CEX6C or later coprocessor. If a CEX5C is also available, depending on the nature of the workloads on your system, you might notice an imbalance in requests that are processed across the CCA cryptographic coprocessors.

Note: When an RSA private key token that is not an RSAAESM2 or RSAAESC2 key type is passed to the Symmetric Key Generate (CSNDSYG/CSNFSYG), Symmetric Key Export (CSNDSYX/CSNFSYX), Digital Signature Verify (CSNDDSV/CSNFDSV), or PKA Encrypt (CSNDPKE/CSNFPKE) callable service, ICSF builds a public key token internally and uses that in the operation. Because a public key token is not compliant, this operation will always result in a non-compliant SMF record. To avoid this non-compliant SMF record and ensure that once you compliant-tag your keys that they work successfully, you should do one of two things:

- Convert the private key tokens to the RSAAESM2 or RSAAESC2 format using the PKA Key Translate (CSNDPKT/CSNFPKT) callable service with the INTUSCHG keyword, or
- Change to using an X.509 certificate instead of a private key token.

Identifying key tokens outside of compliance warning events

Because compliance warnings are based on actual usage, be especially aware of operations (for example, infrequent operations) which might not be started during the period where warnings are being collected. These operations would not show up in a warning log and so must be discovered and analyzed independently.

It might be necessary to do an analysis of the CKDS and PKDS to identify the key tokens to be converted. If the key labels follow a naming convention, the Key Data Set List (CSFKDSL/CSFKDSL6) callable service can be used to produce a list of labels according to a filter. ICSF provides sample REXX programs (CSFCMPLC and CSFCMPLP) which list all tokens in the CKDS and PKDS respectively. It can be modified to produce a list of labels based on a filter. The samples use the Key Data Set List callable service and the listing can be modified by modifying the *label_filter* parameter.

You can also look at the profiles within the CSFKEYS SAF class that an application has access to as an indication of the key tokens the application can use.

Key tokens that are identified in this way do not have any information about how they are used so either the usage of such key tokens must be understood or all the key tokens that are used must be converted together.

If your CKDS contains DES KDF 01 or 02 token, the CKDS samples will help to identify and migrate them. DES KDF 01 or 02 tokens are tokens that were created as compliant-tagged using a coprocessor without the May 2021 or later licensed internal code (LIC). They are no longer considered compliant-tagged and should be migrated to become truly compliant-tagged. See [*z/OS Cryptographic Services ICSF Application Programmer's Guide*](#) for more information about DES KDF 01 or 02 tokens.

Ensure the key tokens identified can become compliant-tagged

When the key tokens to be converted have been identified, you need to ensure that the process of compliant-tagging the key tokens is successful. Compliant-tagged key tokens cannot be used with non-compliant-tagged key tokens so you do not want the conversion of some tokens to fail while others succeed. You do this by compliance-checking the key tokens first. Key tokens are compliance-checked by using the Key Translate2 (CSNBKTR2/CSNEKTR2) or the PKA Key Translate (CSNDPKT/CSNFPKT) callable service with the COMP-CHK keyword. To simplify this process, ICSF provides sample REXX programs (CSFCMPCC and CSFCMPCP) that compliance-check a list of CKDS and PKDS labels respectively. Any key token that cannot become compliant-tagged is identified in the output. For each such key, the cause of the compliance-check failure must be resolved before you attempt to convert the key tokens.

Converting key tokens to become compliant-tagged

When the key tokens to be converted have been identified, are not being used in a non-compliant way, and verified to be compliant, they can be converted to compliant-tagged key tokens. At this point, you must decide what backup strategy, if any, to pursue. Depending on the nature of the ICSF workloads, your backup strategy can include, but is not limited to:

1. Making a backup copy of the CKDS.
2. Retrieving the key tokens to be converted from the CKDS and storing them in a data set.

In lieu of doing a backup copy of the CKDS, you can opt to instead write the compliant-tagged key tokens to new CKDS labels.

Key tokens are converted to compliant-tagged tokens by using the Key Translate2 (CSNBKTR2/CSNEKTR2) or the PKA Key Translate (CSNDPKT/CSNFPKT) callable service with the COMP-TAG keyword. To simplify this process, ICSF provides sample REXX programs (CSFCMPTC and CSFCMPTP) that compliance-tags a list of CKDS and PKDS labels respectively. The samples can be modified to write the compliant-tagged key tokens to new labels instead of overwriting the original key tokens. To begin the conversion, at least one CCA coprocessor must be placed in migration mode by using the TKE workstation. To confirm the compliance mode of a CCA coprocessor, view the hardware status panel or issue the DISPLAY ICSF,CARDS command. Also, take note of the number of CCA coprocessors in PCI-HSM 2016 mode without being in migration mode. A coprocessor in migration mode cannot handle requests

that contain compliant-tagged key tokens. Therefore, if workloads that use compliant-tagged key tokens are already in use (for example, you previously converted some key tokens to compliant-tagged), you must keep one or more coprocessors in PCI-HSM 2016 compliance mode, but not in migration mode.

At this point in the process, none of the key tokens should fail because they were previously compliance-checked. However, if for some reason there is a failure such that some of the key tokens were converted and others were not, the key tokens should be brought back to a consistent state as soon as possible. This means that the key tokens should be updated such that they are all compliant-tagged or all non-compliant-tagged. If a backup copy of the KDS was made, it can be used to make all the key tokens non-compliant-tagged until the issue can be resolved. If the process involves creating compliant-tagged key tokens under new key labels, the old labels can still be used until the issue is resolved.

Chapter 4. Operating ICSF

You use certain commands to operate ICSF. Also, there are different conditions for operating ICSF that you should consider. This topic describes the ICSF operating tasks.

Manually starting and stopping ICSF

To ensure that ICSF is available as early as possible for use by critical system resources, ICSF should be configured to start automatically at IPL-time. For more details and instructions, see [“Starting ICSF during IPL-time”](#) on page 114.

Alternatively, to start ICSF manually, issue the operator START command. If you choose this method, you must issue the START command after each IPL and should include it as one of the first commands in COMMNDxx.

ICSF must be a started task and should be started with SUB=MSTR to eliminate any need to wait for JES. This also allows ICSF to be shut down after JES.

This example shows the format of the START command to start ICSF, assuming that CSF is the name of the start procedure:

```
START CSF,SUB=MSTR
```

To reuse ASIDs, the REUSASID parameter can be added to the START comment:

```
START CSF,SUB=MSTR,REUSASID=YES
```

To stop ICSF, issue the operator STOP command. After you issue the STOP command, all ICSF processing stops. If ICSF stops successfully, a message that states that ICSF is stopped appears on the console.

During shutdown, ideally ICSF is shut down after OMVS and JES are taken down. This allows any final updates to encrypted file systems to be successfully processed. By shutting down ICSF gracefully, it allows ICSF to complete all processing for updates to the key data sets.

This example shows the format of the STOP command to stop ICSF, assuming that CSF is the name of the started procedure:

```
STOP CSF
```

If ICSF is unresponsive to the STOP command, be aware that you are not able to use the CANCEL command to stop ICSF processing. Instead, use the force command:

```
FORCE csfproc,arm
```

For resiliency purposes, consider implementing an ARM (Automatic Restart Manager) policy to allow for the automatic restart of ICSF. For more information, see [“ARM policy”](#) on page 114.

Master key validation

When ICSF is started, the master keys are checked against the key data sets.

For CCA, master key verification patterns (MKVP) stored in the cryptographic key data set (CKDS) and the public key data set (PKDS) are compared to the current master keys. A CCA coprocessor becomes active if the current master keys match the MKVPs found in the CKDS and PKDS. If there is any mismatch, the coprocessor does not become active. When an MKVP is not in the CKDS or PKDS, the master key is ignored.

For an Enterprise PKCS #11 (EP11) coprocessor, ICSF uses the master key validation pattern (MKVP) in the header record of the TKDS to determine which EP11 coprocessors to make active. An EP11

coprocessor is active if the MKVP in the current master key register matched the MKVP in the header record of the TKDS or the TKDS has not been initialized.

When ICSF successfully starts, a message indicating that initialization is complete appears on the console.

Note:

1. If a problem is detected with a cryptographic coprocessor or with an accelerator during initialization, message CSFM540I is generated and the device is bypassed.
2. The ICSF_COPROCESSOR_STATE_NEGCHANGE health check monitors the state of the coprocessors and accelerators daily to detect a negative change in state. For more information about this health check, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).
3. The ICSF_MASTER_KEY_CONSISTENCY health check evaluates the master key states of the coprocessors to detect potential master key problems. For more information about this health check, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

ARM policy

ICSF now has a z/OS® Automatic Restart Manager (ARM) policy that can be defined using the ARM element “SYSICSF_*”. Whenever ARM is used to restart ICSF, message IXC812I is issued by ARM. When restarting ICSF with ARM, ICSF is started as a started task and is included in the output of the DISPLAY JOBS,LIST or DISPLAY A,LIST system commands. For information on setting up ARM, see [z/OS MVS Programming: Sysplex Services Guide](#).

Sample:

```
CSFARM1
//CSFARM1 JOB
//STEP1 EXEC PGM=IXCMIAPU,REGION=2M
//STEPLIB DD DSN=SYS1.MIGLIB,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSABEND DD SYSOUT=A
//SYSIN DD *

DATA TYPE(ARM)

DEFINE POLICY NAME(CSFPOL) REPLACE(YES)
RESTART_GROUP(ICSFGROUP)
TARGET_SYSTEM(*)
ELEMENT(SYSICSF_*)
RESTART_METHOD(BOTH,PERSIST)

/*ELEMENT NAME MUST BE SYSICSF_*
/*IN THIS SAMPLE ICSF WILL BE RESTARTED WITH THE
/*SAME JCL USED TO ORIGINALLY START IT.
```

Starting ICSF during IPL-time

In addition to starting ICSF manually, ICSF can be started automatically during IPL-time. Starting ICSF during IPL-time allows callers of ICSF to take advantage of ICSF functionality during IPL-time. This functionality is available on ICSF FMID HCR77C0 and later running on z/OS V2R3, with PTF for APAR OA55378 applied, and later.

Both the ICSFPROC and ICSF system parameters must be specified in order to start ICSF automatically during IPL-time. You can specify the values of the ICSFPROC and ICSF system parameters in one or more of the following places:

- The IEASYSxx parmlib member.
- By the operator, in response to message IEA101A SPECIFY SYSTEM PARAMETERS.

If you define the values in only the IEASYSxx parmlib member, the system uses that definition. Otherwise, the system determines the ICSFPROC and ICSF system parameters using the values specified via the operator response to message IEA101A SPECIFY SYSTEM PARAMETERS.

To configure ICSF to start during IPL-time:

1. Configure the ICSFPROC system parameter. The ICSFPROC system parameter specifies the ICSF startup procedure to be used during early ICSF initialization. ICSFPROC can be omitted or 'NONE' can be specified to prevent ICSF from starting early. If 'NONE' is specified, ICSF must be started manually. The procedure must reside in a SYS1.PROCLIB data set or an equivalent that is specified by the IEFPSDI DD card specification of the MSTJCLxx PARMLIB member. If the procedure is not in this location, ICSF will not start. For information about MSTJCL, see [z/OS MVS Initialization and Tuning Reference](#).

```
ICSFPROC=CSF2
```

```
ICSFPROC=NONE
```

2. Configure the ICSF system parameter. The ICSF system parameter specifies the xx value of the CSFPRMxx member containing the installation options data set. For example, a value of 00 would correspond to the CSFPRM00 member. ICSF can be omitted or 'NONE' can be specified to prevent ICSF from starting early. If 'NONE' is specified, ICSF must be started manually.

```
ICSF=00
```

```
ICSF=NONE
```

3. Modify the ICSF startup procedure. The ICSF startup procedure must be modified to accept the PRM procedure variable. The PRM procedure variable must be set to the xx value of the CSFPRMxx member containing the installation options data set. The following example shows how this would look using the CSFPARM DD statement:

```
//CSF PROC PRM=00  
//CSF EXEC PGM=CSFINIT,REGION=0M,TIME=1440,MEMLIMIT=NOLIMIT  
//CSFPARM DD DSN=USER.PARMLIB(CSFPRM&PRM),DISP=SHR
```

4. IPL the system. If both the ICSFPROC and ICSF system parameters are configured correctly and the ICSF startup procedure exists and is coded correctly, ICSF starts during IPL-time.

Notes:

- For information on the syntax of the ICSFPROC and ICSF system parameters, see IEASYSxx (system parameter list) in [z/OS MVS Initialization and Tuning Reference](#).
- For information on how to setup the ICSF startup procedure, see [“Steps to create the ICSF startup procedure” on page 28](#).
- It is recommended that you set up an AUTOR policy to auto reply to the BCF005A and BCF006A messages after a specified amount of time has passed. In the example below, ICSF must be started manually if the auto reply is NONE after 60 seconds.

```
MSGID(BCF005A) DELAY(60S) REPLY(NONE)
```

```
MSGID(BCF006A) DELAY(60S) REPLY(NONE)
```

- You should remove any existing invocations that start ICSF and rely on ICSF startup at IPL-time. For example, look for any commands that start ICSF in the COMMNDxx parmlib member. After the system brings up ICSF automatically, the system rejects any attempt to bring up a second instance of ICSF. The system issues the following warning message and terminates the second instance of ICSF:

```
CSFM004A ICSF TERMINATING. ICSF ALREADY ACTIVE.
```

- ICSF, when started during IPL-time, is started as a system address space. Any processing (including automation) that relies on ICSF being started as a job (started task) might need to make changes. For example, ICSF would not be included in the output of the DISPLAY JOBS,LIST or DISPLAY A,LIST system commands.

Note: ICSF address space is still included in the output of the DISPLAY JOBS,ALL and DISPLAY A,ALL system commands.

- For resiliency purposes, consider implementing an ARM (Automatic Restart Manager) policy to allow for the automatic restart of ICSF. For more information, see [“ARM policy” on page 114](#).

Retrieving a protected key early in IPL

In order to retrieve a protected key early in IPL, ICSF must be configured to start early as described in [“Starting ICSF during IPL-time” on page 114](#). The following restrictions also apply:

- The ICSF and ICSFPROC parameters must not be specified as a reply to messages BCF005A or BCF006A. These parameters must come from either the IEASYSxx parmlib member or in response to message IEA101A SPECIFY SYSTEM PARAMETERS.
- The CSFPRMxx parmlib member must specify the CKDSN keyword containing a valid CKDS name.
- The last occurrence of the CKDSN keyword is used. This is true even if the keyword is contained within a BEGIN-END block.

Modifying ICSF

When you issue the MODIFY command, ICSF gives control to the installation exit CSFEXIT5, if it exists. Your installation can write an exit routine for CSFEXIT5 that changes ICSF operations. For example, you might have the installation exit change the CHECKAUTH installation option without having to stop and restart ICSF. See [Chapter 5, “Installation exits,” on page 165](#) for a description of the installation exits.

If your installation does not write an exit routine for CSFEXIT5, no action occurs when you enter the MODIFY command.

Command syntax notation

You must follow certain syntactical rules when you code the ICSF commands.

How to read syntax diagrams

This section describes how to read syntax diagrams. It defines syntax diagram symbols, items that may be contained within the diagrams (keywords, variables, delimiters, operators, fragment references, operands) and provides syntax examples that contain these items.





Syntax diagrams pictorially display the order and parts (options and arguments) that comprise a command statement. They are read from left to right and from top to bottom, following the main path of the horizontal line.

For users accessing IBM Documentation using a screen reader, syntax diagrams are provided in dotted decimal format.

Symbols

The following symbols may be displayed in syntax diagrams:

Symbol	Definition
--------	------------

- | | |
|---|--|
|  | Indicates the beginning of the syntax diagram. |
|  | Indicates that the syntax diagram is continued to the next line. |
|  | Indicates that the syntax is continued from the previous line. |
|  | Indicates the end of the syntax diagram. |

Syntax items

Syntax diagrams contain many different items. Syntax items include:

- **Keywords** - a command name or any other literal information.
- **Variables** - variables are italicized, appear in lowercase, and represent the name of values you can supply.
- **Delimiters** - delimiters indicate the start or end of keywords, variables, or operators. For example, a left parenthesis is a delimiter.
- **Operators** - operators include add (+), subtract (-), multiply (*), divide (/), equal (=), and other mathematical operations that may need to be performed.
- **Fragment references** - a part of a syntax diagram, separated from the diagram to show greater detail.
- **Separators** - a separator separates keywords, variables or operators. For example, a comma (,) is a separator.

Note: If a syntax diagram shows a character that is not alphanumeric (for example, parentheses, periods, commas, equal signs, a blank space), enter the character as part of the syntax.

Keywords, variables, and operators may be displayed as required, optional, or default. Fragments, separators, and delimiters may be displayed as required or optional.

Item type

Definition

Required

Required items are displayed on the main path of the horizontal line.

Optional

Optional items are displayed below the main path of the horizontal line.

Default

Default items are displayed above the main path of the horizontal line.

Syntax examples

The following table provides syntax examples.

Table 12. Syntax examples	
Item	Syntax example
<p>Required choice.</p> <p>A required choice (two or more items) appears in a vertical stack on the main path of the horizontal line. You must choose one of the items in the stack.</p>	
<p>Required item.</p> <p>Required items appear on the main path of the horizontal line. You must specify these items.</p>	
<p>Optional item.</p> <p>Optional items appear below the main path of the horizontal line.</p>	

Table 12. Syntax examples (continued)	
Item	Syntax example
<p>Optional choice.</p> <p>An optional choice (two or more items) appears in a vertical stack below the main path of the horizontal line. You may choose one of the items in the stack.</p>	
<p>Default.</p> <p>Default items appear above the main path of the horizontal line. The remaining items (required or optional) appear on (required) or below (optional) the main path of the horizontal line. The following example displays a default with optional items.</p>	
<p>Variable.</p> <p>Variables appear in lowercase italics. They represent names or values.</p>	
<p>Repeatable item.</p> <p>An arrow returning to the left above the main path of the horizontal line indicates an item that can be repeated.</p> <p>A character within the arrow means you must separate repeated items with that character.</p> <p>An arrow returning to the left above a group of repeatable items indicates that one of the items can be selected, or a single item can be repeated.</p>	
<p>Fragment.</p> <p>The fragment symbol indicates that a labelled group is described below the main syntax diagram. Syntax is occasionally broken into fragments if the inclusion of the fragment would overly complicate the main syntax diagram.</p>	

ICSF operator commands

ICSF provides support for the following operator commands:

“Display ICSF” on page 119

Displays information about ICSF.

“SETICSF” on page 130

Used to perform specific administration functions.

Note: Installation options modified by the SETICSF command are in effect only until ICSF is stopped or restarted. When ICSF is restarted, the installation options will be re-initialized from the ICSF installation options data set. If you want to make the changes permanent, the installation options data set must be manually updated as needed.

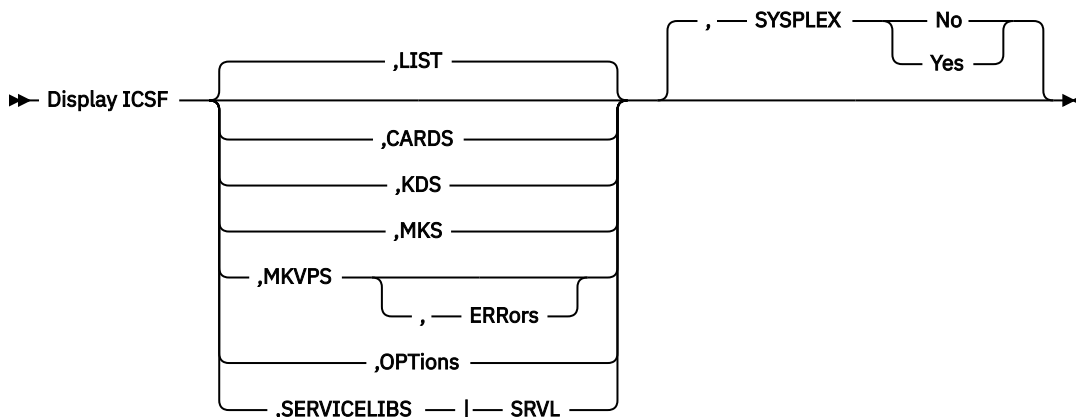
To see how to use RACF profiles to restrict the use of the ICSF operator commands, see ['Controlling the use of operator commands'](#) in *z/OS Security Server RACF Security Administrator's Guide*. The ICSF operator commands are a subset of the MVS system operator commands. Therefore, the subsystem name used in defining the RACF profiles is 'MVS' and the complete profiles names are 'MVS.DISPLAY.ICSF' and 'MVS.SETICSF.ICSF', as defined in ['MVS commands, RACF access authorities, and resource names'](#) in *z/OS MVS System Commands*.

Display ICSF

Use the Display ICSF command to:

- Display the status for available cryptographic devices.
- Display certain ICSF options.
- Display cryptographic usage tracking options.
- Display key lifecycle auditing options.
- Display key usage auditing options.
- Display information about the data set that is currently in use and what is set for a dynamic service update.
- Display information pertaining to active key data sets (KDS).
- Display the status of the master key registers for the available cryptographic devices.
- Display the master key verification pattern information from the KDS and cryptographic devices.
- List the systems that are available to participate in commands with a SYSPLEX scope.

Syntax



Parameters

CARDS

The system displays the following (message CSFM668I) information about the cryptographic devices available on the system or sysplex:

- The active domain.
- For each available device:
 - The device type (for example, CRYPTO EXPRESS5 COPROCESSOR).
 - The device index (for example, 5C36).

- The device status (for example, Active).
- The device serial number (for example, 99EA6059; not applicable for accelerators).
- The cryptographic coprocessor level and firmware version of the device. Not displayed for a cryptographic adapter configured as an accelerator.

For the cryptographic adapter configured as a CCA coprocessor, the format is LEVEL=x.y.zc. For example, LEVEL=7.4.13z, where:

- x is the CCA Cryptographic Adapter Level (CEX7P in this example)
- y is the cryptographic firmware level.
- z is the cryptographic firmware version.
- c is the IBM Z mainframe, where:
 - z is the certified generally available image.
 - a is the certified UDX image.
 - h indicates a debug or simulator image, such as used on zPDT.

For the cryptographic adapter configured as a secure key PKCS #11 coprocessor, the format is LEVEL=x.y.z/csp. For example, LEVEL=4.06.11/040D, where:

- x is the PKCS #11 cryptographic firmware level ('4' in this example indicates CEX7P), where:
 - 1 = CEX4P.
 - 2 = CEX5P.
 - 3 = CEX6P.
 - 4 = CEX7P.
 - 5 = CEX8P.
- y is the IBM Z mainframe hardware level, where:
 - 1 = z12 GA1 (D12K).
 - 2 = z12 GA2 (D15F).
 - 3 = z13 GA1 (D22H).
 - 4 = z13 GA2 (D36C).
 - 5 = z14 GA1 (D32L).
 - 6 = z14 GA2 (D35C).
 - 7 = z15 GA1 (D41C).
 - 8 = z16 GA1 (D51C).
 - 9 = z17 GA1 (D61C).
- z is the cryptographic firmware version level.
- csp is the cryptographic service provider level.

Note: If the PKCS #11 coprocessor is not attached to a system with the PTF for APAR OA61609 applied or the PTF for APAR OA61803 applied, LEVEL will be reported in the previous format of LEVEL=y.z CLIC=csp. If SYSPLEX=YES is used, the CSP level (csp) is not displayed.

- The total number of requests since ICSF initialization. This field supports up to 10 digits where the maximum value is $2^{32} - 1$. If the number of requests exceeds the maximum, ICSF wraps the count and displays a "+" in the high order digit to indicate wrapping (for example, +000000000).
- The number of requests both active and in the work queue for the device.
- The compliance mode of the CCA coprocessor, where applicable (for example, PCI-HSM 2016).
- The compliance mode of the EP11 coprocessor (for example, FIPS 2009).

For example:

```
D ICSF,CARDS
CSFM668I 16.36.34 ICSF CARDS 259
ACTIVE DOMAIN = 044
CRYPTO EXPRESS5 COPROCESSOR 5C00
STATUS=Active SERIAL#=DV4CK428 LEVEL=5.3.13z
REQUESTS=0122008567 ACTIVE=0000
CRYPTO EXPRESS5 ACCELERATOR 5A02
STATUS=Active
REQUESTS=0615576059 ACTIVE=0000
CRYPTO EXPRESS5 COPROCESSOR 5P03
STATUS=Active SERIAL#=DV4CB353 LEVEL=05.03 CLIC=040D
REQUESTS=0000000070 ACTIVE=0000
COMPMODE=FIPS: 09; BSI: NONE
CRYPTO EXPRESS6 COPROCESSOR 6C05
STATUS=Active SERIAL#=DV777392 LEVEL=6.0.5z
REQUESTS=0158807665 ACTIVE=0000
```

When you issue the D ICSF,CARDS command, the device type and device index displayed show the highest adapter type supported by the release of ICSF you are running on. For example, if the highest level supported is the CEX5S, the display shows the Crypto Express6 coprocessor as 5C05 (even though the firmware level could be 6.3.34).

```
D ICSF,CARDS
CSFM668I 16.42.34 ICSF CARDS 259
ACTIVE DOMAIN = 006
CRYPTO EXPRESS6 COPROCESSOR 5C10
STATUS=Active SERIAL#=DV73M319 LEVEL=6.3.34
REQUESTS=0158807003 ACTIVE=0000
```

Starting on ICSF FMID HCR77D1, with the PTF for APAR OA58358 applied, and later releases, if you add the SYSPLEX=YES parameter to the command, the display will show the same coprocessor with the highest device type and device index reported by any participating member in the sysplex. An example showing the D ICSF,CARDS,SYSPLEX=YES output on an HCR77D1 system:

```
CSFM668I 10.25.32 ICSF CARDS 660
CRYPTO EXPRESS6 COPROCESSOR 6C10 SERIAL#=DV73M319 LEVEL=6.3.34
S22 DOMAIN=006 Active REQ= N/A ACT=0000
S24 DOMAIN=004 Active REQ=0187926150 ACT=0000
S25 DOMAIN=008 Active REQ=0000000010 ACT=0000
CRYPTO EXPRESS6 COPROCESSOR 6C11 SERIAL#=DV746301 LEVEL=6.3.34
S22 DOMAIN=006 Active REQ= N/A ACT=0000
S24 DOMAIN=004 Active REQ=0132482073 ACT=0000
PCI-HSM=2016
S25 DOMAIN=008 Active REQ=0000000010 ACT=0000
PCI-HSM=2016
```

KDS

The system displays (message CSFM668I) information about the active key data sets (KDS) on the system or sysplex:

- The dataset name for each active KDS (CKDS, PKDS, and TKDS).
- The format of the KDS (for example, KDSR):
 - Possible values are KDSRL, KDSR, FIXED, and VARIABLE.
- The communication level in place for the KDS (for example, 3). This is only displayed in a sysplex environment.
- Whether the KDS is being shared in a sysplex group (for example, Y).
- The MKVPs initialized in the KDS (for example, DES AES).
 - The possible values are:
 - DES, AES, or both for CKDS.
 - RSA, ECC, or both for PKDS.
 - P11 for TKDS.

- The KDS MKVP date is the date that the MKVP value was first stored in the KDS or the date that the MKVP value was changed in the KDS as a result of reencipher. The date is in ISO 8601 format YYYY-MM-DD HH:MM:ss using UTC time. When new master keys are promoted by the same action that set the KDS MKVP date (for example, coordinated change master key), the time of day tag-length-value field of the SMF record type 82 subtype 49 for the new master key promotion is the same as the MKVP date. For more information, see [“Examples relating the MKVP date on D ICSF,MKVPS and D ICSF,KDS to the SMF subtype 49 record”](#) on page 128.
- 'Unknown' is displayed when the MKVP was stored in the KDS by a system with an ICSF FMID release prior to the lowest release that supports KDS MKVP date=.

For example:

```
-D ICSF,KDS,SYSplex=Y
CSFM668I 10.37.58 ICSF KDS 748
CKDS  ICSFTSTV.KDSRL1.CKDS
  FORMAT=KDSRL      COMM LVL=3  SYSplex=Y  MKVPs=DES AES
  DES MKVP date=2021-06-09 19:37:50
  AES MKVP date=2021-06-09 19:37:50
  Sharing systems:      S2F          S2E          S2D
CKDS  ICSFTSTV.VARREC1.KDSR.CKDS
  FORMAT=KDSR      COMM LVL=3  SYSplex=Y  MKVPs=DES AES
  DES MKVP date=Unknown
  AES MKVP date=Unknown
  Sharing systems:      S0E          S0F
PKDS  ICSFTSTV.KDSRL2.PKDS
  FORMAT=KDSRL      COMM LVL=3  SYSplex=Y  MKVPs=RSA ECC
  RSA MKVP date=2021-06-10 18:04:56
  ECC MKVP date=2021-06-10 18:04:56
  Sharing systems:      S2E          S2B          S2F
PKDS  ICSFTSTV.KDSR1.PKDS
  FORMAT=KDSR      COMM LVL=3  SYSplex=Y  MKVPs=RSA ECC
  RSA MKVP date=Unknown
  ECC MKVP date=Unknown
  Sharing systems:      S0F          S0E
TKDS  ICSFTSTV.KDSR1.GLGSM1.EP11.TKDS
  FORMAT=KDSRL      COMM LVL=3  SYSplex=Y  MKVPs=P11 RCS
  P11 MKVP date=Unknown
  Sharing systems:      S2F          S2E          S0F
```

MKS

The system displays (message CSFM668I) master key information:

- The name of the system (for example, SYSA).
- The active domain (for example, 003).
- For each device on the system:
 - The device index (for example, 5C38).
 - The device serial number (for example, 99EA6059).
 - The status of the device.
 - A status indicator for each possible master key.

For more information on the possible display values, see the Displaying Coprocessor or Accelerator Status topic in [z/OS Cryptographic Services ICSF Administrator's Guide](#).

For example:

```
SYSA  D ICSF,MKS

SYSA  CSFM668I 09.45.18 ICSF MKS 852
      SYSNAME: SYSA      DOMAIN: 003      CPC Name: PR2827A
      FEATURE SERIAL# STATUS      AES DES ECC RSA P11
      5C38  99EA6059 Active      A   A   A   A
      5P39  97006054 Active
```

MKVPS

The system displays the following (message CSFM668I) master key verification pattern information from the KDS and cryptographic devices:

- The dataset name for each active KDS. If there is no active KDS for a particular type of KDS (for example, CKDS), no data set name or device information is displayed for that KDS type.
- Up to six hexadecimal digits of the MKVP information from the header record of the KDS.
- The KDS MKVP date is the date that the MKVP value was first stored in the KDS or the date that the MKVP value was changed in the KDS as a result of reencipher. The date is in ISO 8601 format YYYY-MM-DD HH:MM:ss using UTC time. When new master keys are promoted by the same action that set the KDS MKVP date (for example, coordinated change master key), the time of day tag-length-value field of the SMF record type 82 subtype 49 for the new master key promotion is the same as the MKVP date. For more information, see [“Examples relating the MKVP date on D ICSF,MKVPS and D ICSF,KDS to the SMF subtype 49 record” on page 128.](#)
 - 'Unknown' is displayed when the MKVP was stored in the KDS by a system with an ICSF FMID release prior to the lowest release that supports KDS MKVP date=.
- The system name, coprocessor ID, and up to six hexadecimal digits of the current MKVP for each cryptographic device associated with the KDS.
 - A 'KDS/adaptor mismatch' indicator (*) is displayed if the MKVP of the KDS does not match the MKVP of the cryptographic device or the MKVP of the cryptographic device was 'Empty'.
 - 'Not Set' is displayed when the KDS is not initialized with the MKVP.
 - 'Ignored' is displayed for an MKVP in a cryptographic device if the MKVP in the KDS was not initialized. The MKVP in the cryptographic device is not checked. This is not considered an error when processing the ERRORS option. If the D ICSF,MKVPS,ERR command does not list any errors, issue the D ICSF,MKVPS command to confirm that the KDS MKVPS are set.
 - 'Empty' is displayed when the MKVP in the cryptographic device is empty.
 - 'N/A' is displayed for the ECC MKVP value in the cryptographic device when the cryptographic device is a CEX3C and the ECC value is not set in the cryptographic device.
- The number of hexadecimal digits of the MKVP information displayed is truncated to the value specified on the ICSF options parameter MASTERKCVLEN when that parameter value is less than six. The MASTERKCVLEN value used is the value set on the system issuing the command.

The Display ICSF,MKVPS command collects and displays information from systems at ICSF FMID HCR77B1 and later.

Although unlikely, the output from the D ICSF,MKVPS command could show a KDS and coprocessor MKVP value that is the same, but flagged as a mismatch. If this happens:

- Set MASTERKCVLEN to ALL to make sure the command is displaying the maximum of six hexadecimal digits of the MKVP value.
- If the MKVPs of the coprocessor and KDS still appear to match, use the ICSF Coprocessor Hardware Status panel (CSFCMP40) to see all the hexadecimal digits of MKVP in the coprocessor. Next, create a flat file of the KDS using IDCAMS to see the complete MKVP in the KDS header record. Compare the two values. To see the format of the KDS header records, see [Appendix A, “Diagnosis reference information,” on page 243.](#)

ERRORs

The display is limited to cryptographic devices whose current MKVP is set or empty and does not match the set MKVP in the KDS. If no KDS MKVPS are set, no errors are flagged. See the explanation of 'ignored' above. Use the D ICSF,MKVPS command to ensure that the KDS MKVPS are set.

For example:

```
D ICSF,MKVPS
CSFM668I 18.14.04 ICSF KDS
CKDS  ISFTTEST.CLC.CKDSVAR
  AES MKVP date=2020-04-20 17:25:27
  DES MKVP date=2020-05-13 22:09:56
      ID      AES      DES
KDSMKVPS  ....  2058C8  CA6B40
SY1       6C02  2058C8  CA6B40
PKDS  ISFTTEST.CLC.PKDSR
```

```

ECC MKVP date=2020-03-03 18:02:47
RSA MKVP date=Unknown
      ID      ECC      RSA
KDSMKVPS .... 78D81A  E83F15
SY1      6C02  78D81A  E83F15
No TKDS defined or no EP11 adapters online

```

Example showing that mismatches are found:

```

-D ICSF,MKVPS,SYSPLEX=Y
CSFM668I 10.38.26 ICSF MKVPS
CKDS  ICSFTSTV.KDSRL1.CKDS
AES MKVP Date=2021-06-09 19:37:50
DES MKVP Date=2021-06-09 19:37:50
      ID      AES      DES
KDSMKVPS .... BF494F  12B59A
S2B      6C00  BF494F  12B59A
S2B      7C04  BF494F  12B59A
S2B      7C08  *Empty  *Empty
S2C      5C00  BF494F  12B59A
S2C      5C04  BF494F  12B59A
CKDS  ICSFTSTV.VARREC1.KDSR.CKDS
AES MKVP Date=Unknown
DES MKVP Date=Unknown
      ID      AES      DES
KDSMKVPS .... 2058C8  CA6B40
S0E      5C05  2058C8  CA6B40
S0F      5C05  2058C8  CA6B40
PKDS  ICSFTSTV.KDSRL2.PKDS
ECC MKVP Date=2021-06-10 18:04:56
RSA MKVP Date=2021-06-10 18:04:56
      ID      ECC      RSA
KDSMKVPS .... 78D81A  E83F15
S2B      6C00  78D81A  E83F15
S2B      6C02  78D81A  E83F15
S2C      5C00  78D81A  E83F15
S2C      5C02  78D81A  E83F15
PKDS  ICSFTSTV.KDSR1.PKDS
ECC MKVP Date=Unknown
RSA MKVP Date=Unknown
      ID      ECC      RSA
KDSMKVPS .... 78D81A  E83F15
S0E      5C05  78D81A  E83F15
S0E      5C06  78D81A  E83F15
S0F      5C05  78D81A  E83F15
S0F      5C06  78D81A  E83F15
TKDS  ICSFTSTV.KDSR1.GLGSM1.EP11.TKDS
P11 MKVP Date=Unknown
      ID      P11
KDSMKVPS .... 5B083D
S0E      5P04  5B083D
S0F      5P04  5B083D
S2B      6P01  5B083D
S2B      7P07  5B083D
*KDS/adapter MKVP mismatch

```

Example showing that no errors are found:

```

-D ICSF,MKVPS
CSFM668I 10.38.26 ICSF MKVPS
CKDS  ICSFTSTV.KDSRL1.CKDS
AES MKVP Date=2021-06-09 19:37:50
DES MKVP Date=2021-06-09 19:37:50
      ID      AES      DES
KDSMKVPS .... BF494F  12B59A
S2B      6C00  BF494F  12B59A
PKDS  ICSFTSTV.KDSRL2.PKDS
ECC MKVP Date=2021-06-10 18:04:56
RSA MKVP Date=2021-06-10 18:04:56
      ID      ECC      RSA
KDSMKVPS .... 78D81A  E83F15
S2B      6C00  78D81A  E83F15
TKDS  ICSFTSTV.KDSR1.GLGSM1.EP11.TKDS
P11 MKVP Date=Unknown
      ID      P11
KDSMKVPS .... 5B083D
S2B      6P01  5B083D
S2B      7P07  5B083D

```


Example showing that the Errors keyword is specified and no errors are found:

```
SY1          d icsf,mkvp,err
SY1          CSFM668I 15.41.14 ICSF MKVPS
            No KDS/adapter MKVP mismatches found or KDS MKVPS not set
```

Example showing that either no KDS is defined or no cryptographic adapters are online:

```
SY1          d icsf,mkvp
            CSFM668I 08.49.49 ICSF MKVPS
            No KDS defined or no cryptographic adapters online
```

Example showing that when an MKVP is not set in the KDS, the cryptographic device MKVP value is 'Ignored'. If the MKVP value is set in the KDS, the cryptographic device MKVP is 'Empty':

```
D ICSF,MKVPS
CSFM668I 10.38.26 ICSF MKVPS
CKDS  ICSFTSTV.KDSRL1.CKDS
      AES MKVP Date=2021-06-09 19:37:50
      DES MKVP Date=2021-06-09 19:37:50
          ID      AES      DES
      KDSMKVPS    .... BF494F NotSet
      S2B         6C00 BF494F Ignored
PKDS  ICSFTSTV.KDSRL2.PKDS
      ECC MKVP Date=2021-06-10 18:04:56
      RSA MKVP Date=2021-06-10 18:04:56
          ID      ECC      RSA
      KDSMKVPS    .... 78D81A E83F15
      S2B         6C00 78D81A *Empty
      *KDS/adapter MKVP mismatch
      No TKDS defined or no EP11 adapters online
```

Example showing how the use of the Errors keyword alters the output from the Display ICSF,MKVPS command so that only the line flagged with '*' is displayed:

```
D ICSF,MKVPS,ERRORS
CSFM668I 10.38.26 ICSF MKVPS
PKDS  ICSFTSTV.KDSRL2.PKDS
      ECC MKVP Date=2021-06-10 18:04:56
      RSA MKVP Date=2021-06-10 18:04:56
          ID      ECC      RSA
      KDSMKVPS    .... 78D81A E83F15
      S2B         6C00 78D81A *Empty
      *KDS/adapter MKVP mismatch
```

For information to help resolve KDS/adapter mismatch problems, see 'Managing CCA Master Keys' and 'Managing PKCS #11 master keys' in [z/OS Cryptographic Services ICSF Administrator's Guide](#).

LIST

The system displays (message CSFM668I) members of a sysplex who are eligible to participate in Display ICSF and SETICSF commands. LIST is the default option.

For example:

```
SY1          D ICSF,LIST
SY1          CSFM668I 08.08.57 ICSF LIST 742
            Systems supporting SETICSF and DISPLAY ICSF commands:
            SYSNAME  RELEASE  DOM   CHG_DATE
            SY1      HCR77D0  000   06/18/19
```

OPTions

The system displays (message CSFM668I information):

- The name of the system (for example, SYSA).
- The ICSF release that is active (for example, HCR77B1).
- The most recent build date of ICSF executable code (for example, 01/09/16 or the latest ICSF code change).

- How much time must elapse between key references before a *refdate* change is recorded in the KDS record (*refdate* update interval).
- How often KDS *refdate* updates are hardened to the KDS dataset (*refdate* update period).
- The number of master key verification pattern digits.
- The cryptographic usage statistics that are being tracked.
- The COMPLIANCEWARN and AUDIT information.
- The TRACKCLASSUSAGE information.

For example:

```

SYSA          D ICSF,OPTIONS
SYSA          CSFM668I 10.23.21 ICSF OPTIONS 833
  SYSNAME = SYSA          ICSF LEVEL = HCR77C1
  LATEST ICSF CODE CHANGE = 08/22/17
  Refdate update interval in Days/HH.MM.SS = 030/00.00.00
  Refdate update period   in Days/HH.MM.SS = 000/01.00.00
  MASTERKCVLEN = display 3 digits
  AUDITKEYLIFECKDS: Audit CCA symmetric key lifecycle events
    SYSNAME LABEL TOKEN
    SYSA     Yes  Yes
  AUDITKEYLIFECPKDS: Audit CCA asymmetric key lifecycle events
    SYSNAME LABEL TOKEN
    SYSA     Yes  Yes
  AUDITKEYLIFETKDS: Audit PKCS #11 key lifecycle events
    SYSNAME TOKOBJ SESSOBJ
    SYSA     Yes  Yes
  AUDITKEYUSGCKDS: Audit CCA symmetric key usage events
    SYSNAME LABEL TOKEN Interval Days/HH.MM.SS
    SYSA     Yes  Yes 000/01.00.00
  AUDITKEYUSGPKDS: Audit CCA asymmetric key usage events
    SYSNAME LABEL TOKEN Interval Days/HH.MM.SS
    SYSA     Yes  Yes 000/01.00.00
  AUDITPKCS11USG: Audit PKCS #11 usage events
    SYSNAME TOKOBJ SESSOBJ NOKEY Interval Days/HH.MM.SS
    SYSA     Yes  Yes  Yes 000/01.00.00
  STATS:
    SYSA          ENG, SRV, ALG
  COMPLIANCEWARN: Compliance warning events
    SYSA          PCI-HSM 2016 Yes
  TRACKCLASSUSAGE: Class of operations tracked
    SYSA          DATAENC

```

SERVICELIBS | SRVL

The SERVICELIBS keyword displays the following information (message CSFM668I) about the data sets being used for active ICSF and what would be used in the event of a dynamic service update or after a restart of ICSF.

SCSFMOD0

The information listed shows the data set locations for SCSFMOD0. The data set listed under CURRENT is what the active instance of ICSF is using. The data set listed under NEXT is what is specified for the option SERVSCFMOD0 in the options dataset. NEXT will always be LNKLST unless SERVICELIBS(YES) has been specified.

SIEALNKE

The information listed shows the data set locations for SIEALNKE. The data set listed under CURRENT is what the active instance of ICSF is using. The data set listed under NEXT is what is specified for the option SERVSIEALNKE in the options dataset. NEXT will always be LNKLST unless SERVICELIBS(YES) has been specified.

CURRENT

Refers to the current code running for the instance of ICSF. It is either LNKLST or a data set that was loaded via a service option.

NEXT

Refers to the data set that would be used after the next SETICSF PAUSE command is run or what would be used after a manual start and restart of ICSF. If this information differs from what is in the options data set, either the options data set should be updated to match it, or a SETICSF

OPT,REFRESH command should be run to pick up the new service option values. NEXT will always be LNKST unless SERVICELIBS(YES) has been specified.

```
D ICSF,SERVICELIBS,SYSPLEX=Y
HCR77D0          SCSFMODE0 CURRENT          VOLSER
SYS1             LNKST
SYS2             LNKST
SYS3             SERV1.SCSFMODE0            CSFV01
HCR77D0          SCSFMODE0 NEXT
SYS1             SYS1.SRV1                  SRVDR1
SYS2             SYS1.SRV1                  SRVDR1
SYS3             SERV1.SCSFMODE0            SRVDR1
HCR77D0          SIEALNKE CURRENT            VOLSER
SYS1             LNKST
SYS2             LNKST
SYS3             SERV1.SIEALNKE              CSFV01
HCR77D0          SIEALNKE NEXT
SYS1             SYS1.SRV1                  SRVDR1
SYS2             SYS1.SRV1                  SRVDR1
SYS3             SERV1.SIEALNKE              CSFV01
```

SYSPLEX(YES or NO)

The SYSPLEX keyword increases the scope of the Display ICSF command to all participating members of the sysplex. The Display ICSF output is grouped according to CPC Name and shows the results of the Display ICSF command as it was executed on each member. Specify SYSPLEX=Yes to execute the command on all systems. Otherwise, specify SYSPLEX=No to execute the command only on the local (initiating) system. SYSPLEX=No is the default.

For example:

```
D ICSF,CARDS,SYSPLEX=Y
CSFM668I 11.49.49 ICSF CARDS 919
CPC Name = R01          CPC Sequence# = 0000000000042E08
CRYPTO EXPRESS6 COPROCESSOR 6C57 SERIAL#=99EA6003 LEVEL=6.0.00z
  SYSA      DOMAIN=000 Active REQUESTS=0000
            PCI-HSM=2016 MIGRATION
  SYSB      DOMAIN=002 Active
            REQ=4294967295 ACT=0008
  SYSC      DOMAIN=008 Active
            REQ=N/A      ACT=0001
CRYPTO EXPRESS5 COPROCESSOR 5P58 SERIAL#=97006035 LEVEL=02.09
  SYSA      DOMAIN=000 Active
            REQ=00000000100 ACT=0005
  SYSB      DOMAIN=002 Active
            REQ=0000000010 ACT=0003
  SYSC      DOMAIN=008 Active
            REQ=N/A      ACT=0007
CPC Name = R02          CPC Sequence# = 0000000000042E09
CRYPTO EXPRESS5 COPROCESSOR 5P59 SERIAL#=97006102 LEVEL=02.09
  SYSA      DOMAIN=000 Active
            REQ=0000000030 ACT=0006
CRYPTO EXPRESS5 ACCELERATOR 5P60
  SYSC      DOMAIN=008 Active
            REQ=+000085315 ACT=0004
```

```
SYSA D ICSF,OPT,SYSPLEX=Y
SYSA CSFM668I 11.36.35 ICSF OPTIONS 995
SYSNAME = SYSA          ICSF LEVEL = HCR77B1
LATEST ICSF CODE CHANGE = 01/09/15
Refdte update interval in Days/HH.MM.SS = 030/00.00.00
Refdte update period in Days/HH.MM.SS = 000/01.00.00
MASTERKCVLEN = display 3 digits
SYSNAME = SYSB          ICSF LEVEL = HCR77B1
LATEST ICSF CODE CHANGE = 01/09/15
Refdte update interval in Days/HH.MM.SS = 005/00.00.00
Refdte update period in Days/HH.MM.SS = 000/01.00.00
MASTERKCVLEN = display 3 digits
```

Usage Notes

For information on how to limit the use of MVS console commands to a specific set of users, see the System Operations topic in [z/OS MVS System Commands](#).

Examples relating the MKVP date on D ICSF,MKVPS and D ICSF,KDS to the SMF subtype 49 record

The following examples show that the MKVP date value displayed on the D ICSF,KDS and D ICSF,MKVPS commands matches the TOD in the SMF subtype 49 record when a master key promotion occurs with the setting of the MKVP value in the KDS.

Example 1

In this example, a coordinated change master has resulted in the RSA MKVP being changed in the PKDS and the RSA new master key being promoted to current in a single step. D ICSF,MKVPS and D ICSF,KDS MKVP dates match the TOD in the SMF record for the master key promotion event.

The D ICSF,MKVPS command shows:

```
PKDS  ISFTEST.CLC.PKDSR.NEW
ECC MKVP date=2020-08-04 18:18:16
RSA MKVP date=2020-08-05 20:40:41
      ID      ECC      RSA
KDSMKVPS .... 78D81A EF4C65
SY1      5C03 78D81A EF4C65
SY1      5C09 78D81A EF4C65
```

The corresponding SMF record for the promotion of the RSA new master key shows:

```
Subtype=0031 Master Key Event
Written when a Master Key is set or changed
5 Aug 2020 16:40:42.21
TME... 005B9DFD DTE... 0120218F SID... SP21 SSI... 00000000 STY... 0031
SYSNME SY1
MKFLGS C0
      80 This system initiated a coordinated change master key
      40 A change master key occurred on this system
KDSN.. ISFTEST.CLC.PKDSR.NEW
KDSTYP PKDS
MKVP.. 0320EF4C65754B5088C22D03480BC7B952B2
      03----- RSA
      20----- hex number of digits valid
      EF4C65754B5088C22D03480BC7B952B2 MKVP
SNs
      03-RSA
      99EA6074
      99EA6073
DOM... 0
TOD... 2020-08-05 20:40:41
ICSF Server Identity...
ICSF User Identity...
```

Example 2

In this example, the PKDS was initialized using the ICSF panels causing the RSA new master key to be promoted to current and the KDS to be updated with the RSA MKVP.

The D ICSF,MKVPS command shows:

```
PKDS  ISFTEST.CLC.PKDSR
RSA MKVP date=2020-07-29 21:14:31
      ID      ECC      RSA
KDSMKVPS .... NotSet EF4C65
SY1      5C00 Ignored EF4C65
```

The corresponding SMF record for the event shows:

```
Subtype=0031 Master Key Event
Written when a Master Key is set or changed
29 Jul 2020 17:14:31.69
TME... 005EB6C1 DTE... 0120211F SID... SP21 SSI... 00000000 STY... 0031
SYSNME SY1
MKFLGS 20
      20 New MK promoted to current MK outside of a change master key
KDSN.. ISFTEST.CLC.PKDSR
```

```

KDSTYP PKDS
MKVP.. 0320EF4C65754B5088C22D03480BC7B952B2
      03----- RSA
      20----- hex number of digits valid
      EF4C65754B5088C22D03480BC7B952B2 MKVP
Sns
      03-RSA
      99EA6073
DOM... 0
TOD... 2020-07-29 21:14:31
ICSF Server Identity...
End User Identity...

```

Example 3

In this example, the SET MK panel is used to promote AES, DES, RSA, and ECC new master keys. The KDS are not updated by this action. The TOD for the master key promotion event SMF record is calculated at the time of the promotion of the master keys.

D ICSF,MKVPS and D ICSF,KDS MKVP date does not match the TOD in the SMF record.

The D ICSF,MKVPS command shows:

```

CKDS  ISFTEST.CLC.CKDSVAR
AES MKVP date=2020-08-05 15:26:28
DES MKVP date=2020-08-05 14:19:03
      ID      AES      DES
KDSMKVPS .... 2058C8  CA6B40
SY1      5C03  2058C8  CA6B40
SY1      5C09  2058C8  CA6B40
PKDS  ISFTEST.CLC.PKDSR
ECC MKVP date=2020-08-04 18:18:16
RSA MKVP date=Unknown
      ID      ECC      RSA
KDSMKVPS .... 78D81A  E83F15
SY1      5C03  78D81A  E83F15
SY1      5C09  78D81A  E83F15

```

The corresponding SMF record for the event shows:

```

Subtype=0031 Master Key Event
Written when a Master Key is set or changed
5 Aug 2020 14:16:37.41
TME... 004E6D1D DTE... 0120218F SID... SP21    SSI... 00000000 STY... 0031
SYSNME SY1
MKFLGS 20
      20 New MK promoted to current MK outside of a change master key
MKVP.. 01102058C870E9D3194F
      01----- AES
      10----- hex number of digits valid
      2058C870E9D3194F MKVP
Sns
      01-AES
      99EA6074
MKVP.. 0210CA6B408A02371B1D
      02----- DES
      10----- hex number of digits valid
      CA6B408A02371B1D MKVP
Sns
      02-DES
      99EA6074
MKVP.. 0320E83F158521FEEA23986CC9483DAFD711
      03----- RSA
      20----- hex number of digits valid
      E83F158521FEEA23986CC9483DAFD711 MKVP
Sns
      03-RSA
      99EA6073
      99EA6074
MKVP.. 041078D81AC6C9610A2C
      04----- ECC
      10----- hex number of digits valid
      78D81AC6C9610A2C MKVP
Sns
      04-ECC
      99EA6074
DOM... 0

```

SETICSF

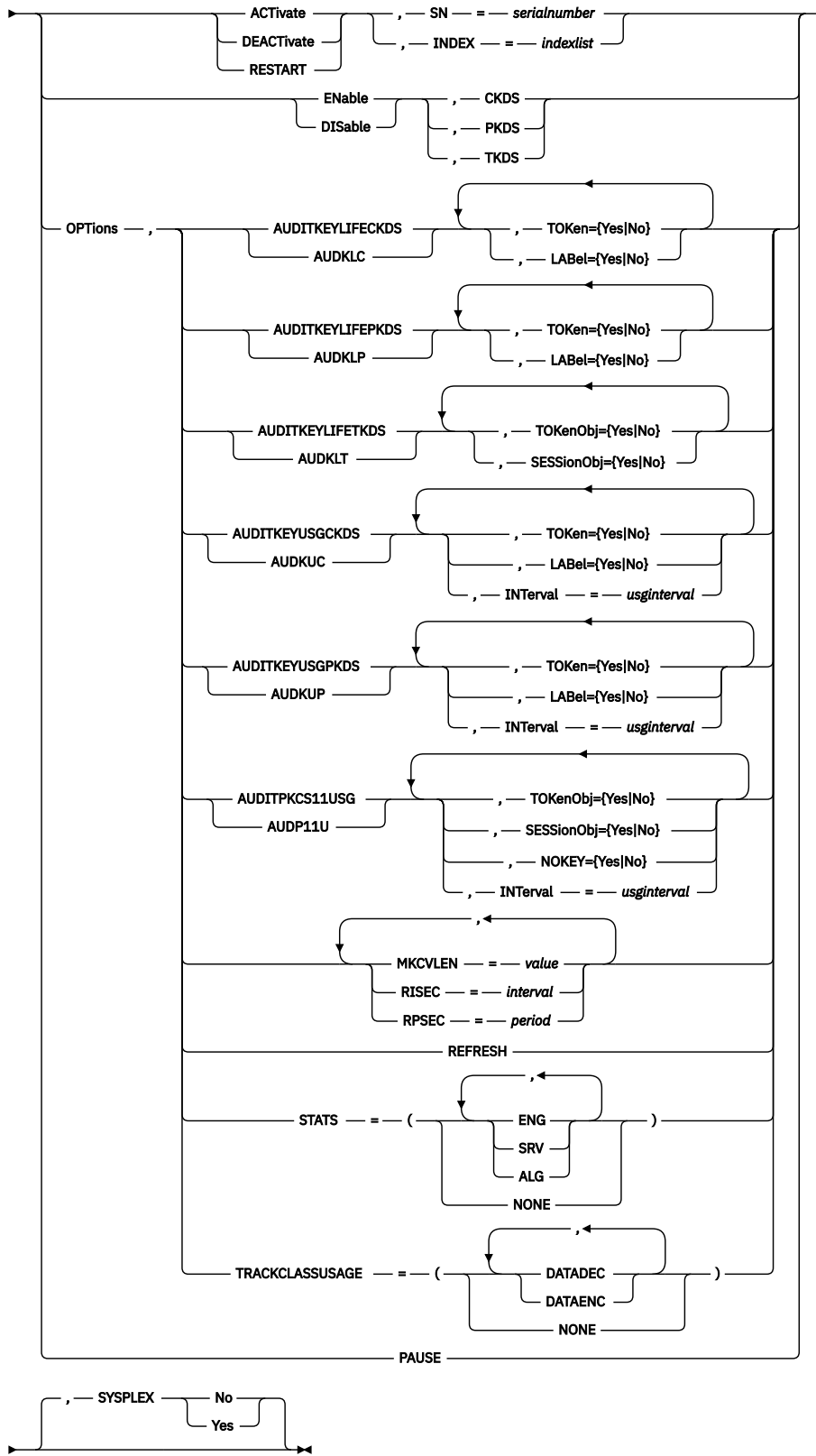
The SETICSF command is used to perform the following specific administration functions:

- Activate, deactivate, or restart a cryptographic device.
- Attempt to reopen sockets that were not previously opened.
- Change a subset of ICSF's installation options.
- Enable or disable updates to a key data set (KDS).
- Change cryptographic usage tracking options.
- Change key lifecycle auditing options.
- Change key usage auditing options.
- Pause transactions until ICSF is restarted.
- Refresh some options in the installation options data set.
- Track classes of cryptographic operations.

Note: For additional information on these administrative functions and their impact on ICSF and cryptographic devices, see [*z/OS Cryptographic Services ICSF Administrator's Guide*](#).

Syntax

➔ SETICSF ➔



Parameters

ACTivate

Activates the specified cryptographic device or devices. The valid device specifications are:

SN=serialnumber

Specify the serial number or numbers of the device or devices to be activated. The *serialnumber* value can be a single serial number or a list of serial numbers separated by commas. When more than one value is provided, the set of values must be enclosed in parentheses. For example:

```
SN=99AE6012
SN=(99AE6012,99AE6013,99AE6014)
```

INDEX=indexlist

Specify the index or indexes of the device or devices to be activated. The valid range is 0 to 63. The *indexlist* value can be a single device index, a range of indexes separated by a colon, or a combination of the two separated by commas. When more than one value is provided, the set of values must be enclosed in parentheses. For example:

```
INDEX=01
INDEX=(02:08)
INDEX=(02,04:07,09)
```

Note: To understand how the use of the INDEX value with the SYSPLEX parameter can result in devices with different serial numbers being modified on other systems sharing the KDS, see the explanation of the SYSPLEX parameter.

DEACTivate

Deactivates specified cryptographic devices. The valid device specification are:

SN=serialnumber

Specify the serial number or numbers of the device or devices to be deactivated. The *serialnumber* value can be a single serial number or a list of serial numbers separated by commas. When more than one value is provided, the set of values must be enclosed in parentheses. For example:

```
SN=99AE6012
SN=(99AE6012,99AE6013,99AE6014)
```

INDEX=indexlist

Specify the index or indexes of the device or devices to be deactivated. The valid range is 0 to 63. The *indexlist* value can be a single device index, a range of indexes separated by a colon, or a combination of the two separated by commas. When more than one value is provided, the set of values must be enclosed in parentheses. For example:

```
INDEX=01
INDEX=(02:08)
INDEX=(02,04:07,09)
```

Note: To understand how the use of the INDEX value with the SYSPLEX parameter can result in devices with different serial numbers being modified on other systems sharing the KDS, see the explanation of the SYSPLEX parameter.

DISable

Disables updates for the specified key data set. The valid KDS specifications are:

- CKDS
- PKDS
- TKDS

ENable

Enables updates for the specified key data set. The valid KDS specifications are:

- CKDS
- PKDS

- TKDS

OPTions

Changes the value of an ICSF option. The supported options are:

AUDITKEYLIFECKDS,AUDKLC

Changes one or more options related to lifecycle auditing of CKDS labels and tokens.

LABEL,LAB = YES|NO

YES

Enables key lifecycle auditing of CKDS labels.

NO

Disables key lifecycle auditing of CKDS labels.

TOKEN,TOK = YES|NO

YES

Enables key lifecycle auditing of CKDS tokens.

NO

Disables key lifecycle auditing of CKDS tokens.

Example:

```
AUDITKEYLIFECKDS, LABEL=YES, TOKEN=NO
```

AUDITKEYLIFEPKDS,AUDKLP

Changes one or more options related to lifecycle auditing of PKDS labels and tokens.

LABEL,LAB = YES|NO

YES

Enables key lifecycle auditing of PKDS labels.

NO

Disables key lifecycle auditing of PKDS labels.

TOKEN,TOK = YES|NO

YES

Enables key lifecycle auditing of PKDS tokens.

NO

Disables key lifecycle auditing of PKDS tokens.

Example:

```
AUDKLP, TOK=NO, LABEL=YES
```

AUDITKEYLIFETKDS,AUDKLT

Changes one or more options related to lifecycle auditing of TKDS token objects and session objects.

TOKENOBJ,TOKO = YES|NO

YES

Enables key lifecycle auditing of TKDS token objects.

NO

Disables key lifecycle auditing of TKDS token objects.

SESSIONOBJ,SESSO = YES|NO

YES

Enables key lifecycle auditing of TKDS token objects.

NO

Disables key lifecycle auditing of TKDS token objects.

Example:

```
AUDKLT,TOKO=YES  
AUDKLT,TOKO=YES,SESSO=YES
```

AUDITKEYUSGCKDS,AUDKUC

Changes one or more options related to key usage auditing of CKDS labels and tokens.

LABEL,LAB = YES|NO

YES

Enables key usage auditing of CKDS labels.

NO

Disables key usage auditing of CKDS labels.

TOKEN,TOK = YES|NO

YES

Enables key usage auditing of CKDS tokens.

NO

Disables key usage auditing of CKDS tokens.

INTERVAL,INT = *usginterval*[H|M|S]

The interval over which key usage records are aggregated before being written out to SMF. The time unit may be specified as H – hours, M – minutes, or S – seconds. If the time unit is not specified, the default is S - seconds. The minimum value of *usginterval* is 1 second. The maximum value is 24 hours.

Example:

```
AUDKUC,LABEL=YES,TOK=YES  
AUDKUC,INT=8H
```

AUDITKEYUSGPKDS,AUDKUP

Changes one or more options related to key usage auditing of PKDS labels and tokens.

LABEL,LAB = YES|NO

YES

Enables key usage auditing of PKDS labels.

NO

Disables key usage auditing of PKDS labels.

TOKEN,TOK = YES|NO

YES

Enables key usage auditing of PKDS tokens.

NO

Disables key usage auditing of PKDS tokens.

INTERVAL,INT = *usginterval*[H|M|S]

The interval over which key usage records are aggregated before being written out to SMF. The time unit may be specified as H – hours, M – minutes, or S – seconds. If the time unit is not specified, the default is S - seconds. The minimum value of *usginterval* is 1 second. The maximum value is 24 hours.

Example:

```
AUDITKEYUSGPKDS,LAB=YES,TOKEN=NO  
AUDKUP,LAB=YES,TOKEN=NO,INT=3600
```

AUDITPKCS11USG,AUDP11U

Changes one or more options related to usage auditing of PKCS #11 services.

TOKENOBJ,TOKO = YES|NO**YES**

Enables key usage auditing of PKCS #11 token objects.

NO

Disables key usage auditing of PKCS #11 token objects.

SESSIONOBJ,SESSO = YES|NO**YES**

Enables key usage auditing of PKCS #11 session objects.

NO

Disables key usage auditing of PKCS #11 session objects.

NOKEY = YES|NO**YES**

Enables usage auditing of PKCS #11 services which do not involve an object.

NO

Disables usage auditing of PKCS #11 services which do not involve an object.

INTERVAL,INT = *usginterval*[H|M|S]

The interval over which key usage records are aggregated before being written out to SMF. The time unit may be specified as H – hours, M – minutes, or S – seconds. If the time unit is not specified, the default is S - seconds. The minimum value of *usginterval* is 1 second. The maximum value is 24 hours.

Example:

```
AUDP11U,TOKO=YES,SESSIONOBJ=NO
AUDP11U,TOKO=YES,SESSIONOBJ=NO,NOKEY=YES,INTERVAL=1440M
```

MKCVLEN = *value*

Specifies the number of hexadecimal digits to display on the ICSF Coprocessor Hardware Status panel (CSFCMP40) for the verification and hash patterns for the master keys. The patterns are also referred to as key check values. The value may be 2, 3, 4, 5, 6, or ALL. When an integer value is specified, that number of digits will displayed. When ALL is specified, all digits will be displayed.

This option can be used to be in compliance with the ISO11568 standard for display of the key check values for master keys.

Notes:

- This option corresponds to the MASTERKCVLEN option in the ICSF installation options data set. Be aware that when ICSF is restarted, the value will revert to the value specified by the MASTERKCVLEN option in the ICSF installation options data set.
- This option has no effect on the output of the DISPLAY ICSF,MKS command.

PAUSE

ICSF terminates after all in-flight transactions finish, and any new transactions are paused until ICSF is restarted. ICSF must then be restarted via ARM policy, customer automation, or manually. Upon restart, ICSF is loaded from the specified service data set, and paused transactions resume.

Note: Before issuing the SETICSF PAUSE command, see [“Dynamic service update” on page 140](#).

REFRESH

Refreshes supported option parameters whose values have been updated in the current installation options data set listed in the ICSF startup procedure on the CSFPARM DD statement or from the CSFPRMxx member in the parmlib concatenation.

Refreshable option parameters are AUDITKEYLIFECKDS, AUDITKEYLIFEPKDS, AUDITKEYLIFETKDS, AUDITKEYUSGCKDS, AUDITKEYUSGPKDS, AUDITPKCS11USG, BEGIN, CHECKAUTH, CICSAUDIT, COMPLIANCEWARN, DEFAULTWRAP, END, FIPSMODE, KEYARCHMSG, KDSREFDAYS, MASTERKCVLEN, MAXSESSOBJECTS, MAXSLOWOPS, RNGCACHE, SSM, TRACKCLASSUSAGE, USERPARM, and WAITLIST.

Note: SETICSF REFRESH cannot be used to dynamically change to or from FIPS 140-3 ENFORCE or INDICATE modes. A recycle of ICSF is required to change the FIPMSODE option to or from FIPSMODE(140-3,ENFORCE) and FIPSMODE(140-3,INDICATE).

RISEC = *interval*

Specifies, in seconds, how often a record should be written for a reference date/time change. The values must be between 0 (write a record for every reference) and 2592000 (30 days) seconds. For example:

```
RISEC=300
```

Note: OPTIONS,RISEC corresponds to the KDSREFDAYS option in the ICSF options data set, which can only be specified in full days. When the RISEC option has been used to change the *refdate* interval, the value for KDSREFDAYS on the Installation Options Display panel is set to SETICSF to indicate that the current value has been modified from the value that is set in the installation options dataset.

RPSEC = *period*

Specifies how often in seconds ICSF hardens *refdate* updates to the appropriate key data set. The value must be between 10 and 3600. The default is 3600 seconds. For example:

```
RPSEC=30
```

Note: There is no corresponding keyword in the ICSF options data set for the RPSEC option. The value can only be changed using the SETICSF command.

STATS

Updates cryptographic usage tracking options. Keywords can be combined to track multiple statistics.

Each issuance of the command replaces the prior settings. For example, if ENG is tracked and SRV is to be added, then STATS=(ENG,SRV) must be issued.

ENG

Enables usage tracking of cryptographic engines. Supports Crypto Express adapters, CPACF, and software.

SRV

Enables usage tracking of cryptographic services. Supports ICSF callable services and UDXes only.

ALG

Enables usage tracking of cryptographic algorithms. Supports cryptographic algorithms that are referenced in cryptographic operations. Limited support for key generation, key derivation, and key import.

NONE

Disables usage tracking of cryptographic statistics.

TRACKCLASSUSAGE

Updates the options for tracking of classes of cryptographic operations. Keywords can be combined to track multiple classes.

Each issuance of the command replaces the prior settings.

DATAENC

Symmetric keys data encryption operations.

DATADec

Symmetric keys data decryption operations.

NONE

Disables tracking of classes of cryptographic operations.

Installation options that are modified by the SETICSF command are in effect only until ICSF is stopped or restarted. When ICSF is restarted, the installation options are re-initialized from the ICSF

installation options data set. If you want to make the changes permanent, the installation options data set must be manually updated as needed.

RESTART

Restarts specified cryptographic devices. For the specified devices, the work queues are cleared and ICSF runs through normal configuration processing in an attempt to return a device that is in an error state to an active state. This is most appropriate for a device that has had an error such as CARD BUSY.

SYSPLEX(YES or NO)

The SYSPLEX keyword increases the scope of the SETICSF command to all participating members of the sysplex. The SETICSF command is executed locally on the initiating system and then again on each participating member of the sysplex. The output indicates which systems were able to process the request as well as those systems that were not able to process the request due to a lack of support or an error.

Specify SYSPLEX=Yes to execute the command on all systems. When SYSPLEX=YES is specified, the command may affect cryptographic devices on all systems within the sysplex as follows:

- When SN is specified, all cryptographic devices that have the specified serial number or numbers are affected. No other filtering criteria is applied.
- When INDEX is specified instead of SN, additional filtering criteria is applied. Cryptographic devices that do not meet this criteria are skipped:
 - The command will only affect those systems within the sysplex that share the same KDS via the SYSPLEXnKDS(YES,...) ICSF installation option. This includes the originating system.
 - For each such system, both the index or indexes and serial number or numbers must match that of the system where the command was issued. The use of SYSPLEX with INDEX results in the command action being performed on all devices at that index on the originating system as well as the cryptographic device at that index on any system that is sharing the KDS.

For example, the command SETICSF DEACT,INDEX=1,SYSPLEX=YES would deactivate the cryptographic device at index 01 on the originating system as well as the cryptographic device at index 01 on any system sharing the KDS. In this case, it is better to use SN rather than INDEX as the SETICSF DEACT command can affect devices that have different serial numbers when INDEX is used with SYSPLEX=YES.

Specify SYSPLEX=No to execute the command only on the local (initiating) system. When SYSPLEX=NO is specified or defaulted, the command affects only affects devices on the system where the command was issued.

SYSPLEX=No is the default.

Usage Notes

Installation options modified by the SETICSF command are in effect only until ICSF is stopped or restarted. When ICSF is restarted, the installation options will be re-initialized from the ICSF installation options data set. If you want to make the changes permanent, the installation options data set must be manually updated as needed.

For information on how to limit the use of MVS console commands to a specific set of users, see the System Operations topic in *z/OS MVS System Commands*.

Using different configurations

A central processor complex can have multiple cryptographic features of various types. This topics describes some of the different configurations available.

You can divide your processor complex into PR/SM logical partitions. When you create logical partitions on your processor complex, you use the usage domain index on the Support Element Customize Image Profile page only if you have, or plan to add a cryptographic feature.

The DOMAIN parameter is optional. The number that is specified for the usage domain index must correspond to the domain number you specified with the DOMAIN(n) keyword in the installation options data set – if you specified one. The DOMAIN keyword is required if more than one domain is specified as the usage domain on the PR/SM panels.

A cryptographic feature can be configured and shared across multiple partitions.

Note: The domain assigned to the TKE Host LPAR must be unique if TKE is to control all the coprocessor cards in the environment. No other LPAR can use the domain assigned to the TKE Host.

The maximum number of LPARs depends on your server. The maximum number of usage domains matches the maximum number of LPARs available on the server. A usage domain can be configured to be unique to one LPAR or assigned to different LPARs accessing different cryptographic features. This is illustrated by LPAR 1 and LPAR 3 in [Figure 1 on page 138](#). They are both assigned to usage domain 0, but on two different CEXnAs.

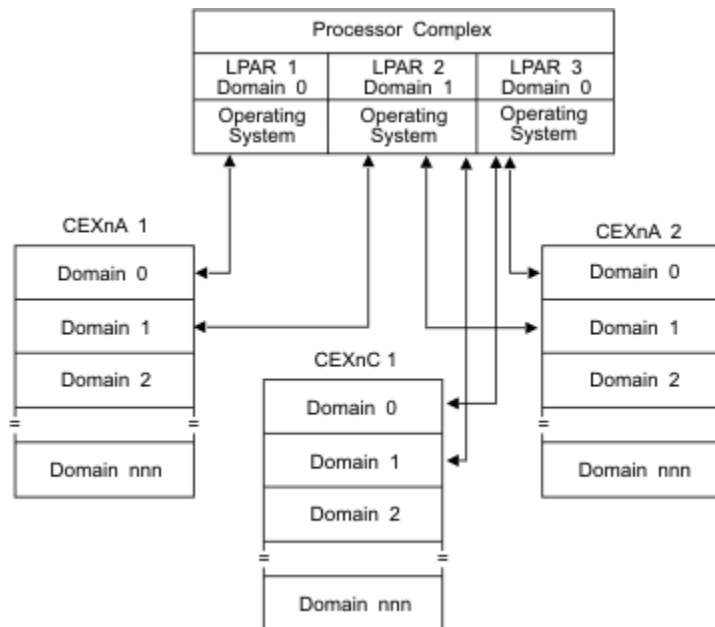


Figure 1. Multiple Crypto coprocessors on a complex

Adding and removing cryptographic coprocessors

It may become necessary for your installation to add or remove cryptographic features. This topic gives you a brief overview of the hardware implications. For more detailed information, refer to the *IBM Z PR/SM Planning Guide* and the *Hardware Management Console Operations Guide*.

There are several terms associated with removing the features. Use the Support Element (SE) panel to configure cryptographic features online and offline (standby). Use the ICSF Coprocessor Management panel from your TSO user ID to activate and deactivate cryptographic features. Use the TKE workstation to enable and disable cryptographic coprocessors.

Adding cryptographic coprocessors

You can dynamically add cryptographic features. You must have feature 3863 installed on your system.

The cryptographic feature number must be in the Candidates list of the LPAR Activation panel. **Configure On** the card. Each feature will display. For coprocessors, once the master keys are entered, they become active. The accelerator will automatically become active.

Note: ALL crypto coprocessors cards must be loaded with the same level of code. Otherwise, unpredictable results can occur. When updating licensed internal code (LIC) on the coprocessors:

- You can migrate to new LIC levels on the coprocessors one at a time without taking an outage, and

- you need to complete the LIC upgrade on all coprocessors before trying to exploit a new function introduced by the new LIC.

Steps for activating/deactivating cryptographic coprocessors

From your TSO userid, select option 1, Coprocessor Mgmt. On the Coprocessor Management panel, you can select the features you want to activate or deactivate.

```
CSFCMP00 ----- ICSF Coprocessor Management ----- Row 1 to 5 of 5
COMMAND ==>                                           SCROLL ==> PAGE

Select the cryptographic features to be processed and press ENTER.
Action characters are: A, D, E, K, R and S. See the help panel for details.

CRYPTO    SERIAL    STATUS    AES    DES    ECC    RSA    P11
FEATURE  NUMBER    -----
-----
.  4A00    N/A      Active
.  4P38    97006070 Active
a  4C39    93X06044 Deactivated
d  4C40    93X06077 Master key incorrect  C    C    U    C
.  4C41    93X06071 Active      A    A    A    A
***** Bottom of data *****
```

Figure 2. ICSF coprocessor management

When a coprocessor or accelerator is deactivated through the Coprocessor Management Panel, the card is only deactivated for that one LPAR.

Note: The SETICSF ACTivate and the SETICSF DEACTivate commands can also be used to activate or deactivate coprocessors and accelerators.

Steps to configure on/off cryptographic coprocessors

To configure the cryptographic features online and offline, you must use the support element (SE) panel.

Before configuring a feature offline, it is strongly recommended that you deactivate the feature first from the ICSF Coprocessor Management panel. You need to 'deactivate' the feature in ALL partitions that are using that feature. This allow jobs to complete before the feature is varied offline. You use the **Configure On/Off** service on the Support Element panel to take the feature offline (standby).

After you configure the feature offline from the SE panel, press ENTER on the Coprocessor Management panel to verify that the feature is offline. This configuring is done to remove and replace features or to load new code for the cryptographic features.

To bring a feature back online, use the SE panel again. If a feature was deactivated and then configured offline, you will need to activate it again through the Coprocessor Management panel.

There are no z/OS operator commands to configure the devices online or offline.

Steps for enabling/disabling cryptographic coprocessors

With TKE you can disable/enable coprocessors. When a coprocessor is deactivated through the Coprocessor Management Panel, the coprocessor is only deactivated for that one LPAR. When a coprocessor disabled by TKE, the card is disabled for the entire system, not just the LPAR that issued the disable.

Intrusion latch on the cryptographic coprocessors

Under normal operation, the intrusion latch on a coprocessor is tripped when the feature is removed. This causes all installation data, master keys, retained keys, roles and authorities to be zeroized in the feature when it is reinstalled.

If a situation arises where a coprocessor needs to be removed, for example, you need to remove your feature for service, and you do not want the installation data to be cleared, perform this procedure to disable the coprocessor before removing.

This process will require you to switch between the TKE application, the ICSF Coprocessor Management panel, and the Support Element.

1. Open an Emulator Session on the TKE workstation and logon to your TSO userid on the Host System where the coprocessor will be removed.
2. From the ICSF Primary Option Menu on TSO, select Option 1 for Coprocessor Management.
3. Leave the Coprocessor Management panel displayed during the rest of this procedure. You will be required to press ENTER on the Coprocessor Management panel at different times. DO NOT EXIT this panel.
4. Open the TKE Host where the coprocessor will be removed. Open the coprocessor. Click on Disable Crypto Module.
5. After the coprocessor has been disabled from TKE, press ENTER on the Coprocessor Management panel. The status should change to DISABLED.
Note: You do not need to deactivate a disabled card.
6. **Configure Off** the coprocessor from the Support Element.
7. After the card has been taken Offline, press ENTER on the Coprocessor Management panel. The status should change to OFFLINE.
8. Remove the coprocessor. Perform whatever operation needs to be done. Replace the coprocessor.
9. **Configure On** the coprocessor from the Support Element.
10. When the initialization process is complete, press ENTER on the Coprocessor Management panel. The status should change to DISABLED.
11. From the TKE Workstation Crypto Module General page, click on Enable Crypto Module.
12. After the coprocessor has been enabled from TKE, press ENTER on the Coprocessor Management panel. The Status should return to its original state. If the Status was ACTIVE in step 2, when the coprocessor is enabled it should return to ACTIVE.

All installation data, master keys, retained keys, roles, and authorities should still be available. The coprocessor data was not cleared with the card removal because it was Disabled first via the TKE workstation.

Dynamic service update

Dynamic service update allows you to apply service updates to ICSF with minimal impact to ICSF availability. It allows ICSF to activate service without a manual stop and start of ICSF. These updates include service updates as well as changes to the options data set that cannot be applied via the SETICSF OPT,REFRESH command. Additionally, dynamic service updates can be used to recycle ICSF when there are problems that are not resolving.

For any service to be applied to ICSF using the dynamic service update method, there must be a +HOLD(DYNACT) with the PTF. Otherwise, the service cannot be applied using dynamic service update. Dynamic service update cannot be used to migrate ICSF to a new ICSF FMID. Read [“Considerations when using dynamic service update” on page 141](#) and [“Steps to initiate dynamic service update” on page 141](#) carefully before starting a dynamic service update.

The following are some possible scenarios of when you might want to use dynamic service update:

- ICSF is currently running with no service keywords. Modules reside in HLQ.SCSFMODE0 and HLQ.SIEALNKE.
 - If a SETICSF PAUSE command is issued when ICSF is running with no service keywords, no service is loaded and ICSF comes up normally when it is restarted.

- When service is ready to be applied for ICSF, dynamic service update can be used to non-disruptively activate the service.
 - At this point, the updates can be applied. If they are satisfactory, the service can be applied to the production HLQ.SCSFMODE0 and HLQ.SIEALNKE data sets using the normal processes for updating LNKST data sets.
- [“Considerations when using dynamic service update” on page 141](#)
- [“Steps to initiate dynamic service update” on page 141](#)

Considerations when using dynamic service update

- When selecting a window of time to initiate a dynamic service update:
 - To reduce the overall disruption to the system applications that are using ICSF, consider a time when ICSF has the least amount of activity. This reduces the amount of time that a dynamic service update takes to complete.
 - Specific ICSF activity at the time that a SETICSF PAUSE command is issued can prolong the length of a dynamic service update. Avoid longer running services and management activities (for example, changing master keys, RSA key pair generation, and so on) because this could potentially cause timeouts for applications that get paused during the dynamic service update window.
- Performing a dynamic service update discards all PKCS #11 session objects. All applications and components that use session objects will need to recreate them. Therefore, all applications that rely on these session objects should be quiesced prior to the dynamic service update.
- If activating dynamic service update for more than one system in a sysplex, it is recommended that you issue the SETICSF PAUSE command on one system at a time. You can then verify that ICSF has successfully restarted before proceeding to apply service to the other systems.
- The SERVSCSFMODE0 and SERVIEALNKE options data set keywords define the service data sets for use with dynamic service update. The service data sets used to contain service SERVSCSFMODE0 and SERVIEALNKE are not the production LNKST data sets. These data sets must be APF authorized.
- Define and verify how ICSF is to be restarted after termination. The following methods can be used to restart ICSF:
 - Use z/OS Automatic Restart Manager (ARM) to restart ICSF. Note: Message IXC812I 'JOBNAME *jobname*, ELEMENT *elementname* FAILED. THE ELEMENT WAS RESTARTED *text*' is issued indicating ICSF was restarted and does not indicate a failure of dynamic service update. For more information about subscribing to ARM, see [“ARM policy” on page 114](#).
 - Automatically restart ICSF after receiving message CSFM401I 'CRYPTOGRAPHY - SERVICES ARE NO LONGER AVAILABLE.' Note that there may be a delay between when the message is issued and when the address space is fully terminated. This may require a delay in message automation to ensure the ICSF address space is fully terminated.
 - Manually restart ICSF after receiving message CSFM401I 'CRYPTOGRAPHY - SERVICES ARE NO LONGER AVAILABLE.'

Steps to initiate dynamic service update

Complete the following steps to initiate a dynamic service update. If running in a sysplex environment, these steps apply to each system that you want to apply dynamic update services.

1. Update the ICSF installation options data set and set the following keywords:

```
SERVSCSFMODE0(dsn,volser)
SERVIEALNKE(dsn,volser)
SERVICELIBS(YES)
```

SERVSCSFMODE0(*dsn*[,*volser*])

The SERVSCSFMODE0 keyword specifies the service data set containing the SCSFMODE0 service. If the SERVSCSFMODE0 option is not specified, the default is to use the code from LNKST.

SERVSIEALNKE(dsn[,volser])

The SERVSIEALNKE keyword specifies the service data set containing the SIEALNKE service. If the SERVSIEALNKE option is not specified, the default is to use the code from LNKLST.

SERVICELIBS(YES | NO)

The SERVICELIBS keyword is used to control whether service data set information is used. If SERVICELIBS(YES), ICSF uses service data sets. If SERVICELIBS(NO), ICSF uses LNKLST. If the SERVICELIBS option is not specified, the default is SERVICELIBS(NO).

- Refresh the ICSF installation options data set to update or validate the SERVSCSFMOD0 and SERVSIEALNKE keyword values by issuing the following console command:

```
SETICSF OPT,REFRESH
```

This step verifies that there are no syntax errors in the ICSF installation options data set and also validates that the service data sets are correctly defined and accessible. If the refresh is successful, SERVSCSFMOD0 and SERVSIEALNKE values are saved by ICSF for later use during a dynamic service update.

Note: Active ICSF is not affected by this change.

If running in a sysplex environment with multiple systems in need of the same service update, issue the SETICSF OPT,REFRESH command on all systems requiring service.

- After the service data set have been defined on all the systems, issue the DISPLAY ICSF, SERVICELIBS,SYSPLEX=Y command.

Examine the output to verify that all systems requiring the service have the correct service data sets defined.

```
D ICSF,SERVICELIBS,SYSPLEX=Y
HCR77D0  SCSFMOD0 CURRENT          VOLSER
SYS1      LNKLST
SYS2      LNKLST
SYS3      SERV1.SCSFMOD0            CSFV01
HCR77D0  SCSFMOD0 NEXT
SYS1      SYS1.SRV1                 SRVDR1
SYS2      SYS1.SRV1                 SRVDR1
SYS3      SERV1.SCSFMOD0            SRVDR1
HCR77D0  SIEALNKE CURRENT          VOLSER
SYS1      LNKLST
SYS2      LNKLST
SYS3      SERV1.SIEALNKE            CSFV01
HCR77D0  SIEALNKE NEXT
SYS1      SYS1.SRV1                 SRVDR1
SYS2      SYS1.SRV1                 SRVDR1
SYS3      SERV1.SIEALNKE            CSFV01
```

In this example, SYS3 is using identical service data sets. This might have occurred from either having the service options specified at initialization in the ICSF installation options data set or after completing a dynamic service update using the SETICSF PAUSE command. The other two systems (SYS1 and SYS2) are pointing to different service data sets and are currently running using the code from LNKLST. When they are restarted, they will be running with the code from SYS1.SRV1. If the service was only located in the SYS3 data set, the other systems would need to update their ICSF installation options data set and issue the SETICSF OPT,REFRESH command to point to the same data set.

- Activate dynamic update service by issuing the following console command:

```
SETICSF PAUSE
```

Note: It is best to issue the SETICSF PAUSE command on one system in a sysplex at a time.

ICSF allows current requests to finish and pauses incoming requests. For current requests, ICSF allows requests to continue processing as long as requests are completing. Once the number of requests that are active is the same for greater than 10 seconds, ICSF terminates and fails the in-progress requests.

5. Once there are no active requests in ICSF, ICSF begins termination. Message CSFM401I 'CRYPTOGRAPHY - SERVICES ARE NO LONGER AVAILABLE' is issued prior to ICSF address space termination.
6. ICSF must be restarted via ARM policy, customer automation, or manually. If automation is being used, no actions are needed. If manually restarting, ICSF must be started now. ICSF utilizes any active service data set at this time.
7. ICSF completes initialization and resumes all paused requests. Normal ICSF processing resumes.
8. Upon completion of the dynamic update service, CSFM694I is issued:

```
CSFM694I ICSF SERVICE UPDATE COMPLETED. CLEANUP=cleanup_time
RESTART=restart_time REINIT=reinit_time CODE DATE OLD=old_date
NEW=new_date
```

cleanup_time

How long it took in seconds for ICSF to finish updating the existing service requests, write any pending SMF records, clean up the address space, and to terminate.

restart_time

How long it took in seconds for the address space termination to complete and for automation to restart ICSF.

reinit_time

How long it took in seconds for ICSF to complete initialization after starting back up.

The sum of these three values is the amount of time the dynamic service update took. This is the maximum amount of time in which applications were paused. If this time interval is likely to cause application timeouts, consider the following ways to reduce the time.

cleanup_time

- Choose a time for the SETICSF PAUSE command when the workload is at the lowest point because the fewer active calls to ICSF is best.
- ICSF writes SMF records for key usage and statistics. Turn off these activities via the SETICSF operator command before issuing the SETICSF PAUSE command to eliminate the writing of SMF records.

restart_time

- Use either z/OS Automatic Restart Manager (ARM) or automation to restart ICSF. Manual restart of ICSF will likely introduce additional delay. To use ARM to restart ICSF, see [“ARM policy” on page 114](#).

reinit_time

- The largest contributor to the reinitialization time is most likely the time it takes to load the CKDS into storage. Eliminating old, unused keys from the CKDS helps reduce this time. Examine console messages like message CSFM653I '*kds* LOADED *num_record* RECORDS WITH AVERAGE SIZE *average_size*' to learn how long it takes for the CKDS to be loaded.
9. To verify that dynamic service update was activated, see [“Verifying dynamic service update” on page 143](#).
 10. To permanently add the dynamic service update, update your LNKST and remove the service options from the ICSF installation options data set.

Verifying dynamic service update

Steps to verify that dynamic service update was activated:

1. Message CSFM716I 'ICSF HAS BEEN INITIALIZED WITH *location* FROM *location_info*' indicates which data set has been used for ICSF initialization.
2. Issue the DISPLAY ICSF,SERVICELIBS[,SYSPLEX=YES] command to verify that the 'current' and 'next' values for this LPAR now match.

Deactivating dynamic service update

If you no longer want to use the service data sets (either because LNKLST has been updated with dynamic service update or dynamic service update is not functioning or no longer needed), dynamic service update can be removed in one of two ways:

1. For an active ICSF:
 - a. Update the ICSF installation options data set with the following option:

```
SERVICELIBS(NO)
```

- b. Refresh the options data set to update SERVICELIBS by issuing the following console command:

```
SETICSF OPT,REFRESH
```

- c. Revert to original service by issuing the following console command:

```
SETICSF PAUSE
```

ICSF runs a dynamic service update and at ICSF initialization, ICSF returns to using LNKLST.

2. Set SERVICELIBS(NO) in the ICSF installation options data set, hard stop ICSF, and then start ICSF. ICSF initializes using LNKLST.

Performance considerations for using installation options

You specify installation options in the installation options data set. The CHECKAUTH installation option provides additional security checking, but affects performance.

In ICSF, SAF always checks non-Supervisor State callers. The CHECKAUTH option allows you to specify whether CSF performs access control checking of Supervisor State and System Key callers. Specify CHECKAUTH(NO) if you do not want CSF to check Supervisor State and System Key callers. Specify CHECKAUTH(YES) if you want CSF to check Supervisor State callers. Checking Supervisor State and System Key callers significantly affects performance.

The SYSPLEXCKDS, SYSPLXPKDS and SYSPLEXTKDS options specify whether sysplex-wide data consistency for the CKDS, PKDS, and TKDS is desired. For a description of the subkeywords, see [“Parameters in the installation options data set” on page 35](#).

The RNGCACHE option specifies whether ICSF should maintain a cache of random numbers for services that return or use random numbers. Specifying RNGCACHE(NO) turns off this caching which will decrease performance for services that use random numbers.

Dispatching priority of ICSF

To avoid performance problems, the dispatching priority of ICSF should be set at least as high as that of the highest task using ICSF.

VTAM session-level encryption

ICSF supports VTAM session-level encryption. VTAM session-level encryption provides protection for messages within SNA sessions, that is, between pairs of logical units that support their respective end users. When this method of protection is in effect, data is enciphered by the originating logical unit and deciphered only by the destination logical unit. Thus, the data never appears in the clear while passing through the network.

ICSF places no restrictions on the addressing mode of calling programs. In particular, when VTAM session-level encryption is used with ICSF, VTAM can use storage greater than 16 megabytes.

System SSL encryption

ICSF supports System SSL encryption on all servers. A cryptographic feature is required. For more information, For more information, see [z/OS Cryptographic Services System SSL Programming](#).

Access method services cryptographic option

In compatibility mode, ICSF supports the Access Method Services Cryptographic Option. The option enables the user of the Access Method Services REPRO command to use the Data Encryption Algorithm to encipher data.

The Access Method Services user can use REPRO to encipher data that is written to a data set, and then store the enciphered data set offline. When desired, you can bring the enciphered data set back online, and use REPRO to decipher the enciphered data. You can decipher the data either on the host processor on which it was enciphered, or on another host processor that contains the Access Method Services Cryptographic Option and the same cryptographic key that was used to encipher the data. You can either use ICSF to create the cryptographic keys, or use keys that the Access Method Services user supplies.

With the exception of catalogs, all data set organizations that are supported for input by REPRO are eligible as input for enciphering. Similarly, with the exception of catalogs, all data set organizations supported for output by REPRO are eligible as output for deciphering. The resulting enciphered data sets are always sequentially organized (SAM or VSAM entry-sequenced data sets).

See [Appendix E, “Using AMS REPRO encryption,” on page 497](#) for more information in using this method.

Remote key loading

The process of remote key loading is loading DES keys to automated teller machines (ATMs) from a central administrative site. Because a new ATM has none of the bank's keys installed, getting the first key securely loaded is currently done manually by loading the first key-encrypting key (KEK) in multiple cleartext key parts. A new standard ANSI X9.24-2 defines the acceptable methods of doing this using public key cryptographic techniques, which will allow banks to load the initial KEKs without having to send anything to the ATMS. This method is quicker, more reliable and much less expensive.

Once an ATM is in operation, the bank can install new keys as needed by sending them enciphered under a KEK it installs at a previous time. Cryptographic architecture in the ATMs is not Common Cryptographic Architecture (CCA) and it is difficult to export CCA keys in a form understood by the ATM. Remote key loading will make it easier to export keys to non-CCA systems without compromising security.

In order to use ATM Remote Key Loading, TKE users will have to enable the access control points for these functions:

- Trusted Block Create - API Keyword = Inactive
- Trusted Block Create - API Keyword = Active
- Public Key Import - Source Key Token = Trusted Block
- Public Key Import - Source Key Token = PKA96 Key Token
- Remote Key Export

Event recording

ICSF records certain ICSF events in the System Management Facilities (SMF) data set. ICSF also sends messages that are generated during processing to the ICSF job log and consoles. The SMF recording and messages help you detect problems and track events. This topic describes the events that ICSF records in the SMF record and describes where ICSF sends certain messages.

These records can be used with RACF SMF type 80 record to audit use of the callable services and the keys. The RACF type 80 records are extracted and formatted using the RACF SMF Unload Utility. See [z/OS Security Server RACF Auditor's Guide](#) for information on how to use this utility. For information about the formatted SMF records see [z/OS Security Server RACF Macros and Interfaces](#).

System Management Facilities (SMF) recording

ICSF uses SMF record type 82 to record certain ICSF events. Record type 82 contains:

- A fixed header / self-defining section: This section contains the common SMF record headers fields and the triplet fields (offset/length/number), if applicable, that locate the other sections on the record.
- A ICSF event specific (subtype) section: Each subtype contains information about the event that caused ICSF to write to the SMF record. For subtypes that log state changes, the SMF record will contain additional auditing sections.
- An auditing header section: This section is present in the record for subtypes that log state changes. It describes the number and overall length of the auditing sections that follow.
- A server user section and, optionally, an end user section: If both sections are present, they can appear in either order.

You can map record type 82 by using the CSFZSM82 macro.

ICSF records information in the SMF data set when these events occur:

- ICSF starts or an options refresh.
- An operational key is imported into the CKDS.
- The in-storage CKDS is refreshed.
- A dynamic change is made to a record in the CKDS.
- A dynamic change is made to a record in the PKDS.
- You use the ICSF panels to load master keys on a coprocessor.
- An RSA retained key is created or deleted.
- The CSFPCI callable service issues a coprocessor command request or receives a reply response from a coprocessor.
- A cryptographic processor is configured online or offline.
- ICSF records processing times for coprocessors and accelerators.
- ICSF joins or leaves the ICSF sysplex group.
- A trusted block is created or activated.
- A dynamic change is made to a record in the TKDS.
- Duplicate tokens were detected in a key data set.
- The in-storage PKDS is refreshed.
- Key store policy checking detects the unauthorized use of a key token.
- Key store policy PKA key extensions checking detects the unauthorized use of a key.
- A secure symmetric key token is used for CPACF encryption.
- The TKE workstation sends an audit record to ICSF.
- Key store policy checking detects an attempt to use an archived or inactive KDS record.
- Cryptographic usage statistics are recorded.
- Compliance warning event information is recorded.
- Key life cycle events are recorded.

Each of these events causes ICSF to record information in a separate subtype in the SMF record.

Recording and Formatting type 82 SMF Records in a Report: Sample jobs are available (in SYS1.SAMPLIB) to assist in the recording and formatting of type 82 SMF data:

- CSFSMFJ - JCL that executes the code to dump and format SMF type 82 records for ICSF. Before executing the JCL, you need to make modifications to the JCL (see the prologue in the sample for specific instructions). After the JCL has been modified, terminate SMF recording of the currently active dump dataset (by issuing I SMF) to allow for the unloading of SMF records. After SMF recording has been terminated, execute the JCL. The output goes into the held queue. This is an example of CSFSMFJ.

```

//CSFSMFJ JOB <JOB CARD PARAMETERS>
//*****
//* LICENSED MATERIALS - PROPERTY OF IBM *
//* (C) COPYRIGHT IBM CORP. 2002 *
//* *
//* This JCL reads Type 82 SMF records and formats them in a report.*
//* *
//* CAUTION: This is neither a JCL procedure nor a complete JOB. *
//* Before using this JOB step, you will have to make the following *
//* modifications: *
//* *
//* 1) Add the job parameters to meet your system requirements. *
//* 2) Change the DUMPIN DSN=hlq.smfdata.input to be the name of *
//* the dataset where you currently have SMF data being *
//* recorded. *
//* 3) Change the STEPLIB VOL=SER=ttttt1 and VOL=SER=ttttt2 to *
//* be the volumes where these sort datasets reside. *
//* 4) Change the SYSPROC DSN=hlq.rexx.dataset to be the name of *
//* the dataset where you have placed the CSFSMFR REXX sample. *
//* *
//* Prior to executing this job, you need to terminate SMF *
//* recording of the currently active dump dataset for allow the *
//* unload of SMF records. *
//* *
//*****
//*-----*
//* UNLOAD SMF 82 RECORDS FROM VSAM TO VBS *
//*-----*
//SMFDMP EXEC PGM=IFASMFDP
//DUMPIN DD DISP=SHR,DSN=hlq.smfdata.input
//DUMPOUT DD DISP=(NEW,PASS),DSN=&&VBS,UNIT=3390,
// SPACE=(CYL,(1,1)),DCB=(LRECL=32760,RECFM=VBS,BLKSIZE=4096)
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
// INDD(DUMPIN,OPTIONS(DUMP))
// OUTDD(DUMPOUT,TYPE(82))
//*
//*-----*
//* COPY VBS TO SHORTER VB AND SORT ON DATE/TIME *
//*-----*
//COPYSORT EXEC PGM=SORT,REGION=6000K
//STEPLIB DD DISP=SHR,DSN=SYS1.SORTLPA,VOL=SER=ttttt1,UNIT=3390
// DD DISP=SHR,DSN=SYS1.SICELINK,VOL=SER=ttttt2,UNIT=3390
//SYSOUT DD SYSOUT=*
//SORTWK01 DD UNIT=3390,SPACE=(CYL,10)
//SORTIN DD DISP=(OLD,DELETE),DSN=&&VBS
//SORTOUT DD DISP=(NEW,PASS),DSN=&&VB,UNIT=3390,
// SPACE=(CYL,(1,1)),DCB=(LRECL=32752,RECFM=VB)
//SYSIN DD *
// SORT FIELDS=(11,4,A,7,4,A),FORMAT=BI,SIZE=E4000
//*
//*-----*
//* FORMAT TYPE 82 RECORDS *
//*-----*
//FMT EXEC PGM=IKJEFT01,REGION=5128K,DYNAMNBR=100
//SYSPROC DD DISP=SHR,DSN=hlq.rexx.dataset
//SYSTSPRT DD SYSOUT=*
//INDD DD DISP=(OLD,DELETE),DSN=&&VB
//OUTDD DD SYSOUT=*
//SYSTSIN DD *
// %CSFSMFR

```

- CSFSMFR - An exec that formats the SMF type 82 records into a readable report. Use the report from the ICSF system with the latest FMID running on your sysplex.

ICSF Initialization (Subtype 1)

When ICSF starts, ICSF writes to subtype 1 after initialization is completed. Subtype 1 describes the values of installation options that are specified in the installation options data set.

Subtype 1 contains this information:

- Special secure mode (SSM) option
- SAF checking of Supervisor State and System Key callers (CHECKAUTH) option
- Compatibility mode with CUSP or PCF (COMPAT) option

- Cryptographic domain number (DOMAIN) option
- CKDS name (CKDSN) option
- Maximum length for data in a callable service (MAXLEN) option

Beginning with z/OS V1 R2, the MAXLEN parameter may still be specified in the options data set, but only the maximum value limit will be enforced (2147483647). If a value greater than this is specified, an error will result and ICSF will not start.

- User parameter (USERPARM) option
- PKDS name (PKDSN) option
- TKDS name (TKDSN) option

SMF records for this subtype will also contain a server user audit section.

Operational Key Part Entry (Subtype 7)

ICSF writes to subtype 7 when key parts are entered using the TKE workstation and are processed using the operational key entry ICSF panels. Subtype 7 contains this information:

- The ENC-ZERO verification pattern of the completed key
- A bit indicating whether the verification pattern is valid
- The cryptographic coprocessor domain number
- The cryptographic coprocessor number
- The name of the CKDS that contains the entry with the key part
- The label of the CKDS entry that contains the key part

SMF records for this subtype will also contain server user and end user audit sections.

CKDS Refresh (Subtype 8)

ICSF writes to subtype 8 when the in-storage CKDS is successfully refreshed. ICSF refreshes the in-storage CKDS by reading a disk copy of a CKDS into storage. Subtype 8 contains this information:

- Name of the current in-storage CKDS that ICSF refreshes
- Name of the disk copy of the CKDS that ICSF read into storage to replace the current CKDS

SMF records for this subtype will also contain server user and end user audit sections.

Dynamic CKDS Update (Subtype 9)

ICSF writes to subtype 9 when an application uses the dynamic CKDS update or the KDS metadata write services to write to the CKDS. ICSF writes to subtype 9 when the CKDS KEYS utility is used to update the CKDS. Subtype 9 contains this information:

- Name of the changed CKDS
- An indication of the operation performed.
- The CKDS entry (which includes the label name and key type) that was changed

SMF records for this subtype will also contain server user and end user audit sections.

Dynamic PKDS Update (Subtype 13)

ICSF writes to subtype 13 when an application uses the dynamic PKDS update or the KDS metadata write services to change the PKDS. ICSF writes to subtype 13 when the PKDS KEYS utility is used to update the PKDS. Subtype 13 contains this information:

- The name of the changed PKDS
- An indication of the operation performed.
- The name of the changed entry in the PKDS

SMF records for this subtype will also contain server user and end user audit sections.

Cryptographic Coprocessor Clear Master Key Entry (Subtype 14)

ICSF writes to subtype 14 whenever you use ICSF panels to update AES-MK, DES-MK, ECC-MK, or RSA-MK in the new master key register on a coprocessor. Subtype 14 contains this information:

- The master Key valid indicator
- The type of coprocessor
- The new master key verification pattern
- The key part verification pattern
- The cryptographic coprocessor processor number
- The cryptographic coprocessor serial number
- The cryptographic coprocessor domain index

SMF records for this subtype will also contain server user and end user audit sections.

Cryptographic Coprocessor Retained Key Create or Delete (Subtype 15)

ICSF writes to subtype 15 whenever you create or delete a retained private key in a coprocessor. Subtype 15 contains this information:

- The operation performed (created, deleted from coprocessor, deleted from PKDS)
- The type of coprocessor
- The retained key label
- The cryptographic coprocessor processor number
- The cryptographic coprocessor serial number
- The domain index

SMF records for this subtype will also contain server user and end user audit sections.

Cryptographic Processor Interface Command Request or Reply (Subtype 16)

ICSF writes to subtype 16 whenever the CSFPCI callable service issues a coprocessor command request or receives a reply response from a coprocessor. Subtype 16 contains this information:

- The indicator for request or reply.
- The type of coprocessor.
- The cryptographic coprocessor processor number.
- The cryptographic coprocessor serial number.
- The cryptographic coprocessor domain index.
- The request command block or reply response block length.
- The request command data block or reply response data block length.
- The request or reply CPRB.
- The length of the fixed audit data.
- The number of relocate sections.
- The function id.
- The function return code.
- The function description, which describes the function id.

SMF records for this subtype will also contain server user and end user audit sections.

Cryptographic Coprocessor Configuration (Subtype 18)

ICSF writes subtype 18 when the configuration of a coprocessor or accelerator changes. Subtype 18 contains this information:

- The operation performed (coprocessor brought online, taken offline, had a compliance change).
- The coprocessor number.
- The coprocessor serial number, or accelerator number.

Cryptographic Coprocessor Timing (Subtype 20)

ICSF periodically records processing times for coprocessor or accelerator operations in subtype 20. Subtype 20 contains this information:

- The device type
- The time immediately before the operation begins
- The time immediately after the operation ends
- The time immediately after the results of the operation have been communicated to the caller address space
- The number of processes waiting to submit work to the same coprocessor, domain, and reference slot used by this operation
- The function code for this operation
- The coprocessor or accelerator processor number
- The coprocessor serial number
- The coprocessor or accelerator domain
- A reference number that identifies an internal ICSF queue element

ICSF Sysplex Group (Subtype 21)

ICSF writes subtype 21 when ICSF joins or leaves the ICSF sysplex group. Subtype 21 contains this information:

- The name of the ICSF sysplex group
- The name of the sysplex member
- An indication of whether the member joined or left the sysplex group
- An indication of whether the join or leave was due to normal initialization/termination processing
- An indication of whether the leave was due to error recovery processing
- The time of the join or leave
- The name of the active CKDS

Trusted Block Create (Subtype 22)

ICSF writes subtype 22 when the Trusted Block Create callable services are invoked. Subtype 22 contains this information:

- Type of call, Active or Inactive
- If a Public Key Section was present in the Trusted Block Token
- ASID of the Caller
- If Input Trusted Block Token is in the PKDS, save it's Label
- If Output Trusted Block Token is in the PKDS, save it's Label
- If the Transport Key Token is in the CKDS, save it's Label

SMF records for this subtype will also contain server user and end user audit sections.

Token Data Set (TKDS) (Subtype 23)

ICSF writes subtype 23 when the Token Data Set (TKDS) record is updated using the Token Data Set callable services. ICSF writes to subtype 23 when the PKCS11 TOKEN utility is used to update an object in the TKDS. Subtype 23 contains this information:

- The name of the changed TKDS
- An indication of the operation performed
- The name of the changed entry in the TKDS

SMF records for this subtype will also contain server user and end user audit sections.

Duplicate Key Tokens (Subtype 24)

ICSF writes subtype 24 when the security administrator has indicated that duplicate key tokens must be identified. Subtype 24 contains this information:

- The data set name
- The number of key labels
- The key labels

Key Store Policy Key Token Authorization Checking (Subtype 25)

ICSF writes subtype 25 when a callable service is called and the key token authorization checking detects the key token is not authorized to the caller. The key token is a duplicate of one or more records in the key data set. The check of the CSFKEYS profiles of the record with the key token found the user was unauthorized to use the records. Subtype 25 contains this information:

- Key store and list information.
- The number of key labels.
- The unauthorized duplicate key label and key type.

SMF records for this subtype will also contain server user and end user audit sections.

PKDS Refresh (Subtype 26)

ICSF writes to subtype 26 when the in-storage PKDS is successfully refreshed. ICSF refreshes the in-storage PKDS by reading a disk copy of a PKDS into storage. Subtype 26 contains this information:

- Name of the current in-storage PKDS that ICSF refreshes
- Name of the disk copy of the PKDS that ICSF read into storage to replace the current PKDS

SMF records for this subtype will also contain server user and end user audit sections.

Key Store Policy PKA Key Management Extensions (Subtype 27)

When PKA Key Management Extensions are enabled, ICSF writes to subtype 27 to record operational and error information related to PKA Key Management Extensions. A subtype 27 record is written:

- when a CSF.PKAEXTNS.ENABLE or CSF.PKAEXTNS.ENABLE.WARNONLY profile in the XFACILIT class uses the APPLDATA field to specify a trusted certificate repository, an SMF record is cut to indicate if the trusted certificate repository was successfully changed, or whether there was an error. The APPLDATA field and the repository it specifies will be checked at startup and whenever the XFACILIT class is RACLISTed. ICSF will write a subtype 27 record if the certificate repository is changed, or if there is an error. In this case, subtype 27 will indicate if:
 - the trusted certificate repository was changed
 - the specified trusted certificate repository is empty
 - an error was detected while extracting the APPLDATA
 - the specified repository was not found

- one or more certificates could not be parsed
- when an application calls a service attempting to use a key in a way that is not allowed by the ICSF segment specifications within the CSFKEYS or XCSFKEY profile that covers the key. The SMF record will be written at the completion of the callable service, which, depending on whether PKA Key Management Extensions had been enabled in warning or fail mode, may or may not allow the requested operation on the key. Subtype 27 contains this information. In this case, subtype 27 will indicate if:
 - an asymmetric key may not be used for the requested function
 - a symmetric key cannot be exported by the provided asymmetric key

SMF records for this subtype will also contain server user and end user audit sections.

High Performance Encrypted Key (Subtype 28)

Symmetric Key Encipher (CSNBSYE, CSNBSYE1, CSNESYE and CSNESYE1), Symmetric Key Decipher (CSNBSYD, CSNBSYD1, CSNESYD and CSNESYD1), Field Level Encipher (CSNBFLE, CSNEFLE), and Field Level Decipher (CSNBFLD, CSNEFLD) callable services exploit CP Assist for Cryptographic Functions (CPACF) for improved key management performance. An encrypted DATA key stored in the CKDS can be used in these services, but only when SYMCPCFWRAP(YES) is specified in the ICSF segment of the CSFKEYS class profile that covers the key. For Field Level Encipher and Field Level Decipher, an encrypted DATA key that is not stored in the CKDS can be used, but only when SYMCPCFWRAP(YES) is specified in the ICSF segment of the CSF-PROTECTED-KEY-TOKEN CSFKEYS class profile. ICSF writes to subtype 28 at the completion of functions that attempt to wrap an encrypted key under the CPACF wrapping key. Subtype 28 will indicate if the rewrapping operation is:

- Permitted for this symmetric key
- Not permitted for this symmetric key

SMF records for this subtype will also contain server user and end user audit sections.

For more information about protected-key CPACF, see [z/OS Cryptographic Services ICSF Overview](#).

TKE Workstation Audit Record (Subtype 29)

If you have the optional TKE Workstation, you can use the TKE Audit Record Upload Configuration Utility to send Trusted Key Entry workstation security audit records to an IBM Z host, where they will be saved in the z/OS System Management Facilities (SMF) dataset. Each TKE security audit record is stored in the SMF dataset as a type 82 subtype 29 record. For more information on the TKE Audit Record Upload Configuration Utility, refer to the [z/OS Cryptographic Services ICSF TKE Workstation User's Guide](#).

Key Store Policy Archived and Inactive Checking (Subtype 30)

ICSF writes subtype 30 when a callable service attempts to use an archived record. ICSF writes subtype 30 when a callable service attempts to use an inactive (outside the key material validity dates) record. Subtype 30 contains this information:

- The reference activity.
- Key data set name.
- The entry that was referenced.

SMF records for this subtype will also contain server user and end user audit sections.

Cryptographic usage statistics (Subtype 31)

ICSF writes subtype 31 whenever cryptographic usage tracking is enabled. Each ICSF instance can track the usage of cryptographic engines (ENG), cryptographic services (SRV), and cryptographic algorithms (ALG) for that LPAR. Subtype 31 contains information about the cryptographic user's HOME address space job ID, SECONDARY address space job name, HOME address space user ID, HOME task level user ID, and ASID. See the STATS option in [“Parameters in the installation options data set”](#) on page 35 for more details about enabling these events.

CCA symmetric key lifecycle event (Subtype 40)

ICSF writes subtype 40 whenever a CCA symmetric key undergoes a lifecycle event. A lifecycle event is any event that changes a key, the key's metadata, or the key's state. Examples of lifecycle events include generating a key, updating a key, and a key becoming active. Subtype 40 contains information about the event, information identifying the key, metadata about the key, and information identifying the user. See the AUDITKEYLIFECKDS option in [“Parameters in the installation options data set” on page 35](#) for more details about enabling these events.

SMF records for this subtype will also contain server user and end user audit sections.

CCA asymmetric key lifecycle event (Subtype 41)

ICSF writes subtype 41 whenever a CCA asymmetric key undergoes a lifecycle event. A lifecycle event is any event which changes a key, the key's metadata, or the key's state. Examples of lifecycle events include generating a key, updating a key, and a key becoming active. Subtype 41 contains information about the event, information identifying the key, metadata about the key, and information identifying the user. See the AUDITKEYLIFECPKDS option in [“Parameters in the installation options data set” on page 35](#) for more details about enabling these events.

SMF records for this subtype will also contain server user and end user audit sections.

PKCS #11 key lifecycle event (Subtype 42)

ICSF writes subtype 42 whenever a PKCS #11 key undergoes a lifecycle event. A lifecycle event is any event which changes a key, the key's metadata, or the key's state. Examples of lifecycle events include generating a key, updating a key, and a key becoming active. Subtype 42 contains information about the event, information identifying the key, metadata about the key, and information identifying the user. See the AUDITKEYLIFETKDS option in [“Parameters in the installation options data set” on page 35](#) for more details about enabling these events.

SMF records for this subtype will also contain server user and end user audit sections.

CCA symmetric key usage event (Subtype 44)

ICSF writes subtype 44 whenever a CCA symmetric key is used. Subtype 44 contains information about the event, information identifying the key, metadata about the key, and information identifying the user. See the AUDITKEYUSGCKDS option in [“Parameters in the installation options data set” on page 35](#) for more details about enabling these events.

SMF records for this subtype will also contain server user and end user audit sections.

CCA asymmetric key usage event (Subtype 45)

ICSF writes subtype 45 whenever a CCA asymmetric key is used. Subtype 45 contains information about the event, information identifying the key, metadata about the key, and information identifying the user. See the AUDITKEYUSGPKDS option in [“Parameters in the installation options data set” on page 35](#) for more details about enabling these events.

SMF records for this subtype will also contain server user and end user audit sections.

PKCS #11 key usage event (Subtype 46)

ICSF writes subtype 46 whenever a PKCS #11 key is used. Subtype 46 contains information about the event, information identifying the key, metadata about the key, and information identifying the user. See the AUDITPKCS11USG option in [“Parameters in the installation options data set” on page 35](#) for more details about enabling these events.

SMF records for this subtype will also contain server user and end user audit sections.

PKCS #11 no key usage event (Subtype 47)

ICSF writes subtype 47 whenever a supported PKCS #11 event does not involve a key or object. Subtype 47 contains information about the event and information identifying the user. See the AUDITPKCS11USG option in [“Parameters in the installation options data set” on page 35](#) for more details about enabling these events.

SMF records for this subtype will also contain server user and end user audit sections.

Compliance warning event (Subtype 48)

ICSF writes subtype 48 for CCA compliance warning events. These events can assist when migrating applications to a compliance standard. Subtype 48 contains the result of the operation, information identifying any keys involved, metadata about the key or keys, and information identifying the user. See the COMPLIANCEWARN option in [“Parameters in the installation options data set” on page 35](#) for information about enabling these events.

SMF records for this subtype will also contain server user and end user audit sections.

Master key event resulted in a new master key value being promoted to current (Subtype 49)

ICSF writes subtype 49 for master key events that result in a new master key value being promoted to current. This subtype consists of a number of tag-length-value (TLV) triplets.

Message recording

ICSF writes messages to the job log, and to the security console and the operator console.

ICSF writes most of its messages to the job log. Messages that demand action from the master console operator will display on the operator console, and messages related to system security will display on the security console. Some of these console messages will appear only on the console, and some will also be written to the job log. Messages that are not displayed on either the operator or security console are written to the job log.

For a description of each ICSF message, see [z/OS Cryptographic Services ICSF Messages](#).

Security considerations

You can provide enhanced security on ICSF by controlling access to resources and changing the values of your keys periodically. This topic describes these aspects of security:

- Controlling access to utility programs - KGUP, CSFDUTIL
- Controlling access to the callable services
- Controlling access to cryptographic keys
- Controlling access to CCA key tokens
- Scheduling changes for cryptographic keys
- Controlling access to panel functions
- Controlling access to RACF SMF log records

Controlling the program environment

Some programs or applications, which use ICSF, require that the environment is program controlled. In a program controlled environment, programs within the address space are defined to the Security Server (RACF). Defining a program to RACF requires the program name and the name of the data set that contains the program.

If there is not already an * or ** profile in PROGRAM class, you must define one using RDEFINE instead of RALTER for the first command.

```

RALTER PROGRAM ** ADDMEM('CSF.SCSFMODE' /volser/NOPADCHK)
RALTER PROGRAM ** ADDMEM('CSF.SCSFMOD1' /volser/NOPADCHK)
RALTER PROGRAM ** ADDMEM('CSF.SCSFSTUB' /volser/NOPADCHK)
RDEFINE PROGRAM CSF* ADDMEM('SYS1.SIEALNKE' /volser/NOPADCHK)
RDEFINE PROGRAM CSN* ADDMEM('SYS1.SIEALNKE' /volser/NOPADCHK)

```

The VOLSER specification is optional.

For more information, see [z/OS Security Server RACF Security Administrator's Guide](#).

Controlling access to KGUP

Anyone running the key generator utility program can read and alter an unprotected cryptographic key data set (CKDS). Therefore, only authorized users should have access to the key generator utility program. To make it difficult for an unauthorized person to execute the key generator utility program, store the program in an APF-authorized library that is protected by the Security Server (RACF). Additionally, a security administrator can define a CSFKGUP profile in the CSFSERV class and permit or deny users access to the utility.

Controlling access to CSFDUTIL

CSFDUTIL reads through a CKDS or PKDS and generates a report for duplicate secure key tokens. Only authorized users should have permission to access the CKDS or PKDS datasets directly.

Controlling access to the callable services

Unauthorized persons should not perform the cryptographic or key management functions that the callable services provide. The security administrator should be the only one able to access some services like those used in managing keys. The security administrator can give access to some services, such as enciphering and deciphering data, to persons who are authorized on the system.

You can use the Security Server (RACF) to control which users can use ICSF callable services. For example, you can use the key export service to export any type of key. Your installation may want only the security administrator to be able to use the key export function.

ICSF provides security exit points that you can use to control access to a callable service instead of Security Server (RACF). For information about the security exit points, see [“Security installation exits” on page 197](#).

Your installation may want other users to just be able to export data keys, because sending encrypted data between systems is a common function. The data key export callable service permits the export of data keys only. Your security administrator can have access to the key export service and can use the Security Server (RACF) to give other users access to the data key export service. For more information on controlling who can use ICSF callable services, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

Access control points for specific functions may be enabled/disabled through the TKE workstation. See the [z/OS Cryptographic Services ICSF TKE Workstation User's Guide](#) for additional information.

Controlling access to cryptographic keys

Besides the key generator utility program and services, your installation should also control access to the cryptographic keys. First, it is highly recommended that you store cryptographic keys in data sets that are protected by RACF or an equivalent product. You should limit access to authorized persons or applications. Second, you can use RACF to control access to keys in the in-storage cryptographic key data set. For more information on protecting cryptographic keys, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

When clear DES or AES keys are added to the CKDS, protect all clear DES and AES keys by label name using CSFKEYS profiles on all systems sharing the CKDS.

ICSF also provides security exit points that you can use to control access to keys in the in-storage CKDS and in the PKDS. For information about the security exit points, see [“Security installation exits”](#) on page 197.

Controlling access to secure key tokens

You and your installation have the option of controlling access to a secure tokens that have the same token value and different key labels. To do this, define a key store policy. Key store policy are a system wide setting, using RACF profiles to define the policy. Because key store policy makes use of additional RACF checks, careful planning should occur before implementing the support.

For details on key store policy, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

Scheduling changes for cryptographic keys

You should periodically change the value of cryptographic keys to reduce the possibility of exposing a key value. It is recommended that you change the master keys at least every 12 months.

The security administrator can use the key generator utility program (KGUP) to change the cryptographic keys. KGUP updates keys in the disk copy of the cryptographic key data set while the callable services access keys in the in-storage copy of the cryptographic key data set. Therefore, you can change the keys without affecting cryptographic operations. For more information on using KGUP, refer to [z/OS Cryptographic Services ICSF Administrator's Guide](#).

Controlling access to administrative panel functions

You can perform many ICSF administration functions by using the TSO panels. RACF can protect access to these functions. The functions include:

- Refreshing the CKDS or PKDS
- Setting the master keys
- Changing the master keys
- Clear key entry (access can also be controlled through the TKE workstation, domain controls)
- Pass phrase MK/KDS initialization
- Administrative control functions (enabling and disabling dynamic CKDS access, PKA callable services, and dynamic PKDS access)

These functions are treated the same way as callable services. To view and change system status, see [z/OS Cryptographic Services ICSF Administrator's Guide](#) for more information.

Obtaining RACF SMF log records

For information on how to capture SMF log records for RACF access events, see [z/OS Security Server RACF Auditor's Guide](#) and [z/OS Security Server RACF Command Language Reference](#).

You can extract RACF log records from the SMF data set that can be correlated to the ICSF log records. For more information on how to obtain RACF log records from the SMF data set, see [z/OS Security Server RACF Auditor's Guide](#).

Debugging aids

This topic contains information you can use when diagnosing problems on ICSF. This topic describes:

- Component trace
- Abnormal endings
- Using the IPCS formatting routine
- Detecting ICSF serialization contention conditions

Component trace

ICSF component trace is on all of the time. How much is traced depends on the CTRACE options that are specified in the CTICSFxx parmlib member.

ICSF Component Trace is configured by using a PARMLIB member. A default PARMLIB member, CTICSF00, is shipped and installed with ICSF starting at the ICSF FMID HCR77A1 release level. This PARMLIB member can be specified with the CTRACE option within the ICSF options data set.

Optionally, this PARMLIB member can be copied and customized to a CTICSFxx PARMLIB data set, where xx is a value that is used to make a copy. The new CTICSFxx PARMLIB member can then be specified at ICSF startup time by using the CTRACE option within the ICSF options data set.

For more information on creating a CTICSFxx PARMLIB member, see [“Creating an ICSF CTRACE configuration data set”](#) on page 26.

The TRACEENTRY option in the ICSF Options data set is deprecated. If this option is specified, it is ignored and produces a CSFO0212 message.

ICSF Component Trace can also be dynamically updated by using the TRACE CT command. A CTICSFxx PARMLIB member can be passed to the TRACE CT command. Specific ICSF Component Trace options can also be specified through replies to the TRACE CT command on the operator console.

Following are examples of how to use the TRACE CT command to specify a CTICSFxx PARMLIB member and individual command options.

- To configure ICSF CTRACE to use minimal tracing, use this TRACE OFF command:

```
TRACE CT,OFF,COMP=CSF
```

- To specify a new CTICSFxx PARMLIB member, issue this command:

```
TRACE CT,ON,COMP=CSF,PARM=CTICSFxx
```

- To specify that you want to trace ASID 0042, issue this command:

```
TRACE CT,ON,COMP=CSF
```

Follow the TRACE ON command with this reply:

```
R nn,ASID=(0042),END
```

- To specify that you want to trace JOBNAME MYJOB, issue this command:

```
TRACE CT,ON,COMP=CSF
```

Follow the TRACE ON command with this reply:

```
R nn,JOBNAME=(MYJOB),END
```

- To specify that you want to change the trace buffer size to 250K, issue this command:

```
TRACE CT,250K,COMP=CSF
```

Follow the TRACE command with this reply:

```
R nn,END
```

- To specify that you want to change the trace filtering to CARDIO, issue this command:

```
TRACE CT,ON,COMP=CSF
```

Follow the TRACE ON command with this reply:

```
R nn,OPTIONS=(CARDIO),END
```

- To display the current active trace options, issue this command:

```
DISPLAY TRACE,COMP=CSF
```

Abnormal endings

ICSF has an abnormal ending in these cases only:

- When an error occurs during ICSF initialization.
- When you specify FAIL(ICSF) in the callable service exit installation option.
- When the setting of a cryptographic domain index fails.

If an abnormal end occurs in any other cases, your application or unit of work ends; however, ICSF is still available.

ICSF has an abnormal end code unique to ICSF. Errors specific to ICSF result in an abnormal end code of X'18F' and a unique reason code. In general, all abnormal ends occurring within ICSF result in an appropriate system dump, user dump, or LOGREC recording.

Review the reason code to see whether the abnormal end was an installation or user error. For a list of the reason codes for abnormal end code X'18F', refer to [z/OS MVS System Codes](#). If you cannot resolve the problem, save the dump and contact the IBM Support Center.

IPCS formatting routine

When you look at a dump, you can format ICSF CTRACE entries or request some analysis functions.

CTRACE COMP(CSF) FULL

Formats the trace entries in the trace buffers within the dump or CTRACE captured to an external writer.

This data is most likely to be used by ICSF service.

CTRACE COMP(CSF) OPTIONS((COUNTS))

Shows you which services are being used in the trace entries within the dump.

Sample output:

```
ICSF COUNTS FROM CTRACE:
SERVICE CALLS_FOUND = 00004349
FAILING SERVICES    = 00000145
SERVICE  #SUCCESS  #FAILED
CSFTCTRC  00002106  00000145
CSFTCTRD  00002098  00000000
```

When you see that there are some service failures, you can request more details. If you want more details about who is calling which services, you can use the IPCS statistics support that generates SMF records.

CTRACE COMP(CSF) OPTIONS((FAILURES)) FULL

Sample output:

SYSNAME	MNEMONIC	ENTRY ID	TIME STAMP	DESCRIPTION
S24	ICSF	00000001	11:49:57.925969	ICSF
	ASID H=0047	S=0047	TCB=007B96E8	MOD=CSFVCAPC SrvExit
	Stack=1EEFA010	Service=CSFTCTRC		
	Return code	= 00000008		
	Reason code	= 00000BCD		

This sample output shows that ASID 47 called service CSFPTRC (CSFTCTRC is the internal ICSF routine) and this request failed with return code 8 (application error) and reason code BCD:

```
BCD (3021)
The call to add a z/OS PKCS #11 token failed because the token already
```

exists in the TKDS data space or a request to add a z/OS PKCS #11 token object failed because an object with the same handle already exists.

For more information about return codes, see [z/OS Cryptographic Services ICSF Application Programmer's Guide](#).

CTTRACE COMP(CSF) OPTIONS((PERFC))

Pairs up start and end trace entries for requests to cryptographic coprocessors.

You can use the Interactive Problem Control System (IPCS) to format and display the certain ICSF control blocks. The IPCS CBFORMAT command displays the control block's eye-catcher name, its location in the address space, and its field names with their offsets. ICSF provides format routines for many of its control blocks.

To see a complete list of the control blocks that can be formatted, you can browse SYS1.PARMLIB(CSFIP CSP). This parmlib member provides the definition of all ICSF format and analysis routines to IPCS.

You can also run IPCS command:

```
IPCSDATA CURRENT ACTIVE
```

This command provides an internal view of all format routines and exits defined to IPCS. Look for CSF to find all the routines that are provided by ICSF.

Some ICSF control blocks can be formatted by symbol name. CSFCCVT, CSFCCVE, CSFMGST, and CSFENT can be formatted with the IPCS command below when you specify the control block name as the symbol.

```
CBFORMAT symbolname
```

For example,

```
CBF CSFCCVT
```

This command locates the CCVT and formats it.

Most of the blocks that can be formatted must be located by the user:

```
CBF address STR(structurename)
```

For example,

```
CBF 7F60EF18 STR(CSFZTSKT)
```

For more information about using the CBFORMAT command, see [z/OS MVS IPCS User's Guide](#).

VERBX

ICSF supports two verb exits:

VERBX CSFDATA 'options'

Use to specify a category of control blocks to format. If you specify VERBX CSFDATA without options, the output shows the options that are valid:

```
VERBX CSFDATA
VERBX CSFDATA Output:

No valid options were specified on VERBX CSFDATA.
Valid options are CELL,CCPA,CCPS,CACB,CCPD,OCSV,OCSK,STACKS
```

Each option formats the control blocks that are relevant to the selected option:

CELL

Formats the ICSF storage management cell pool control blocks.

CCPA

Formats the control blocks that keep track of cryptographic coprocessors.

CCPS

Formats the control blocks for active requests to cryptographic coprocessors.

CACB

Formats the control blocks for I/O to CKDS, PKDS, and TKDS.

CCPD

Formats the main control block that is used to manage cryptographic coprocessors.

STACKS

Formats all the dynamic area stacks in ICSF.

VERBX CSF2 'options'

Use to populate the IPCS pointer panel with the control block definitions based on the selected options. If you specify VERBX CSF2 without options, the output shows the options that are valid:

```
VERBX CSF2
VERBX CSF2 Output:

No valid options were specified on VERBX CSF2.
Valid options are BASE,CCMK,PLEX,CKDS,PKDS,TKDS,CARD,KU,STAT,WARN
```

The pointers that are added to the IPCS pointer panel:

BASE

Creates pointers for CCVT, CCVE, and TSKT. Also finds the active stack and related blocks.

CCMK

Creates pointers for any active or residual coordinated change master key requests.

PLEX

Creates pointers for sysplex-related control blocks. Combine with CKDS/PKDS/TKDS.

CKDS

Creates pointers relating to I/O for the KDS. Also, tailors PLEX.

PKDS

Creates pointers relating to I/O for the KDS. Also, tailors PLEX.

TKDS

Creates pointers relating to I/O for the KDS. Also, tailors PLEX.

CARD

Creates pointers for control blocks that are used to keep track of cryptographic coprocessors.

KU

Creates pointers for control blocks that are used for Key Usage tracking.

STAT

Creates pointers for control blocks that are used for Statistics monitoring.

WARN

Creates pointers for control blocks that are used for Warn mode processing.

At a minimum, when you are looking at a dump of ICSF, issue:

```
VERBX CSF2 'BASE'
```

Detecting ICSF serialization contention conditions

If a user task or address space holds an ENQ or latch for an extended period of time, it is likely hung and needs to be canceled so that other work can obtain the ENQ or latch. Some applications might provide controls or document procedures for addressing situations in which the application appears to be gating the rest of the workload. The ICSF system programmer should consult the application's system programmer or administrator regarding actions to take for or against the application. Such action might include stopping or canceling the application.

ICSF requires Global Resource Serialization (GRS) ENQ resources to manage concurrent operations involving the key data sets (CKDS, PKDS and TKDS), and the ICSF ENQ scheme has ICSF itself obtaining

any necessary data set ENQ, in a proxy fashion, on behalf of an application unit of work driving an ICSF API request requiring an ENQ. ICSF also manages any set of extra, different application requests that might be waiting for that same ENQ resource. For this reason, GRS always perceives only ICSF as a key data set ENQ resource owner or waiter, and a DISPLAY GRS,CONTENTION command would not illustrate key data set ENQ contention between two or more competing application requests within a single system scope. For sysplex scope ENQ contention, DISPLAY GRS,CONTENTION would, without any internal assistance, illustrate only ICSF itself as an ENQ holder or waiter, and would not reflect any client application identity or information that is associated with ICSF's ENQ resource usage.

ICSF provides an internal capability to embellish the DISPLAY GRS command output to illustrate the ICSF client applications for which ICSF is holding an ENQ resource, and on the general conditions involving client waiters for an ENQ resource. This enhanced capability is transparently provided and requires no additional ICSF or GRS installation or configuration action. The ICSF support to enhance the DISPLAY GRS output is relevant on a DISPLAY GRS,CONTENTION command only if GRS can detect contention, which is not the case when two or more ICSF client application requests are competing for the same ENQ resource within a single system scope. The ICSF support is relevant on a DISPLAY GRS,RES=(*qname-rname*) command whenever the ENQ resource specified in the *qname-rname* option is held, regardless of whether contention exists. For this reason, the DISPLAY GRS,RES=() command version is recommended as the reliable technique for obtaining information about ICSF key data set ENQ serialization conditions. The DISPLAY GRS command syntax for the various ICSF key data set ENQ resources can be summarized as follows:

Table 13. DISPLAY GRS command syntax ICSF key data set ENQ resources	
This command:	Displays ENQ information for the:
DISPLAY GRS,RES=(SYSZCKT.*)	CKDS
DISPLAY GRS,RES=(SYSZPKT.*)	PKDS
DISPLAY GRS,RES=(SYSZTKT.*)	TKDS

Sample command output for the DISPLAY GRS,RES=(SYSZCKT.*) command:

```
ISG343I 12.01.33 GRS STATUS 360
S=SYSTEM SYSZCKT SYSZCKT
SYSNAME      JOBNAME      ASID      TCBADDR      EXC/SHR
SY1          CSFJM70 /APPL107  0040/0045  007D8E88  EXCLUSIVE
ADDITIONAL RESOURCE INFORMATION FROM:  ICSF Managed ENQ
Owner: APPL107  TTOKEN: 000001200000000300000003007FF050 Waiters: 005
```

In this example, the display command result illustrates that ICSF on system SY1 started under jobname CSFJM70 and executing in ASID 40, has obtained the CKDS ENQ resource exclusively on behalf of the client application running with a jobname of APPL107 and executing in ASID 45. Furthermore, the APPL107 application unit of work that caused ICSF to obtain this ENQ was the task that is identified by task token 000001200000000300000003007FF050, and there are five more application requests on system SY1 that are awaiting access to this ENQ resource.

The DISPLAY GRS,RES=() command must be executed on (or routed to) all of the systems within the scope of a sysplex to obtain the comprehensive understanding of an ICSF key data set ENQ resource.

ICSF also exploits Global Resource Serialization (GRS) latches for serializing resources that are managed within the scope of a single system. In the case of ICSF latches, whenever a client application request requires an ICSF latch for serialization, the latch is obtained under the application's unit of work (not proxied like the ENQ), and therefore, the DISPLAY GRS,CONTENTION command always illustrates the application information for the current latch owner or owners.

The following operational steps are recommended when ICSF serialization contention is suspected as a cause for a workload slowdown or hang:

1. Issue the DISPLAY GRS,CONTENTION command to illustrate sysplex scope contention on ICSF ENQ serialization resources, or system level contention on ICSF latch serialization resources. If the

command result demonstrates latch contention, go to step 3. If the command result demonstrates ICSF key data set ENQ contention and discloses the ENQ owner client application information, go to step 3. If the command result does not demonstrate contention, or does not disclose the ENQ owner client application information, proceed to the next step.

2. Issue the following commands as needed (depending on the key data sets you are using):

```
DISPLAY GRS,RES=(SYSZCKT.*)  
DISPLAY GRS,RES=(SYSZPKT.*) Issue this command only if you are utilizing a PKDS  
DISPLAY GRS,RES=(SYSZTKT.*) Issue this command only if you are utilizing a TKDS
```

The commands need to be executed either on all systems within a sysplex, or on the local system where the ENQ resource is known to be owned. The command result should disclose the ENQ owner client application information.

3. Initiate an action for or against the client application to end the unit of work on behalf of which ICSF has obtained the ENQ resource. Such action might include stopping or canceling the application.

IPCS support for diagnosing contention issues in a dump

ICSF uses GRS ENQs and latches to serialize resources such as the CKDS, PKDS, and TKDS, and serialization for the cryptographic coprocessors. Latches are heavily used to serialize ICSF structures.

When you are looking at a dump, you can use the command ANALYZE. ANALYZE drives analyze exits in IPCS. The GRS analyze exit and the ICSF analyze exit combine data to give you a picture of resources that are in contention. If there is contention, the analyze command generates output. For example,

```
CONTENTION EXCEPTION REPORT  
  
JOBNAME=CSFALLR  ASID=003A  TCB=007D2430  
  
JOBNAME=CSFALLR  HOLDS THE FOLLOWING RESOURCE(S):  
  
RESOURCE #0002:  There are 0020 units of work waiting for this resource  
NAME=SYSZTKT ENQ 7F5B7EC0 STR(DCTL)  
DATA=CSFMISDT task: 007D2430 requested ENQ
```

ICSF manages the ENQ SYSZTKT to serialize access to the TKDS. If you see this sort of problem, enter the following command to load up the pointer panel with control blocks that are related to TKDS updates:

```
ip verbx csf2 'tkds plex'
```

ENF signals

ICSF sends an ENF signal to listeners in the following situations:

- Whenever ICSF is started.
- Whenever ICSF is terminating.
- Whenever a master key is changed.

Listener exit routines are invoked synchronously. Listeners of these signals should follow the guidelines documented in *z/OS MVS Programming: Authorized Assembler Services Guide* on coding listener exit routines. In particular, avoid any processing that may take an extended period of time to complete.

Table 14. ICSF ENF codes

Event code	Description	Qualifier	Parameter list passed to the user exit	Exit type / Cross-system capable
19	ICSF has encountered a change.	None	<p>A four-byte parameter area.</p> <p>X'00000002' ICSF has started and is ready for requests.</p> <p>X'00000003' ICSF is terminating and will no longer accept requests.</p> <p>X'nn000004' One or more master keys (MKs) have been changed. Byte <i>nn</i> indicates the MKs that have been changed as follows.</p> <p>Bit Meaning when set:</p> <p>0 DES MK changed.</p> <p>1 AES MK changed.</p> <p>2 RSA MK changed.</p> <p>3 ECC MK changed.</p> <p>4 P11 MK changed.</p> <p>5-7 Reserved.</p>	EXIT or SRBEXIT / NO

Chapter 5. Installation exits

Your installation can define exit routines to supplement Integrated Cryptographic Service Facility (ICSF), the key generator utility program (KGUP), and the PCF conversion program. Exit routines are programs that programmers at your installation write to allow you to "customize" an application. Your installation may need to perform specific functions with the data that your cryptographic application manipulates. At various points in processing, ICSF, KGUP, and the PCF conversion program release control to an exit routine.

Some common uses for installation exits include:

- Identifying and verifying users.
- Accessing alternate data sets.
- Manipulating input commands.
- Manipulating output data.

This topic describes the various types of exit points in ICSF and the functions that your exits can perform.

Important: Only an experienced system programmer should use the ICSF installation exits. Writing an exit routine and installing a new exit are tasks that require a thorough knowledge of system programming in an ICSF and z/OS environment. An unknowledgeable programmer who attempts to write exit routines or to install new exit points, runs the risk of seriously degrading the performance of your system and causing complete system failure.

Types of exits

ICSF provides several types of exit points:

- Exits that are called during initialization, stopping, and modification of ICSF itself, which are known as the mainline exits
- Exits that are called from the services
- An exit called when a record is read from or written to a fixed length record CKDS.
- An exit called when you update the CKDS with a key that is entered through the key entry hardware or during conversion program processing
- An exit called when records are retrieved from the in-storage CKDS
- Security exits that are called during initialization and stopping of ICSF, during a call to a service, and when accessing a CKDS entry
- An exit called at various points during KGUP processing

These topics briefly describe the different types of exits available in ICSF.

Note: Although IBM no longer supplies security exit routines, the exit points still remain.

Mainline exits

You can supply three exits that are called during ICSF initialization. You can also define an exit routine to run after an operator issues the STOP command and another exit to run after the MODIFY command. Thus, mainline exits can run at these five different points:

- Initialization points
 - Before ICSF initialization
 - After ICSF reads and interprets the installation options
 - Before the completion of ICSF initialization
- When an operator issues a STOP ICSF command

- When an operator issues a MODIFY ICSF command

You can use a mainline exit to alter values in the Cryptographic Communication Vector Table, to end ICSF, or to change ICSF installation options. For more information about the mainline exits, see [“Mainline installation exits” on page 170](#).

Exits for the services

Each of the services in ICSF calls an exit before and after processing. [z/OS Cryptographic Services ICSF Application Programmer's Guide](#) describes the services in greater detail.

You can use a service exit to change, augment, or replace processing or to bypass the IBM-supplied processing for the service entirely. [“Services installation exits” on page 178](#) gives further details about exits for the services.

The PCF CKDS conversion program exit

The PCF conversion program changes a CKDS from PCF to ICSF CKDS format. See [Chapter 8, “Migration from PCF to z/OS ICSF,” on page 227](#) for more information about the conversion program.

ICSF provides three exit points for the same exit routine:

- During the initialization of the conversion program
- While the conversion program is processing individual records
- During the ending of the conversion program

See [“PCF conversion program installation exit” on page 190](#) for more information about the conversion program installation exit (CSFCONVX).

The single-record, read-write exit

Certain ICSF processes read records from or write records to the CKDS. These processes include running a conversion program, refreshing and reenciphering the CKDS, and using the key entry hardware to enter a key. When these processes read or write CKDS records, they call the exit. You can customize the processing of a CKDS record read-write with the single-record, read-write exit (CSFSRRW). See [“Single-record, Read-write installation exit” on page 193](#) for more information about the single-record, read-write exit.

Note: This exit is given control only for a fixed-length record CKDS. The exit is not given control for the variable-length record format or KDSR format of the CKDS.

The cryptographic key data set entry retrieval exit

You can use certain services to manage keys on ICSF. A service can access a key in the in-storage CKDS by specifying a key label. For more information about the services, see [z/OS Cryptographic Services ICSF Application Programmer's Guide](#).

When a service requests a record from the in-storage CKDS by label, ICSF calls the CKDS entry retrieval exit. For instance, you can use this exit to perform a specific search of the installation data field in the record. See [“Cryptographic key data set entry retrieval installation exit” on page 188](#) for more information about the CKDS entry retrieval exit.

Note: This exit is given control only for a fixed-length record CKDS. The exit does not work with the variable-length record format of the CKDS.

Security exits

You can supply four different exits to control access to resources on ICSF. ICSF calls the security exits at these points:

- During CSF initialization
- During CSF termination

- When an application calls an ICSF service
- When an entry in the in-storage CKDS is accessed

See [“Security installation exits” on page 197](#) for more information about the security exits.

The KGUP exit

You use KGUP to generate and maintain keys in the CKDS. KGUP creates key values that systems can use in key exchanges. The ICSF administrator uses job control language to start KGUP and specifies information to KGUP through the use of a control statement.

As opposed to the five different mainline exits, ICSF provides one exit for KGUP processing that is called at four different points. ICSF calls the KGUP exits at these points:

- During KGUP initialization
- Before KGUP processes a key that is identified by a control statement
- Before KGUP updates the CKDS
- During KGUP termination

The KGUP exit receives a parameter that identifies the exit's calling point. Thus, the installation exit can perform different functions at each of the calls.

You can use the KGUP exit to change key values, make a copy of a CKDS entry, or end KGUP. [“Key generator utility program installation exit” on page 201](#) gives a more detailed description of the KGUP exit.

Entry and return specifications

All of the exits described in [“Types of exits” on page 165](#) use standard linkage conventions on entry and return from the exits.

Registers at entry

The mainline exits have these register contents on entry:

Register

Contents

0

Address of the exit parameter block (EXPB)

1

Address of a parameter list

2–12

Not applicable

13

Address of register save area

14

Return address

15

Entry point address

The service exits have these register contents on entry:

Register

Contents

0

Address of the exit parameter block (EXPB)

1

Address of a parameter list

2-13

Not applicable

14

Return address

15

Entry point address

The CKDS entry retrieval installation exit has these register contents on entry:

**Register
Contents**

0

Not applicable

1

Address of a parameter list

2-12

Not applicable

13

Address of register save area

14

Return address

15

Entry point address

The conversion program, single-record, read-write, and KGUP exits have these register contents on entry:

**Register
Contents**

0

Not applicable

1

Address of a control block (CVXP, RWXP, or KGXP, depending on the exit)

2-12

Not applicable

13

Address of register save area

14

Return address

15

Entry point address

The particular control blocks that are passed through register 0 or register 1 are described with each exit.

Registers at return

Registers for all exits must contain the original contents on entry with the exception of register 15 which must contain a valid return code. See each exit for a list of valid return codes. The registers should contain this information on return.

**Register
Contents**

0-14

Same as entry contents

15

Valid return code

Exits environment

ICSF calls different types of exits in distinct environments. The exits differ regarding the mode in which they run and how they address data.

Mainline exits

ICSF mainline exits run in task mode in the ICSF address space. All the passed storage pointers specify addresses in the ICSF address space and are not ALET qualified. There are essentially no restrictions on the use of z/OS services for these exits.

Service exits

ICSF calls the service exits in cross memory mode after a space switch PC. The exits run in the ICSF address space, which is the primary address space. The exits need to address parameters in the caller's address space, which is the secondary address space. In general, user-passed parameters, including the parameter list itself, are in the secondary address space. An exit that is running in access register (AR) mode using an ALET of 1 can access these parameters. For information about cross memory mode and AR mode, see *z/OS MVS Programming: Extended Addressability Guide*.

CKDS entry retrieval exit

The exit runs in cross memory mode. The addresses of the CKDS records that are used by the exit are ALET-qualified. The exit receives both the current CKDS record address and the record's associated ALET as parameters in the exit parameter list. The exit must run in AR mode, and must use the information passed in the exit parameter list to access CKDS entries. For information about cross memory mode and AR mode, see *z/OS MVS Programming: Extended Addressability Guide*.

KGUP, Conversion Programs, and Single-record, Read-write exits

The exits run in task mode in the caller's home address space. The exits do not run in cross memory mode and are not passed ALET-qualified storage pointers. There are essentially no restrictions on the use of z/OS services for these exits.

Security exits

The initialization and termination security exits run in task mode in the ICSF address space. The passed storage pointers specify an address in the ICSF address space and are not ALET-qualified. There are essentially no restrictions on the use of z/OS services for these exits.

ICSF calls the security service exit and the security keys exit in cross memory mode after a space switch PC. The security service exit runs in the ICSF address space, which is the primary address space. The security key exit runs in cross memory and AR mode.

Exit recovery

An ESTAE routine provides recovery for the mainline exits; the single-record, read-write exit; and the security initialization and termination exits. If an exit ends abnormally, the ESTAE routine intercepts the abnormal ending code and schedules a system dump. If the conversion program exit ends abnormally, the conversion program ends abnormally. If the KGUP exit ends abnormally, KGUP also ends abnormally. ESTAE routines provide recovery for the conversion program and KGUP.

The ICSF Functional Recovery Routine (FRR) provides recovery for the service exits, the CKDS entry retrieval exit, and the security service and key exits. If an exit ends abnormally, the FRR intercepts the abnormal ending code and schedules a system dump.

There are times during ICSF processing that ICSF suppresses dumps. For example, ICSF does not schedule dumps when integrity checking user data. This action avoids the possibility of user errors that

can severely affect system performance. However, ICSF does write a record to SYS1.LOGREC if the error occurs.

When writing exits, you may also want to suppress dumps under certain circumstances. You can suppress dumps by setting a bit on in the SPB. This bit, the SPBTERM bit, is the third bit of the flag byte at offset 18 in the SPB. An exit might want to suppress dumps whenever the exit writes user storage. The exit can turn the bit on before the WRITE instruction and turn the bit off again after the instruction.

Mainline installation exits

ICSF begins when an operator issues a START command from the operator console. When ICSF issues this command, the initialization process begins.

After ICSF starts, operators can issue the MODIFY or STOP commands. You can define installation exits to customize ICSF at the initialization, stopping, and modification points.

Purpose and use of the exits

ICSF calls the mainline exits during the startup, modification, and shutdown stages. The exits allow your installation to change the initialization options, issue special messages, and bypass operator commands. This is a description of each point at which ICSF calls mainline exit routines.

CSFEXIT1

ICSF calls this exit after an operator issues a START command, but before any processing takes place. You can use this exit to change the allocation of the installation options data set. If CSFEXIT1 changes the DDNAME, it is treated as CSFPARM DD throughout ICSF initialization.

ICSF always calls the exit. If this exit does not exist, ICSF continues normal processing. If this exit exists, ICSF starts it.

CSFEXIT2

ICSF calls this exit during the initialization process after the installation options data set is read and interpreted. You can use this exit to change certain installation options.

CSFEXIT3

ICSF calls this exit just before ICSF initialization is complete. You can use this exit to issue commands to start other cryptographic work.

CSFEXIT4

ICSF calls this exit when an operator issues a STOP command. You can use this exit to decide to allow or disallow the STOP command.

CSFEXIT5

CSFEXIT5 receives the command input block (the string that is entered by the operator), so you can customize CSFEXIT5 to perform any processing you require. ICSF calls this exit when an operator issues a MODIFY command. ICSF provides the MODIFY command exit to allow each installation the flexibility of defining its own command. ICSF does no processing when an operator uses the MODIFY command. The MODIFY command is simply a call to CSFEXIT5.

Environment of the exits

The exits receive control with these characteristics:

- Supervisor state
- Key 0

- APF-authorized
- TCB mode
- Address Space Control mode=access register mode
- AMODE(31) or AMODE(64)

The exit receives control in AMODE(64) if the service was invoked in AMODE(64); otherwise the exit receives control in AMODE(31). If you have a callable service exit for a service which supports invocation by an AMODE(64) caller, once HCR7720 is installed, you should recode your exit to be sure it can handle being invoked in AMODE(64).

- RMODE(ANY)

The exits can change the characteristics during their processing. However, the exits must return to ICSF with the same characteristics as on entry.

Installing the exits

Because ICSF calls CSFEXIT1 before any initialization occurs, the exit is not defined in the same way as the other exits. For all the mainline exits, install the load module that contains the exit into an APF-authorized library. ICSF uses this normal z/OS search order to locate the exit:

- Job pack area
- Steplib (if one exists)
- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

You must define CSFEXIT2, CSFEXIT3, CSFEXIT4, and CSFEXIT5 in the installation options data set. However, you *must not* define CSFEXIT1 in the installation options data set, and the load module name for the exit must be CSFEXIT1.

To define the exits in the installation options data set, define the ICSF exit point name and load module name on the EXIT keyword in the installation options data set. For information about the installation options data set, see [“Parameters in the installation options data set” on page 35](#). The EXIT keyword has this syntax:

EXIT (ICSF exit point name, load module name, FAIL (options))

The **ICSF exit point name** portion of the keyword refers to the ICSF name for each exit, CSFEXIT2, CSFEXIT3, CSFEXIT4, and CSFEXIT5. The **load module name** is the name of the load module that contains the exit. The name can be any valid name your installation chooses. The **FAIL** portion of the EXIT keyword specifies the action ICSF takes if the exit cannot be loaded. The valid FAIL options are:

NONE

Initialization continues even if exits cannot be loaded.

SERVICE

Initialization continues even if exits cannot be loaded.

EXIT

Initialization continues even if exits cannot be loaded.

ICSF

End ICSF if exits cannot be loaded.

You must specify a FAIL option. If you do not, ICSF returns an error message, abnormally ends, and generates an SVC dump when attempting to load the exit.

Input

All mainline exits receive the address of an exit parameter block (EXPB) passed in register 0. Each exit receives the address of an address list passed in register 1. Each address in the list points to a parameter.

[Figure 3 on page 172](#) illustrates the contents of register 0 and EXPB for the mainline exits.

Register 0

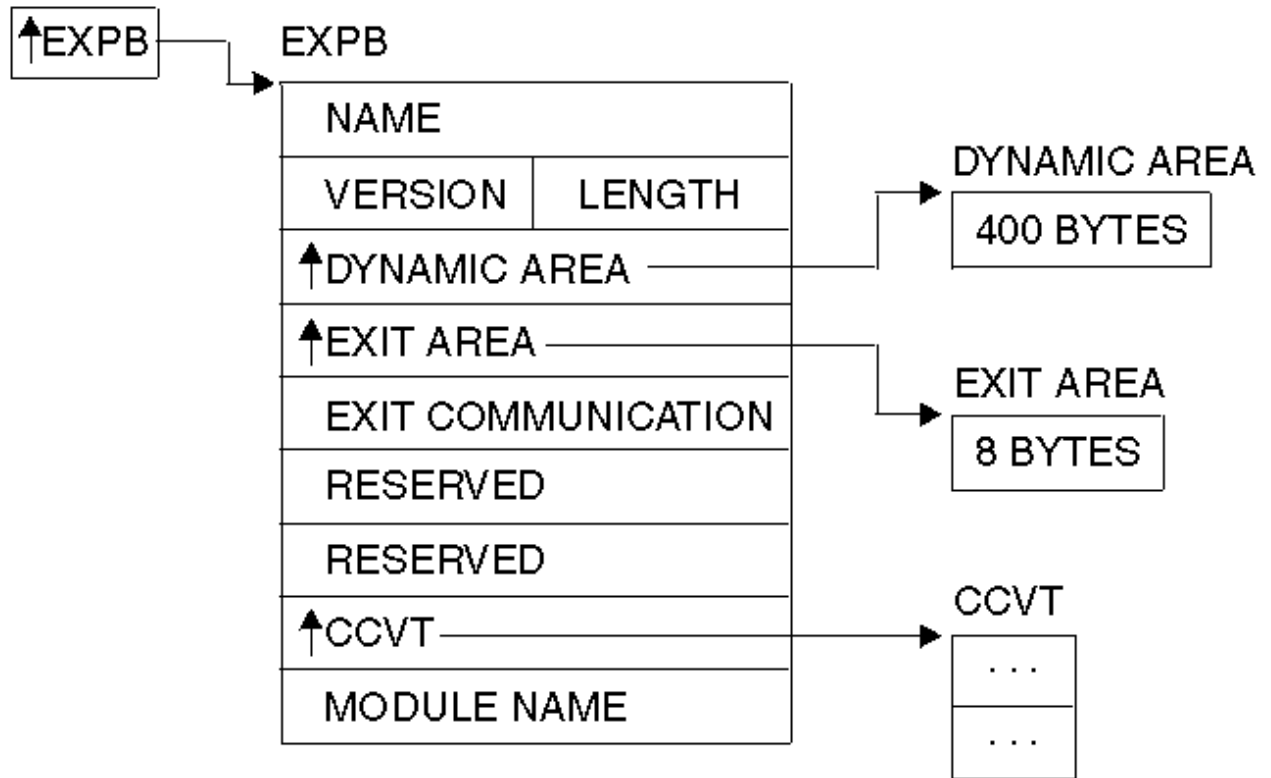


Figure 3. EXPB control block for mainline exits

Both the mainline exits and the services exits receive the address of EXPB in register 0. Some of the fields in EXPB are used only by the service exits and are reserved fields for the mainline exits.

The Exit Parameter Block

Table 15 on page 172 describes the contents of the exit parameter block.

Table 15. EXPB Control Block format for Mainline Exits		
Offset (Dec)	Number of Bytes	Description
0	4	Name. The name of the control block. This field contains the character string EXPB.
4	2	Version. The version of the control block. This field contains the character string 01.
6	2	Length. The length of the control block. The value of this field is 40 in decimal.
8	4	Dynamic area address. The address of a 400-byte area that the exit can use as a dynamic area.

Table 15. EXPB Control Block format for Mainline Exits (continued)		
Offset (Dec)	Number of Bytes	Description
12	4	Exit area address. The address of an 8-byte area the exits can use to communicate with each other. ICSF does not check or change this field.
16	4	Exit communication area. A character string that can be used for communication between the exits. The field is initialized to zero before CSFEXIT1 is called, and ICSF does not modify this field.
20	4	Flags. Reserved. The flag field is used only by the exits for the services. The field contains binary zeros for the mainline exits.
24	4	Secondary parameter block (SPB) address. Reserved. The SPB is used only by the exits for the services. The field contains binary zeros for the mainline exits.
28	4	CCVT address. Address of the Cryptographic Communication Vector Table (CCVT). “The Cryptographic Communication Vector Table (CCVT)” on page 400 describes the CCVT in greater detail.
32	8	Module name. The installation exit's load module name. The field contains the value of the load module name you specified on the EXIT keyword in the installation options data set. The field is 8 bytes of characters, and the value is left-justified and padded with blanks.

Parameters

All mainline exits receive an address list that uses standard entry linkage. Register 1 points to the address list. Each address in the list points to a parameter. Tables in the next four topics describe the parameters for each of the mainline exits.

CSFEXIT1

This table describes the parameters for CSFEXIT1:

Table 16. CSFEXIT1 parameters		
Parameter	Number of Bytes	Description
1	8	The data set name (DDNAME) of the installation options data set.
2	Variable	The command input block for the START command. The command control block is mapped by IEZCIB.

When ICSF calls this, the Cryptographic Communication Vector Table exists, but the table is not yet complete.

CSFEXIT2 and CSFEXIT3

Both CSFEXIT2 and CSFEXIT3 receive the same parameters. [Table 17 on page 174](#) describes these parameters.

Table 17. CSFEXIT2 and CSFEXIT3 parameters		
Parameter	Number of Bytes	Description
1	44	A character string that is the CKDS name specified in the CKDSN installation option.
2	4	A decimal value that is the maximum length permitted for data passed to services specified in the MAXLEN installation option. Beginning with z/OS V1 R2, the MAXLEN parameter may still be specified in the options data set, but only the maximum value limit will be enforced (2147483647). If a value greater than this is specified, an error will result and ICSF will not start.
3	4	ICSF environmental options. Note: Do not change bits 1 - 5. Byte 1: Bit Meaning When Set On 0 Special secure mode enabled. 1 - 5 Reserved. 6 SAF checking required for authorized callers. 7 PCF coexistence. Bytes 2–4: Reserved
4	4	Address of the exit name table. Table 19 on page 175 describes the exit name table.

CSFEXIT4 and CSFEXIT5

Both CSFEXIT4 and CSFEXIT5 receive the same parameters. [Table 18 on page 174](#) describes these parameters.

Table 18. CSFEXIT4 and CSFEXIT5 parameters		
Parameter	Number of Bytes	Description
1	44	A character string that is the CKDS name specified in the CKDSN installation option.

Table 18. CSFEXIT4 and CSFEXIT5 parameters (continued)

Parameter	Number of Bytes	Description
2	4	A decimal value that is the maximum length permitted for data passed to services specified in the MAXLEN installation option. Beginning with z/OS V1 R2, the MAXLEN parameter may still be specified in the options data set, but only the maximum value limit will be enforced (2147483647). If a value greater than this is specified, an error will result and ICSF will not start.
3	4	ICSF environmental options. Note: Do not change bits 1 - 5. Byte 1: Bit Meaning When Set On 0 Special secure mode enabled. 1 - 5 Reserved. 6 SAF checking required for authorized callers. 7 PCF coexistence. Bytes 2–4: Reserved
4	4	Address of the exit name table. Table 19 on page 175 describes the exit name table.
5	Variable	The command input block. You can use the IEZCIB mapping macro to map the control block.

The Exit Name Table

The exit name table contains a list of all of the exits and their load module names. [Table 19 on page 175](#) describes the format of the exit name table.

Table 19. Format of the Exit Name table

Offset (Dec)	Number of Bytes	Description
0	4	Exit name table ID. The value is always the character string ENT.
4	2	Exit name table version. The value is always the character string 01.
6	2	Length of the exit name table. This value is in decimal.
8	4	Number of entries in the array which is the number of exits ICSF supplies. This value is in decimal.
12	4	Subpool that the exit name table is in.
16	4	Reserved.
20	4	Reserved.

Table 19. Format of the Exit Name table (continued)

Offset (Dec)	Number of Bytes	Description
24	4	Reserved.
28	4	Reserved.
32	8	ICSF exit name 1. This value is a character string.
40	8	Installation load module name 1. This value is a character string.
48	4	<p>Flags.</p> <p>Flag bytes. Only the first two bytes are used; bytes 3 and 4 are reserved.</p> <p>Byte 1:</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0</p> <p>Exit has been requested by the installation.</p> <p>1</p> <p>Exit has been loaded.</p> <p>2</p> <p>Exit is active.</p> <p>3</p> <p>If exit fails, end ICSF.</p> <p>4</p> <p>If exit fails, do not call the exit again.</p> <p>5</p> <p>If exit fails, fail the service.</p> <p>6</p> <p>If exit fails, do nothing.</p> <p>7</p> <p>Exit has failed previously.</p> <p>Byte 2:</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0</p> <p>The exit should be called.</p> <p>1</p> <p>The exit is available to the installation.</p> <p>2</p> <p>If the security exit fails, fail the service.</p> <p>3–7</p> <p>Reserved.</p>
52	4	Address of the exit.
56	4	Reserved.
60	4	Reserved.
64	8	ICSF exit name 2. This value is a character string.

Table 19. Format of the Exit Name table (continued)

Offset (Dec)	Number of Bytes	Description
72	8	Installation load module name 2. This value is a character string.
80	4	Flags. See offset +48 for flag byte definitions.
84	4	Address of the exit.
88	4	Reserved.
92	4	Reserved.
x	8	ICSF exit name a.
x+8	8	Installation load module name a.
x+16	4	Flags. See offset +48 for flags.
x+20	4	Address of the exit.
x+24	4	Reserved.
x+28	4	Reserved.

Return Codes

All mainline exits can pass back a return code in register 15. CSFEXIT1, CSFEXIT2, and CSFEXIT3 support these decimal return codes:

Return Code Description

0
Proceed with initialization.

16
End ICSF.

Any return codes other than those listed cause ICSF to end abnormally.

CSFEXIT4 supports these decimal return codes:

Return Code Description

0
Proceed with the STOP command.

4
Do not allow the STOP command to proceed.

Any return codes other than those listed cause processing of the STOP command to end abnormally.

CSFEXIT5 supports these decimal return codes:

Return Code Description

0
Continue processing.

4
End ICSF.

Any return codes other than those listed cause processing of the MODIFY command to end abnormally.

Services installation exits

ICSF provides services that you can use to perform various cryptographic functions. Examples of these functions include enciphering and deciphering data, generating and verifying message authentication codes, generating and verifying PINs, and dynamically updating the CKDS and PKDS. You can define an installation exit for each of the services to customize processing.

ICSF provides a single service exit called CSF_SERVICE_EXIT that gets control for all services. The intent of this exit is for statistics generation. For more information, see [“CSF_SERVICE_EXIT - ICSF callable services exit” on page 186](#).

For a detailed description of the services, see [z/OS Cryptographic Services ICSF Application Programmer's Guide](#).

Use this general format to request a service:

```
CALL CSNBxxx (
    return_code
    ,reason_code
    ,exit_data_length
    ,exit_data
    ,parameter_5
    ,parameter_6
    .
    .
    ,parameter_N)
```

[Appendix G, “Resource names for CCA and ICSF entry points,” on page 503](#) lists the ICSF exit names for each of the services. The parameters that the application passes to a service are known as the service parameter list. The parameters vary from service to service. [“Parameters” on page 185](#) describes the services parameter lists in more detail.

Purpose and use of the exits

Each of the services has an installation exit. Each installation exit for a service has two exit points:

- **The Preprocessing exit point.** This exit point occurs after an application program calls a service, but before the service starts processing. For example, you can use this exit point to check or change the parameters that the application passes on the call, or to end the call. You can also perform additional security checks.
- **The Postprocessing exit point.** This exit point occurs after the service has finished processing, but before the service returns control to the application program. For example, you can use this exit point to check and change the return code from the service or perform cleanup processing.

Environment of the exits

The exits receive control with these characteristics:

- Supervisor state
- Key 0
- APF-authorized
- TCB or SRB mode
- Cross memory mode
- AR mode
- AMODE(31) or AMODE(64)

The exit receives control in AMODE(64) if the callable service was invoked in AMODE(64); otherwise the exit receives control in AMODE(31). If you have a callable service exit for a service which supports

invocation by an AMODE(64) caller, you must recode your exit to be sure it can handle being invoked in AMODE(64).

- RMODE(ANY)

The exits can change the characteristics during their processing. However, the exits must return to their caller with the same characteristics as on entry.

You must write the exits in assembler, because you are in AR and cross memory mode and the addresses of some of the parameters you may access are ALET-qualified. In particular, parameters passed into a service are in the user's address space which you can access with an ALET of 1.

For information about cross memory and AR mode, see [z/OS MVS Programming: Extended Addressability Guide](#).

Installing the exits

You install an exit for a service by installing the load module that contains the exit into an APF-authorized library. ICSF uses this normal search order to locate the exit:

- Job pack area
- Steplib (if one exists)
- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

Define the ICSF name and the load module name as a value on the EXIT keyword in the installation options data set. For more information about the installation options data set, see [“Parameters in the installation options data set” on page 35](#). The EXIT keyword has this syntax:

EXIT (ICSF name, load module name, FAIL (options))

The **ICSF name** portion of the keyword refers to the ICSF name for each service exit. Note that the ICSF name for each service exit is the same as its name. [Appendix G, “Resource names for CCA and ICSF entry points,” on page 503](#) lists the ICSF names for each of the service exits. [Table 20 on page 180](#) lists the ICSF names for each of the compatibility service exits. The **load module name** is the name of the load module that contains the exit. The name can be any valid name that your installation chooses. The **FAIL** portion of the EXIT keyword specifies the action ICSF takes if the exit cannot be loaded or it ends abnormally. When the exit fails to load, the valid FAIL options mean:

NONE

Initialization continues. The exit is not available to be called.

EXIT

Initialization continues. The exit is not available to be called.

SERVICE

Initialization continues. The exit is not available to be called.

ICSF

ICSF is ended.

When the exit ends abnormally, the valid FAIL options are:

NONE

No action is taken. The exit can be called again and will end abnormally again.

EXIT

The exit is no longer available to be called again.

SERVICE

The service or program that called the exit is no longer available to be called again.

ICSF

ICSF is ended.

You must specify a FAIL option. If you do not, ICSF returns an error message, ends abnormally, and generates an SVC dump when attempting to load the exit. If the exit ends abnormally, the service call fails regardless of the fail option you specified. Fail options apply only to subsequent requests for the service.

Table 20. Compatibility services and their ICSF names	
Compatibility Service	ICSF Name
Encipher under Master Key	CSFEMK
Generate a key	CSFGKC
Import a key	CSFRTC
Cipher/Decipher	CSFEDC

Input

The installation exit for each service gets the address of the exit parameter block (EXPB) in register 0. ICSF obtains and initializes an EXP for every service call. [Figure 4 on page 180](#) illustrates the contents of register 0, and [Table 21 on page 181](#) illustrates the EXPB for the service exits.

Register 1 contains the address of an address list. Each address in the list points to a parameter. “Parameters” on [page 185](#) describes the service parameter list. The parameters the exit receives are the same parameters that are passed on the call to the service. For more information about the parameters for each service, see [z/OS Cryptographic Services ICSF Application Programmer's Guide](#).

Register 0

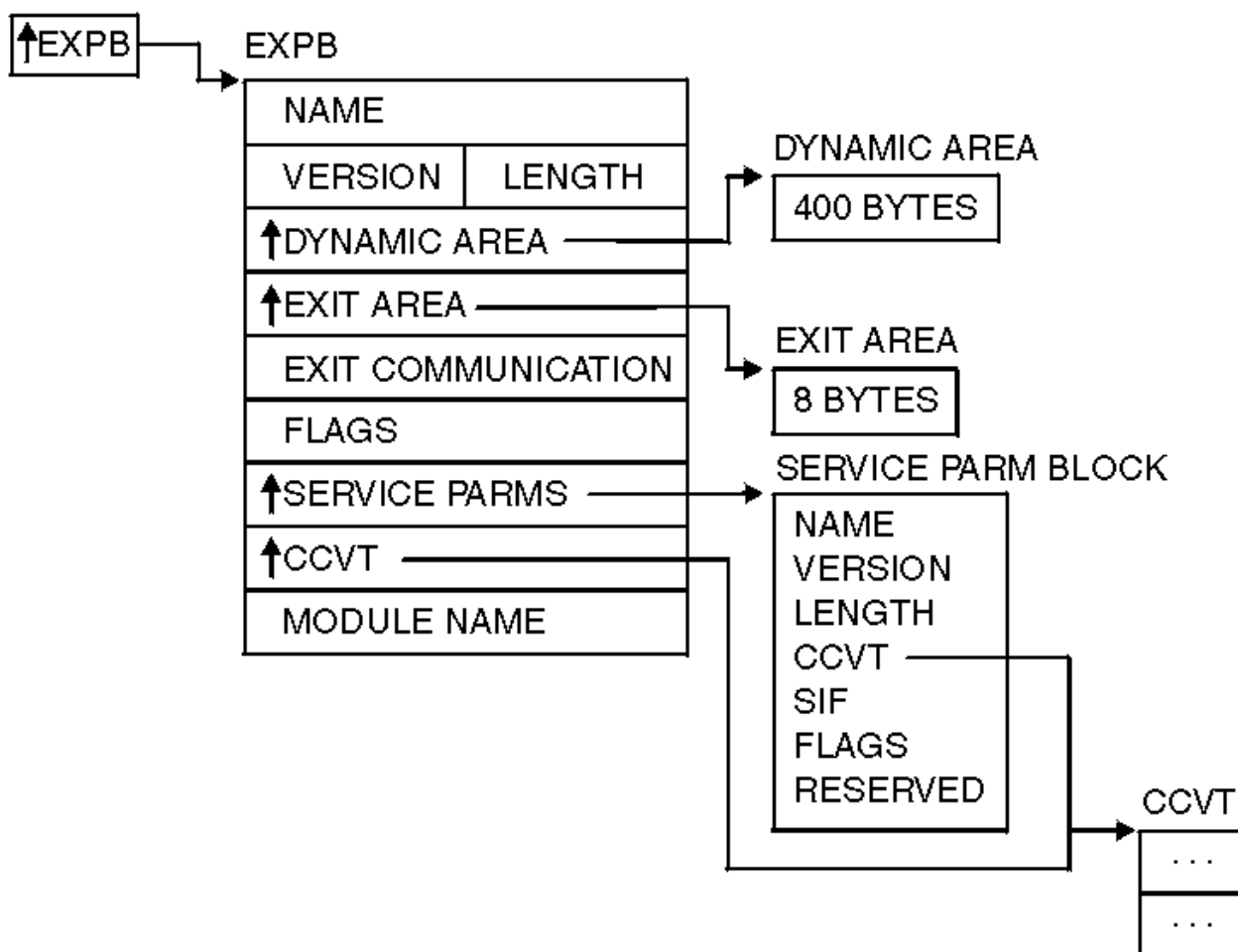


Figure 4. EXPB control block in the service exits

Exit parameter block

Table 21 on page 181 describes the contents of the exit control block.

Table 21. EXPB Control Block Format for Services		
Offset (Dec)	Number of Bytes	Description
0	4	Name. The name of the control block. The field contains the character string EXPB.
4	2	Version. The version of the control block. The field contains the character string 01.
6	2	Length. The length of the control block. The value is 40 in decimal.
8	4	Dynamic area. The address of a 400-byte area that the exit can use as a dynamic area.
12	4	Exit area address. The address of an 8-byte area for the preprocessing and postprocessing invocations of the exit to use for communication. ICSF does not check or change this field.
16	4	Exit communication area. A character string that can be used for communication between preprocessing and postprocessing invocations of a service exit.

Table 21. EXPB Control Block Format for Services (continued)

Offset (Dec)	Number of Bytes	Description
20	4	<p>Flags.</p> <p>A flag byte. Each bit setting (on/off) indicates a particular condition. ICSF sets bit 0 and an exit cannot change that bit. Your exit can set any of the other bits.</p> <p>Bit</p> <p>Meaning When Set On/Off</p> <p>0 Postprocessing invocation./Preprocessing invocation.</p> <p>1 Reserved.</p> <p>2 Use the return and reason code that the exit places in register 0 and register 15 as the service's return code/reason code. Do not use the exit's return code as the service return code in registers 0 and 15.</p> <p>The exit can pass any valid return code in register 15 and any valid reason code in register 0. If this bit is set on, ICSF uses these codes as the service's return and reason codes. See "Return Codes" on page 185 for more information about using exit return codes.</p> <p>3 Do not call the postprocessing invocation of the service exit./Call the postprocessing invocation of the service exit.</p> <p>4 Bypass the service./Run the service.</p> <p>5 Use the return and reason code that the exit places in the service's parameter list./Do not store codes the exit places in the service's parameter list.</p> <p>The exit can pass any valid return and reason code in the first two parameters of the service's parameter list. "Parameters" on page 185 describes the service parameter list.</p> <p>6 CSFSKRC bypass input label parsing./CSFSKRC parse the input label.</p> <p>7–31 Reserved.</p>
24	4	<p>Secondary parameter block.</p> <p>The address of the secondary parameter block. The exit can use the SPB to determine the environmental information of the service. For a description of the SPB, see "Secondary parameter block" on page 183.</p>

Table 21. EXPB Control Block Format for Services (continued)

Offset (Dec)	Number of Bytes	Description
28	4	CCVT. Address of the Cryptographic Control Vector Table (CCVT). For a description of the CCVT, see “The Cryptographic Communication Vector Table (CCVT)” on page 400.
32	8	Module name. The installation exit's load module name. The field contains the value of the load module name you specified on the EXIT keyword in the installation options data set. The field is 8 bytes of characters, and the value is left-justified and padded with blanks.

Secondary parameter block

Offset +24 of EXPB contains the address of the secondary parameter block (SPB). The exit can use the SPB to determine the environmental conditions of the service. [Table 22 on page 183](#) describes the contents of SPB.

Table 22. SPB Control Block Format

Offset (Dec)	Number of Bytes	Description
0	4	Name. The name of the control block. The field contains the character string SPB.
4	2	Version. The version of the control block. The field contains the character string 04.
6	2	Length. The length of the control block.
8	4	CCVT. The address of the Cryptographic Communication Vector Table (CCVT). For a description of the CCVT, see “The Cryptographic Communication Vector Table (CCVT)” on page 400.
12	4	Signal Information Word. Bytes 1–2 Reserved. Bytes 3–4 of the field contain the installation-assigned code number for an installation-defined service.

Table 22. SPB Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
16	4	<p>Flags and Indicators. Each byte of this field is either an indicator byte or contains flag bits. The contents of each byte in the field are:</p> <p>Byte 1—PSW key. This byte contains the original caller's program status word key. The first four bits are the key and the remaining four bits are zeros.</p> <p>Byte 2—Caller's state. Each bit in byte 2 indicates a condition of the caller's state.</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0</p> <p>ICSF was entered via SVC entry from a PCF compatibility macro.</p> <p>1</p> <p>Original caller in AMODE(31).</p> <p>2</p> <p>Original caller in AR mode.</p> <p>3</p> <p>Original caller in SRB mode.</p> <p>4</p> <p>Original caller in supervisor state or system key.</p> <p>5</p> <p>Original caller in AMODE(64).</p> <p>6–7</p> <p>Reserved.</p> <p>Byte 3—Flag byte 1. The first flag byte. Each bit that is set on indicates a particular condition.</p> <p>Note: These bits are informational. Do not change bits 0 and 1.</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0</p> <p>Reserved.</p> <p>1</p> <p>Reserved.</p> <p>2</p> <p>The recovery routine should not retry.</p> <p>3 - 7</p> <p>Reserved.</p> <p>Byte 4—Flag byte 2</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0</p> <p>The service parameter list has a position for a return code.</p> <p>1</p> <p>The service parameter list has a position for a reason code.</p> <p>2</p> <p>Reserved.</p> <p>3</p> <p>The caller has no exit data.</p> <p>4 and 5</p> <p>Reserved.</p> <p>6–7</p> <p>Reserved.</p>
20	4	Reserved.
24	4	Auxiliary SPB Pointer
28	4	EDC buffer pointer.
32	4	EDC buffer length.

Table 22. SPB Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
36	4	Address of XPB.
40	8	ID for latch manager.
48	4	Address for ERPB.
52	8	Original caller's register 1.
60	4	Address of CPRB request storage.
64	4	Length of CPRB request storage.
68	4	Address of CPRB reply storage.
72	4	Length of CPRB reply storage.
76	4	CCPS address.
80	4	Serialization block address.
84	4	Recovery token.
88	8	Recovery footprint for hash tables.
96	4	Reserved.
100	4	Pointer to metal C stack.
104	2	Entry point index of metal C caller.
106	2	Flags and indicators Byte 1 - Reserved. Byte 2 - Saved value of the caller's key.
108	4	ASCB of SPB owner.
112	4	Register 14 from CSFMIREC.
116	4	Reserved.
120	4	ENVR object address.
124	4	ENVR object length.
128	4	Reserved.
132	20	Reserved.
152	512	CTRACE buffer.

Parameters

Each service has a unique parameter list. Parameters 1–4 are always the return code, reason code, exit data length, and exit data. The other parameters differ with each service. The installation exit gets passed the address of the service parameter list in Register 1. For a description of each service's parameter list, refer to *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

Return Codes

To use a return code and reason code that are set in the postprocessing exit, you must set bit 2 in Offset +20 of EXPB. Setting bit 2 on causes ICSF to return the return code from the exit in register 15 and the reason code in register 0. Even though the application program receives the codes from the exit in the registers, the program still receives the codes from the service in the parameter list. The return code is the first parameter, and the reason code is the second parameter in the list.

Some control languages can access registers more easily than others. For this reason, ICSF allows you to return the return code and the reason code in both the registers and the parameter list. To do this, set bit 5 as well as bit 2 in Offset +20 of EXPB. The application then receives the return code and the reason code from the exit in both the registers and the parameter list.

If you do not set either of or both of the flag bits, the service ignores any return or reason code from the exit. The application program receives the codes from the service in both the registers and the parameter list.

The exit can pass back any valid return code for each service. For a listing of each service's return codes, see *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

CSF_SERVICE_EXIT - ICSF callable services exit

The ICSF callable services exit CSF_SERVICE_EXIT can be used to generate statistics for all ICSF callable services. This exit point occurs after the callable service finished processing, but before the service returns control to the application program. The intent of this exit is for statistics generation.

Controlling the exit routine through the dynamic exits facility

ICSF defined CSF_SERVICE_EXIT to the dynamic exits facility. You can refer to the exit by the name CSF_SERVICE_EXIT. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its exit routines.

If you do not associate any exit routines with CSF_SERVICE_EXIT in the PROGxx parmlib member, the system defaults to having no exit routine.

To limit the number of times the exit routine abnormally ends before it becomes inactive, use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro or the ABENDNUM and CONSEC parameters of the SETPROG EXIT operator command or of the EXIT statement of the PROGxx parmlib member. An ABEND is counted when both of the following conditions exist:

- The exit routine does not provide recovery or the exit routine does provide recovery, but percolates the error.
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system disables the exit routine after two consecutive ABENDs.

Exit routine environment

CSF_SERVICE_EXIT receives control in the following environment:

- Enabled for I/O and external interrupts.
- In supervisor state with PSW key 0.
- In AMODE(31) or AMODE(64), depending on the AMODE of the exit routine.
- In Primary ASC mode.
- In cross-memory mode with P = ICSF's address space and S = user's address space. Home might or might not equal secondary.
- With no locks held.
- Task or SRB mode.

If the callable service was started in AMODE(64) and an exit routine, which is AMODE(31), needs to access the user's parameters, the exit routine needs to switch to AMODE(64). Regardless of AMODE, the exit routine must not rely on the high 32 bits of any general register having a specific value on entry.

An exit routine can change characteristics (AMODE, ASC mode, locks-held state, cross-memory state, and so on) during its processing. However, the exit routine must return with the same characteristics as on entry. If you plan to access the user's parameters, you must write the exits in a language that can access ALET-qualified variable. This is because you are in AR mode and all the user's parameters, including the parameter list itself, are ALET-qualified. In particular, parameters that are passed into a service are in the user's address space, which you can access with an ALET of 1 (secondary).

Exit recovery

If an exit routine ABENDs and does not have recovery that retries, the system records the error to LOGREC and ICSF calls any exit routines that remain to be called. Whether the exit routine continues to be started depends on the ABEND processing of the dynamic exits facility.

Note: ICSF recommends, for system performance reasons, that the exit not establish recovery unless it modifies critical resources.

Entry specifications

ICSF passes the address of the IXIB (ICSF exit interface block) to exit CSF_SERVICE_EXIT.

The contents of the registers on entry to the exit are as follows.

Register

Contents

R0

N/A

R1

Address of IXIB - ICSF exit interface block.

R2 - R12

N/A

R13

Address of 144-byte save area.

R14

Return address.

R15

Entry point address of exit.

AR0

First 4 bytes of 8-byte PARAM area that is provided by the exit routine owner on CSVDYNEX ADD.

AR1

Second 4 bytes of 8-byte PARAM area that is provided by the exit routine owner on CSVDYNEX ADD.

Parameter list contents: Register 1 contains the address of the ICSF exit interface block (IXIB), which resides in the primary address space (ICSF's address space). The IXIB is mapped by macro CSFZIXIB and the layout is shown in [Table 23 on page 187](#).

Table 23. IXIB control block format		
Offset (Dec)	Number of bytes	Description
0	4	Parmlist with a single entry that points to the IXIB.
4	4	EBCDIC ID.
8	2	Version number of this IXIB.
10	1	Flags X'80' Bit = 1 Caller is AMODE(31). X'40' Bit = 1 Caller is AMODE(64).
11	1	PSW key is in the first 4 bits.
12	4	Address of 2048-byte work area.
16	2	IBM assigned service number.

Table 23. IXIB control block format (continued)

Offset (Dec)	Number of bytes	Description
18	2	Installation service number.
20	4	Reserved.
24	8	Service name.
32	8	Original caller's R1.
40	4	Return code from service.
44	4	Reason code from service.
48	16	STCKE value before service called.
64	16	STCKE value after service called.
80	32	Reserved.

Note on original caller's R1

Each ICSF callable service has a unique parameter list. For a description of each service's parameter list, see [z/OS Cryptographic Services ICSF Application Programmer's Guide](#).

Note on exit behavior

Exit routines that are registered to CSF_SERVICE_EXIT must not change any parameters, including the IXIB or anything it points to.

Cryptographic key data set entry retrieval installation exit

The cryptographic key data set entry retrieval installation exit (CSFCKDS) is called when a service requests an entry from the in-storage cryptographic key data set (CKDS) by label. ICSF calls this exit after it finds the record in the CKDS and before it returns the record to the service.

Note: This exit is given control only for a fixed-length record CKDS. The exit does not work with the variable-length record format of the CKDS.

Purpose and use of the exit

The exit point lists the entry that matches a certain label and type. You can use the exit to check fields in a record and decide whether to use the record. The exit sets a return code that specifies whether to use the record or not. Use the *exit_data* parameter in the service to specify what the exit should use as a search value.

For example, you can use the CKDS entry retrieval exit to perform a specific search of the installation data field. An installation can specify whatever it chooses to in the installation data field. The exit can select a record that matches a certain key label and key type. You can check the record and accept or reject it based on the installation data field.

Note: The cryptographic key data set entry retrieval installation exit will not be given control if SYSPLEXCKDS(YES,FAIL(xxx)) is specified in the ICSF installation options data set.

Environment of the exit

The exit receives control with these characteristics:

- Supervisor state
- Key 0
- APF-authorized

- TCB or SRB mode
- AR mode
- AMODE(31)
- RMODE(ANY)
- Cross memory mode

The exit can change the characteristics during its processing. However, the exit must return to its caller with the same characteristics as on entry.

The exit runs in the cross memory mode in the ICSF address space. The CKDS records are ALET-qualified. ICSF supplies the address and the ALET of a CKDS record as parameters to the CKDS retrieval exit.

For information about cross memory mode and AR mode, see [z/OS MVS Programming: Extended Addressability Guide](#).

Installing the exit

Install the CKDS entry retrieval exit by installing the load module that contains the exit into an APF-authorized library. ICSF uses this normal z/OS search order to locate the exit:

- Job pack area
- Steplib (if one exists)
- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

Define the ICSF name and the load module name on the EXIT keyword in the installation options data set. “Parameters in the installation options data set” on page 35 describes the installation options data set in further detail. The EXIT keyword has this syntax:

EXIT (ICSF name, load module name, FAIL (options))

The **ICSF name** portion of the keyword refers to the ICSF name for the exit. The ICSF name for the CKDS entry retrieval exit is CSFCKDS. The **load module name** is the name of the load module that contains the exit. The name can be any valid name that your installation chooses. The **FAIL** portion of the EXIT keyword specifies the action ICSF takes if the exit cannot be loaded or if it ends abnormally. The valid FAIL options are:

NONE

Do not take any action.

EXIT

Do not call this exit again. The exit will not receive control during subsequent attempts at CKDS retrieval.

SERVICE

Fail the service. All subsequent attempts at CKDS entry retrieval fail.

ICSF

End ICSF.

You must specify a FAIL option. If you do not, ICSF returns an error message, ends abnormally, and generates an SVC dump when attempting to load the exit. If the exit ends abnormally, the attempt at CKDS entry retrieval fails, regardless of the FAIL option you specified. FAIL options only apply to subsequent attempts at CKDS entry retrieval.

Input

The CKDS entry retrieval exit receives the address of an address list passed in register 1. Each address in the list points to a parameter. The address list exists in the ICSF address space, and register 1 is not ALET-qualified.

[Table 24 on page 190](#) describes the parameters for the CKDS entry retrieval exit.

Table 24. The CKDS Entry Retrieval Exit Parameters

Parameter	Description
1	The address of the current CKDS record. See “Cryptographic Key Data Set (CKDS) formats” on page 243 for a description of the CKDS record format.
2	The address of the ALET of the current CKDS record. This record is a fullword address.
3	The address of the record that matches a certain label and type. This value is a fullword integer. The parameter is in the ICSF address space and the exit can access the parameter using an ALET of 0.
4	The address of the record chosen. This value is a fullword integer. The parameter is in the ICSF address space and the exit can access the parameter using an ALET of 0.
5	The address of the exit data length. This value is a fullword integer. The parameter is in the caller's address space, which is the secondary address space, and the exit can access the parameter using an ALET of 1.
6	The address of the exit data. For a description of exit data, see z/OS Cryptographic Services ICSF Application Programmer's Guide . The parameter is in the caller's address space, which is the secondary address space, and the exit can access the parameter using an ALET of 1.
7	The address of the secondary parameter block. See “Secondary parameter block” on page 183 for a description of the secondary parameter block. The parameter is in the ICSF address space and the exit can access the parameter using an ALET of 0.

Return codes

You can pass a return code back in register 15.

The valid decimal return codes are:

Return Code	Description
-------------	-------------

0

Use the record.

4

Do not use the record.

If you specify not to use any of the records that match the search value, ICSF returns control to the application. It returns with return code 12 and reason code 10024, which indicate that the exit rejected all the keys in the search.

PCF conversion program installation exit

Use the PCF conversion program to convert a CKDS from the Programmed Cryptographic Facility (PCF) format to the ICSF format. The conversion program converts each record in the PCF CKDS to the CKDS format that ICSF uses, and then writes the new record to an ICSF CKDS. The conversion program extends the label field to 64 bytes.

An ICSF CKDS record contains an installation data field that you can use to further identify the record. This field can contain any information about a record that your installation would like to use. You can use the conversion program exit to change the information in this field. You can also use the conversion program exit to have the conversion program not place a converted CKDS entry in the ICSF CKDS.

[Chapter 8, “Migration from PCF to z/OS ICSF,”](#) on page 227 contains more information about the PCF conversion program.

Purpose and use of the exit

The PCF conversion program installation exit (CSFCONVX) is called at three points during processing of the conversion program:

- **During conversion program initialization.** This is known as the conversion preprocessing invocation. At this point, you can use the exit to change the ICSF CKDS header record installation data field.
- **During conversion program individual record processing.** This is known as the record processing invocation. At this point, the conversion program is converting the PCF entry but has not yet placed the entry into the ICSF CKDS. You can use the exit to change the installation data field in the entry for the ICSF CKDS. You can also specify that the conversion program not place the entry into the ICSF CKDS.
- **Just prior to conversion program termination.** This is known as the conversion postprocessing invocation. At this point, like the preprocessing exit point, you can use the exit to change the ICSF CKDS header record installation data field.

Environment of the exit

The exit receives control with these characteristics:

- Problem program state.
- APF-authorized
- TCB mode
- Address Space Control mode=primary
- AMODE(31)
- RMODE(ANY)

The exit can change the characteristics during its processing. However, the exit must return to its caller with the same characteristics as on entry.

The exit runs in task mode in the caller's own address space.

Installing the exit

Install the load module that contains the exit into an APF-authorized library. ICSF uses this normal z/OS search order to locate the exit:

- Job pack area
- Steplib (if one exists)
- Joblib (if one exists)
- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

Define the ICSF name and load module name on the EXIT keyword in the installation options data set. For more information about the installation options data set, see [“Parameters in the installation options data set”](#) on page 35. The EXIT keyword has this syntax:

EXIT (ICSF name, load module name, FAIL (options))

The **ICSF name** portion of the keyword refers to the ICSF name for the exit. The ICSF name for the conversion program exit is CSFCONVX. The **load module name** is the name of the load module that contains the exit. This name can be any valid name that your installation chooses. The **FAIL** portion of the EXIT keyword specifies the action ICSF takes if the exit cannot be loaded. The valid FAIL options are **NONE**, **EXIT**, **SERVICE**, and **CSF**. For the conversion program exit, you can use these options only:

NONE

Initialization continues even if exit cannot be loaded.

ICSF

Initialization ends if exit cannot be loaded.

You must specify a FAIL option. If you do not, ICSF returns an error message, ends abnormally, and generates an SVC dump when attempting to load the exit.

If the exit ends abnormally, the conversion program does also.

Input

ICSF supplies the address of the conversion program exit parameter block (CVXP) in register 2 each time it calls the PCF conversion program exit. The exit does not receive a parameter list. [“Entry and return specifications”](#) on [page 167](#) gives a complete list of the registers on entry to the conversion program exit.

[Table 25](#) on [page 192](#) describes the contents of the exit control block.

Table 25. CVXP Control Block Format		
Offset (Dec)	Number of Bytes	Description
0	4	Name. The name of the control block. The field contains the character string CVXP.
4	2	Version. The version of the control block. The field contains the character string 01.
6	2	Length. The length of the control block. The value is 28 in decimal.
8	4	Return Code. The value the exit returns. Valid decimal values for this field are: Return Code Description 0 Normal. 4 Do not process the entry. 8 End conversion program.
12	4	Address of the ICSF CKDS installation data area.
16	4	The value in decimal of the length of the ICSF CKDS installation data area.
20	1	Action. Bit 0 is set on if the action was to change an entry on the ICSF CKDS. Bit 0 is set off if the action was to add an entry to the ICSF CKDS. The rest of the bits in this byte are reserved.

Table 25. CVXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
21	1	<p>Call Point.</p> <p>Indicates the invocation point of the exit. The exit cannot change this field and the conversion program does not use this field on return from the exit. You can determine the invocation point by the bit that is set on.</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0 Conversion preprocessing invocation.</p> <p>1 Conversion postprocessing invocation.</p> <p>2 Record processing invocation.</p> <p>3-7 Reserved.</p>
22	6	Reserved.

Return codes

You can pass a return code back to the conversion program in the CVXP control block (offset +8). The exit can use return codes to reject records for conversion processing or end the conversion program.

Return Code Description

- 0**
Normal.
- 4**
Do not process the entry.
- 8**
End conversion program.

Single-record, Read-write installation exit

ICSF provides an exit that is called when a record is read from or written to a CKDS. ICSF calls the single-record, read-write (CSFSRRW) exit under these conditions:

- The PCF conversion program converts a record into ICSF CKDS format. The conversion program calls the exit before it writes a converted record to the ICSF CKDS.
- ICSF reenciphers a disk copy of a CKDS under a new master key. ICSF calls the exit two times during this processing; after ICSF reads a record to reencipher it and before ICSF writes the reenciphered record.
- ICSF refreshes the in-storage copy of a CKDS. ICSF calls this exit after reading a record from the disk copy to place into storage.

Using the exit, you can do such things as prevent the record from being processed, or add user information to the record.

Note: This exit is given control only for a fixed-length record CKDS. The exit does not work with the variable-length record format of the CKDS.

Purpose and use of the exit

The exit receives a parameter block that describes the CKDS record and the action occurring to the record. By setting a return code in the parameter block, the exit may affect the processing of the record. Depending on the return code, one of these actions occurs:

- ICSF continues to read the record.
- ICSF does not read or write the record.
- ICSF does not read or write the entire CKDS.

The parameter block contains the address of the CKDS record. The exit can add information into the installation data field of the record. For integrity reasons, ICSF receives only changes to this particular field. If the exit sets a return code to continue processing, ICSF processes the record with this information.

The KGUP exit, the PCF conversion program exit, and the single-record, read-write exit can add information to the installation data field of the CKDS header record to identify the data set. If the header record installation data field contains information identifying the CKDS, the single-record, read-write exit can check the field to ensure that it is processing the correct data set. If the exit finds that it is processing the wrong CKDS, the exit can set a return code to stop the processing of the entire data set.

You can use the exit to prevent processing of a record. You can check certain fields in the record and specify that the record not be processed. For example, during postprocessing conversion, you can prevent the processing of any record of a certain key type. However, the exit should never prevent processing of a record containing a system key because ICSF uses these keys in its processing. You differentiate a system key record from other key records by its key label. A system key record label contains all binary zeros. All other key labels contain an alphabetic first character with the remaining characters as either alphabetic or numeric.

Environment of the exit

The exit receives control with these characteristics:

- Problem program state
- APF-authorized
- TCB mode
- Address Space Control mode=primary
- AMODE(31)
- RMODE(ANY)

The exit can change the characteristics during its processing. However, the exit must return to ICSF with the same characteristics as on entry.

When the single-record, read-write exit is called, the exit parameter block is in the caller's address space. The exit is loaded in the caller's address space. The caller is either the PCF conversion program, the utility program (CSFEUTIL), or an ICSF panel.

Installing the exit

Install the load module that contains the exit into an APF-authorized library. ICSF uses this search order to locate the exit:

- Job pack area
- Steplib (if one exists)
- Joblib (if one exists)
- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

Define the ICSF name and load module name on the EXIT keyword of the installation options data set. For more information about the installation options data set, see [“Parameters in the installation options data set”](#) on page 35. The EXIT keyword has this syntax:

EXIT (ICSF name, load module name, FAIL (options))

The **ICSF name** portion of the keyword refers to the ICSF name for the exit. The ICSF name for the single-record, read-write exit is CSFSRRW. The **load module name** is the name of the load module that contains the exit. The name can be any valid name that your installation chooses. The **FAIL** portion of the EXIT keyword specifies the action ICSF takes if the exit cannot be loaded or ends abnormally. The valid FAIL options are:

NONE

Do not take any action.

EXIT

Do not call this exit again.

SERVICE

Fail the service that called the exit.

ICSF

Fail the service that called the exit.

You must specify a FAIL option. If you do not, ICSF returns an error message, ends abnormally, and generates an SVC dump when attempting to load the exit. If you specify FAIL(ICSF) and the exit cannot be loaded, ICSF initialization does not continue. If you specify FAIL(ICSF) and the exit ends abnormally, ICSF issues an advisory message that ICSF should be ended.

Input

The single-record, read-write exit receives the address of the address list passed in register 1. The first address in the address list is for the read-write exit parameter block (RWXP). The exit does not receive a parameter list. [“Entry and return specifications”](#) on page 167 gives a complete list of the registers on entry to the single-record, read-write exit.

The RWXP parameter block contains the address of the CKDS record that is being processed and information about the situation in which the exit is called. The exit sets a return code in a field in the block to specify whether the processing should continue. [Table 26 on page 195](#) describes the RWXP control block.

Table 26. RWXP Control Block Format		
Offset (Dec)	Number of Bytes	Description
0	4	Name. The name of the control block. The field contains the character string RWXP.
4	2	Version. The version of the control block. The field contains the character string 01.
6	2	Length. The length of the control block. The value of this field is 32 in decimal.

Table 26. RWXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
8	4	<p>Return Code.</p> <p>The value the exit returns. Valid decimal values for this field are:</p> <p>Return Code Description</p> <p>0 Process current CKDS record</p> <p>4 Do not process current CKDS record</p> <p>8 End processing</p>
12	4	Address of the CKDS record.
16	4	The value in decimal of the length of the CKDS record.
20	7	<p>Action.</p> <p>The field is a 7-byte character string describing the action performed on the CKDS record. The field can contain these values:</p> <ul style="list-style-type: none"> • READ • WRITE • DELETE • REWRITE <p>Note that the value of the field is left-justified and padded with blanks.</p>
27	1	<p>Exit Invocation Reason</p> <p>The reason that the exit was invoked. The field relates to only the CKDS and can contain one of these values:</p> <p>2 Refresh of the in-storage CKDS with a disk copy of a CKDS. The value of the Action field is READ.</p> <p>3 Reencipher of the in-storage CKDS from a disk copy of a CKDS. The value of the Action field is READ or WRITE.</p> <p>5 Conversion record postprocessing. The value of the Action field is WRITE.</p> <p>8 Key entry hardware input. The value of the Action field is READ or WRITE.</p>
28	4	Data set type.

Return codes

You can pass a return code back to the single-record, read-write process in the RWXP control block (offset +8). The exit can use the return code to reject records or to end the single record read-write process. These values are valid decimal return codes:

Return Code	Description
--------------------	--------------------

0	Process the current CKDS record.
4	Do not process the current CKDS record.
8	End processing.

Exit points for security installation exits

IBM-supplied security exit routines were removed in ICSF/MVS Version 2 Release 1. The exit points themselves are still available.

Security installation exits

ICSF provides these exit points to control access to the keys in the in-storage CKDS and to the services.

- Security Initialization Exit
- Security Termination Exit
- Security Service Exit
- Security Key Exit

Purpose and use of the exits

There are two groups of security exits. The security initialization exit (CSFESECI) and security termination exit (CSFESECT) are called during ICSF mainline processing to maintain a security communication area that is used by the other security exits.

Next is a description of each point where ICSF calls security exit routines.

Security initialization exit

ICSF calls this exit during initialization just before calling the ICSF mainline exit CSFEXIT. You can use this exit to anchor resource lists, work areas, and other data to the security communication area. The security service exit (CSFESECS) and security key exit (CSFESECK) can be used to control access to resources on ICSF and for logging in SMF the results of any authorization checks that are made. The security initialization exit defined in the options data set is only invoked if CSFESECS, CSFESECK, or both are also defined.

Security termination exit

ICSF calls this exit as the last function when ICSF ends, before deleting all the installation exits. You can use this exit to free whatever is anchored to the security communication area.

Security service exit

ICSF calls this exit when an application uses an IBM-supplied service, before calling any other installation exit that is associated with that service. You can use this exit to control access to a service. See [Appendix G, “Resource names for CCA and ICSF entry points,” on page 503](#) for a list of services.

Security key exit

ICSF calls this exit when an application uses a key in the in-storage CKDS, before any other installation exit associated with that use of the key is called. You can use this exit to control access to the keys in the CKDS.

Environment of the exits

The security initialization and termination exits receive control with these characteristics:

- Supervisor state
- Key 0
- APF-authorized
- TCB mode
- Address Space Control mode=access register mode
- AMODE(31)
- RMODE(ANY)

The exits can change the characteristics during their processing. However, the exits must return to ICSF with the same characteristics as on entry.

The security service and key exits receive control with these characteristics:

- Supervisor state
- Key 0
- APF-authorized
- TCB mode
- Cross memory mode
- AR mode
- AMODE(31)
- RMODE(ANY)

The exits can change the characteristics during their processing. However, the exits must return to ICSF with the same characteristics as on entry.

Note: The security exits are not called in SRB mode.

Installing the exits

You install the security exits by installing the load module that contains the exit into an APF authorized library. ICSF uses this normal search order to locate the exit:

- Job pack area
- Steplib (if one exists)
- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

Use the EXIT keyword in the installation options data set to define the ICSF name and load module name. For information about the installation options data set, see [“Parameters in the installation options data set”](#) on page 35. The EXIT keyword has this syntax:

EXIT (*ICSF name*, *load module name*, *FAIL (options)*)

The **ICSF name** portion of the keyword refers to the ICSF identifier for each exit, CSFESECI, CSFESECT, CSFESECS, and CSFESECK. The **load module name** is the name of the load module that contains the exit. The name can be any valid name your installation chooses. The action that the **FAIL** portion of the EXIT keyword specifies depends on the type of security exit.

For the security initialization and termination exits, the FAIL portion specifies the action ICSF takes if the exit cannot be loaded. The valid FAIL options mean:

NONE

Continue initialization even if exits cannot be loaded.

SERVICE

Continue initialization even if exits cannot be loaded.

EXIT

Continue initialization even if exits cannot be loaded.

ICSF

End ICSF if exits cannot be loaded.

You must specify a FAIL option. If you do not, ICSF returns an error message, ends abnormally, and generates an SVC dump when attempting to load the exit.

If the security initialization exit ends abnormally, ICSF ends. If the security termination exit ends abnormally, ICSF continues to end.

For the security service and key exits, the FAIL portion specifies the action ICSF takes if the exit cannot be loaded or ends abnormally. When the service or key exit is loaded, the valid FAIL options mean:

NONE

Continue initialization even if exits cannot be loaded.

SERVICE

Continue initialization even if exits cannot be loaded.

EXIT

Continue initialization even if exits cannot be loaded.

ICSF

End ICSF if exits cannot be loaded.

You must specify a FAIL option. If you do not, ICSF returns an error message, ends abnormally, and generates an SVC dump when attempting to load the exit.

When the security service exit ends abnormally, the valid FAIL options mean:

NONE

Process subsequent calls to the service as if no abnormal ending occurred. Call the exit for each call of a service.

SERVICE

Fail on subsequent calls to the particular service.

EXIT

Do not call the exit again. Bypass the exit on subsequent calls to any IBM service.

ICSF

End ICSF.

If the security service exit ends abnormally, ICSF ends the service call before performing the service.

When the security key exit ends abnormally, subsequent attempts to access the in-storage CKDS are processed as if no abnormal ending occurred. The exit continues to be called for each access attempt regardless of the FAIL option.

If the security key exit ends abnormally, ICSF ends the attempt to access the CKDS before performing the access.

Input

The security initialization and termination exits receive the address of an 8-byte security communication area in register 1. When ICSF starts, the security initialization exit can use this area as an anchor for resource lists, work areas, or any other data that your service or keys security exits need to check authorizations. When ICSF ends, the security termination exit can free any system resources that are anchored to this area and used by the service or keys security exits. For example, the exit can free storage that is allocated from the common storage area (CSA).

When a call to a service occurs, the security service exit receives the address of an address list passed in register 1. [Table 27 on page 200](#) describes the parameters the exit receives:

Table 27. Parameters received by the Security Service Exit

Parameter	Number of Bytes	Description
1	8	The security communication area.
2	8	The character string CSFSERV.
3	8	The name of the service being called.

When an attempt to access a CKDS entry occurs, the security key exit receives the address of an address list passed in register 1. [Table 28 on page 200](#) describes the parameters this exit receives:

Table 28. Parameters received by the Security Key Exit

Parameter	Number of Bytes	Description
1	8	The security communication area.
2	8	The character string representing the SAF class being checked. May be CSFKEYS or XCSFKEY.
3	64	The label of the key entry being accessed.

Register 0 contains the address of the exit parameter block (EXPB). See [Figure 4 on page 180](#) and [Table 21 on page 181](#).

Return codes

All the security exits can pass back a return code in register 15. The security initialization exit supports these decimal return codes:

Return Code Description

0
Proceed with initialization.

16
End ICSF.

Any return codes other than those listed cause ICSF to end abnormally.

The security termination exit supports these decimal return codes:

Return Code Description

0 or 16
Proceed with termination.

Any return codes other than those listed cause ICSF to end abnormally.

The security service exit supports these decimal return codes:

Return Code Description

0 or 4
Proceed with the service call.

Any return codes other than those that are listed cause the service call to fail.

The security key exit supports these decimal return codes:

Return Code Description

0

Proceed with the access of the CKDS entry.

4

If the second input parameter is CSFKEYS, proceed with the access of the CKDS entry. Otherwise, the access is failed.

Any return codes other than those that are listed cause the access of the key to fail.

Key generator utility program installation exit

The key generator utility program (KGUP) generates and maintains keys in the cryptographic key data set (CKDS). You can use KGUP to generate or supply a key to update the CKDS. KGUP generates keys to use in key exchange with other systems. ICSF provides an exit for customizing KGUP processing. For information about using KGUP to managing cryptographic keys, see [*z/OS Cryptographic Services ICSF Administrator's Guide*](#).

Purpose and use of the exit

You can use the KGUP installation exit (CSFKGUP) to modify records in the CKDS, write copies of records to alternate data sets, or put additional information in the SMF record. There are many other uses for the KGUP exit depending on your installation's needs. Examine the calling points for an exit and the active control block fields at each calling point to determine other applications for the exit.

KGUP calling points

After an ICSF administrator submits a KGUP job for processing, KGUP calls exits at four points in processing:

1. **During KGUP initialization.** This is known as the KGUP preprocessing exit. After the KGUP job begins but before KGUP starts processing a control statement, KGUP calls this exit.

You can use this exit to place additional information in the installation data field of the CKDS header record. You may want to do this if you need to process different cryptographic key data sets differently. You can place information in the installation data field of the record, and then subsequent calls of the exit can use this information as the basis for performing processes.

2. **Before KGUP processes a key that is identified by a control statement.** This is known as the record preprocessing exit. Before KGUP accesses the CKDS to retrieve the key that is requested in the control statement, KGUP calls the exit again.

Note: This call occurs before KGUP accesses the CKDS. If an exit routine alters a key entry at this call, KGUP accesses the CKDS with the altered entry.

You can use this exit to provide additional security for entering clear key values. When a user enters a clear key in a control statement, use the exit to change the value. In this way, the user never knows the actual clear value in the CKDS. For example, a user enters zeros for clear key values. Your exit generates some random number and replaces the user's clear key value. KGUP then processes the exit's random number as the value to write to the CKDS.

3. **Before KGUP updates the CKDS with a key entry.** This is known as the record postprocessing exit. After KGUP processes a key and before KGUP updates the CKDS, KGUP calls the exit a third time.

At this call, the installation exit can change any information in the Key Output Data Set. Changing the Key Output Data Set also enters the changed keys into the Control Statement Output Data Set, if the keys are exportable. You can use this exit to create audit trails.

KGUP will not call the exit for this calling point when the CKDS is in KDSR format.

4. **During KGUP termination.** This is known as the KGUP postprocessing exit. Calls to this exit occur after KGUP completes processing but before KGUP returns control to ICSF.

Note: If an error occurs in exit processing, KGUP does not call the remaining exit invocations. If an error occurs in KGUP processing that does not result in an abnormal ending, KGUP does not call the remaining exit invocations.

Processing in the exit

At each call, the exit receives the address of the KGUP exit parameter block (KGXP) in register 1. The exit can access any of the data in KGXP. The exit can alter some of the fields in KGXP, while others are simply references. Also, the KGUP exit can alter some fields at some calls but not at other calls.

A field in KGXP gives the calling point of the exit. The exit uses this field to determine when to call the exit to perform appropriate processing. [“Input” on page 203](#) gives a more detailed explanation of the KGXP control block, the values it contains, and when an exit can use or change the values.

Environment of the exit

The KGUP calls the exit only in the address space where KGUP is running. The exit receives control with these characteristics:

- Supervisor state
- APF-authorized
- TCB mode
- Address Space Control mode=primary
- AMODE(31)
- RMODE(ANY)

The exit can change the characteristics during its processing. However, the exit must return to its caller with the same characteristics as on entry.

Installing the exit

Install the load module that contains the exit into an APF authorized library. ICSF uses this search order to locate the exit:

- Job pack area
- Steplib (if one exists)
- Joblib (if one exists)
- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

Define the ICSF name and load module name on the EXIT keyword in the installation options data set.

Note: The load module name must not be named CSFKGUP

For more information about the installation options data set, see [“Parameters in the installation options data set” on page 35](#). The EXIT keyword has this syntax:

EXIT (ICSF name, load module name, FAIL (options))

The **ICSF name** portion of the keyword refers to the ICSF name for the KGUP exit. The ICSF name for the KGUP exit is CSFKGUP. The **load module name** is the name of the load module that contains the exit. The name can be any valid name that your installation chooses. The **FAIL** portion of the EXIT keyword specifies the action ICSF takes if the exit cannot be loaded. The valid FAIL options are **NONE**, **EXIT**, **SERVICE**, and ICSF. The FAIL options available to the KGUP exit are:

NONE

Initialization continues even if exit cannot be loaded.

ICSF

Initialization ends if exit cannot be loaded.

You must specify a FAIL option. If you do not, ICSF returns an error message, ends abnormally, and generates an SVC dump when attempting to load the exit. If the exit ends abnormally, KGUP also ends abnormally.

Input

At each of the invocation points, the exit receives the address of the KGUP exit parameter block (KGXP) in register 1. The exit does not receive a parameter list. [“Entry and return specifications” on page 167](#) gives a complete list of the registers on entry to the KGUP exit.

The KGUP exit can alter some of the fields in KGXP. Some fields only provide information to the exit and cannot be changed, and some fields do not apply to particular calls to the exit.

[Table 29 on page 203](#) describes the KGXP control block.

Table 29. KGXP Control Block Format		
Offset (Dec)	Number of Bytes	Description
0	4	Block Identifier. The name of the control block. The field must contain the character string KGXP. The exit must not change the value and KGUP does not use the field upon return from the exit.
4	2	Block Version Number. The version of the control block. The field must contain the character string 03. The exit cannot change this field and KGUP does not use this field on return from the exit.
6	2	Block Length. The length of the control block. The decimal value of the field is 408. The exit cannot change the field and KGUP does not use this field on return from the exit.
8	4	Return Code. The return code the exit supplies upon completion. Upon entry, KGUP initializes this field to zeros. The valid decimal return codes for each of the invocation points are: Record Pre- or postprocessing. 0 Normal, continue processing. 4 Reject control statement, but do not end KGUP. 8 End KGUP immediately. KGUP pre- or postprocessing. 0 Normal, continue processing. > 0 End KGUP immediately.

Table 29. KGXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
12	1	<p>Call Point.</p> <p>Indicates the invocation point of the exit. The exit cannot change this field and KGUP does not use this field on return from the exit. You can determine the invocation point by the bit that is set on.</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0 KGUP preprocessing invocation.</p> <p>1 KGUP postprocessing invocation.</p> <p>2 Record preprocessing invocation.</p> <p>3 Record postprocessing invocation.</p> <p>4-7 Reserved.</p>
13	1	<p>Options.</p> <p>Indicates the keywords specified on the KGUP control statement. The exit cannot change this field and KGUP does not use the field upon return from the exit. The field is used only during the record preprocessing and postprocessing invocations. You can determine the keywords on the control statement by the bits that are set on.</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0 LABEL with multiple values specified.</p> <p>1 RANGE specified.</p> <p>2 KEY specified.</p> <p>3 CLEAR specified.</p> <p>4 SINGLE specified.</p> <p>5 NOCV specified.</p> <p>6 OUTTYPE specified.</p> <p>7 DOUBLEO specified.</p>

Table 29. KGXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
14	1	<p>Verb Type.</p> <p>Indicates the verb used on the KGUP control statement. The exit cannot change this field and KGUP does not use this field on return from the exit. The field is used only for the record preprocessing and record postprocessing invocations. You can determine the verb on the control statement by the bit that is set on.</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0 ADD</p> <p>1 UPDATE</p> <p>2 DELETE</p> <p>3 RENAME</p> <p>4 SET</p> <p>5 OPKYLOAD</p> <p>6–7 Reserved.</p>
15	1	<p>KGUP Flags.</p> <p>Indicates the processing conditions encountered by KGUP at the record postprocessing invocation. The exit cannot change this field and KGUP does not use the field upon return from the exit. The field is not used for the KGUP pre- or postprocessing invocations or the record preprocessing invocation. The processing conditions can be determined by examining whether bit 0 is set on.</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0 Non-odd parity key was imported.</p> <p>1 Algorithm is AES.</p> <p>2 Algorithm is DES.</p> <p>3 \$TRIPLEO keyword specified.</p> <p>4–7 Reserved.</p>

Table 29. KGXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
16	72	<p>Action Key.</p> <p>Contains the key index accessed by the KGUP control statement. The key index consists of the key label and type fields of a CKDS record entry (“Debugging aids” on page 156 describes the CKDS record format in greater detail). The key index is the first 72 bytes of a CKDS record, and the information in the key index is used to differentiate one key from another.</p> <p>The exit can modify the field at the record preprocessing invocation. The field is not used for the KGUP pre- or postprocessing invocation or the record postprocessing invocation.</p> <p>If the exit modifies the field, KGUP uses the modified field to access the CKDS upon return from the exit.</p> <p>Before the record preprocessing invocation, KGUP places this in this field:</p> <ul style="list-style-type: none"> • The key label or key old label from the LABEL or key label from the RANGE keyword of the control statement • The key type from the TYPE keyword of the control statement <p>The exit cannot modify the key label, key old label, or key type.</p>
88	72	<p>Rename Key.</p> <p>Contains the key index used to rename a key when RENAME is the verb on the control statement. The key index consists of the key label and type fields of a CKDS record entry.</p> <p>The exit can modify the field at the record preprocessing invocation. The field is not used for the KGUP pre- or postprocessing or record postprocessing invocations.</p> <p>If the exit modifies the field, KGUP uses the modified field to access the CKDS upon return from the exit.</p> <p>Before the record preprocessing invocation, KGUP places this information in this field:</p> <ul style="list-style-type: none"> • The key new label from the LABEL keyword of the control statement. • The key type from the TYPE keyword of the control statement. <p>The exit cannot modify the key new label or the key type.</p>

Table 29. KGXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
160	72	<p>Transkey key-label1.</p> <p>The key index of the TRANSKEY key-label1 on the KGUP control statement. The key index is the key label and type of the CKDS record entry.</p> <p>The exit can modify the field at the record preprocessing invocation. The field is not used for the KGUP pre- or postprocessing and record postprocessing invocations.</p> <p>If the exit modifies the field, KGUP uses the modified field to access the CKDS upon return from the exit.</p> <p>Before the record preprocessing invocation, KGUP places this information in this field:</p> <ul style="list-style-type: none"> • The key-label1 from the TRANSKEY keyword of the control statement. • The key type. The type is IMPORTER, if keys are supplied; the type is EXPORTER, if keys are not supplied. <p>The exit cannot modify the key-label1 or the key type.</p>
232	72	<p>Transkey key-label2.</p> <p>The key index of the TRANSKEY key-label2 on the KGUP control statement. The key index is the key label and type of the CKDS record entry.</p> <p>The exit can modify the field at the record preprocessing invocation. The field is not used for the KGUP pre- or postprocessing and record postprocessing invocations.</p> <p>If the exit modifies the field, KGUP uses the modified field to access the CKDS upon return from the exit.</p> <p>Before the record preprocessing invocation, KGUP places this information in this field:</p> <ul style="list-style-type: none"> • The key-label2 from the TRANSKEY keyword of the control statement. • The key type. The key type is IMPORTER, if keys are supplied; the type is EXPORTER, if keys are not supplied. <p>The exit cannot modify the key-label2 or the key type.</p>
304	8	<p>The OUTTYPE value, if specified. If no OUTTYPE is specified, this field set to binary zeros.</p>
312	4	<p>Key length in bytes.</p> <p>The value supplied by the LENGTH keyword or the byte length of the key value if the KEY option was selected.</p> <p>This value is for ease of processing the key values. The exit may not modify this value.</p>

Table 29. KGXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
316	16	<p>Key key-value 1.</p> <p>The value of the key supplied on the KGUP control statement. The 16 bytes are hexadecimal characters representing the 8-byte hexadecimal key value. The field contains a value only if the KEY option was specified and a key value was supplied on the control statement. You can determine whether the KEY option was used by examining bit 2 at offset +13 in KGXP.</p> <p>If TRANSKEY was specified on the control statement, KGUP decrypts key-value1 under the transport key specified with the TRANSKEY keyword. If CLEAR was specified on the control statement, KGUP does not decrypt key-value1.</p> <p>The exit can modify the field at the record preprocessing invocation. This field is not used for the KGUP pre- or postprocessing invocations or the record postprocessing invocation. The field does not contain a value when generating keys.</p> <p>The exit is permitted to put values in this field only if a key was supplied on the control statement. The exit-supplied value must be edited for hexadecimal values and it then replaces the values entered on the input control statement.</p>
332	16	<p>Key key-value 2.</p> <p>The value of the second key supplied on the KGUP control statement. The 16 bytes are hexadecimal characters representing the 8-byte hexadecimal key value. The field contains a value only if the KEY option was specified and a key value was supplied on the control statement. You can determine whether the KEY option was used by examining bit 2 at offset +13 in KGXP.</p> <p>If TRANSKEY was specified on the control statement, KGUP decrypts the key-value 2 under the transport key specified with the TRANSKEY keyword. If SINGLE was specified on the control statement, the key-value 2 will be equal to the key-value. If CLEAR was specified on the control statement, KGUP does not decrypt the key-value 2.</p> <p>The exit can modify the field at the record preprocessing invocation. This field is not used at the KGUP pre- or postprocessing invocation or the record postprocessing invocation.</p> <p>The field does not contain a value when generating keys.</p> <p>The exit can put values in this field only if a key was supplied on the control statement. The exit-supplied value must be edited for hexadecimal values; it then replaces the values entered on the input control statement.</p>

Table 29. KGXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
348	16	<p>Key key-value 3.</p> <p>The value of the third key supplied on the KGUP control statement. The 16 bytes are hexadecimal characters representing the 8-byte hexadecimal key value. The field contains a value only if the KEY option was specified and a key value was supplied on the control statement. You can determine whether the KEY option was used by examining bit 2 at offset +13 in KGXP.</p> <p>If TRANSKEY was specified on the control statement, KGUP decrypts the key-value 3 under the transport key specified with the TRANSKEY keyword. If CLEAR was specified on the control statement, KGUP does not decrypt the key-value 3.</p> <p>The exit can modify the field at the record preprocessing invocation. This field is not used at the KGUP pre- or postprocessing invocation or the record postprocessing invocation.</p> <p>The field does not contain a value when generating keys.</p> <p>The exit can put values in this field only if a key was supplied on the control statement. The exit-supplied value must be edited for hexadecimal values; it then replaces the values entered on the input control statement.</p>
364	16	<p>Key key-value 4.</p> <p>The value of the fourth key supplied on the KGUP control statement. The 16 bytes are hexadecimal characters representing the 8-byte hexadecimal key value. The field contains a value only if the KEY option was specified and a key value was supplied on the control statement. You can determine whether the KEY option was used by examining bit 2 at offset +13 in KGXP.</p> <p>The exit can modify the field at the record preprocessing invocation. This field is not used at the KGUP pre- or post-processing invocation or the record post-processing invocation. The field does not contain a value when generating keys.</p> <p>The exit can put values in this field only if a key was supplied on the control statement. The exit-supplied value must be edited for hexadecimal values; it then replaces the values entered on the input control statement.</p>
380	4	<p>CSFKEYS record for transkey, key-label1.</p> <p>The address of the CSFKEYS data set record that is output for transkey key-label1 on the KGUP control statement. This field only contains a value when CLEAR keys are generated.</p> <p>The exit can modify the field at the record postprocessing invocation. KGUP sets the address to zero for the KGUP pre- or postprocessing and record preprocessing invocations.</p> <p>KGUP does not check the field upon return from the exit. Normal CSFKEYS processing applies. KGUP uses key values on control statement creation.</p>

Table 29. KGXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
384	4	<p>CSFKEYS record for transkey, key-label2.</p> <p>The address of the CSFKEYS data set record that is output for transkey key-label2 on the KGUP control statement. This field only contains a value when TRANSKEY key-label2 is specified for generated keys.</p> <p>The exit can modify the field at the record postprocessing invocation. KGUP sets the address to zero for the KGUP pre- or postprocessing and record preprocessing invocations.</p> <p>KGUP does not check the field upon return from the exit. Normal CSFKEYS processing applies. KGUP uses key values on control statement creation.</p>
388	4	<p>CSFCKDS header record.</p> <p>The address of the CSFCKDS data set header record.</p> <p>The exit can check the field at the KGUP pre- or postprocessing invocations. However, the exit can modify the field only at the KGUP postprocessing invocation. KGUP sets the value of the field to zero for the record pre- or postprocessing invocations.</p> <p>The exit can modify the installation data field of the CKDS header record (see “Debugging aids” on page 156 for a description of the CKDS header record. Offset +196 of the CKDS header record is the installation data field). The installation data field supplied by the exit is placed in the CKDS header record after the KGUP postprocessing invocation returns control to KGUP.</p>
392	4	<p>CSFCKDS record.</p> <p>The address of the CSFCKDS data set record processed by the KGUP control statement. KGUP sets the address to zero if the TRANSKEY keyword has two labels of transport keys.</p> <p>The exit can check the field only at the record postprocessing invocation. KGUP sets the address to zero for the record preprocessing and KGUP pre- or postprocessing invocations.</p> <p>The exit can modify the record area if the TRANSKEY keyword does not have two labels.</p>
396	4	<p>RENAME CSFCKDS record.</p> <p>The address of the CSFCKDS data set record processed when the RENAME verb is used in a control statement. You can determine whether the RENAME verb was used by examining bit 3 at offset +14 in KGXP.</p> <p>The exit can modify the field at the record postprocessing invocation. KGUP sets the address to zero for the record preprocessing and KGUP pre- or postprocessing invocations.</p> <p>The exit can modify the record area. KGUP does not check this field upon return from the invocation. Normal CSFCKDS processing applies.</p>

Table 29. KGXP Control Block Format (continued)		
Offset (Dec)	Number of Bytes	Description
400	4	Installation data. The address of the data specified on the INSTDATA keyword of the KGUP control statement. The address of the area is zero if a SET control statement has not been processed. “The SET statement” on page 211 describes how to use the field in greater detail.
404	4	Installation exit area. The address of an area set by the installation that is preserved across all invocations of the exit. The first byte of the area contains the length of the area (including the length byte). After KGUP completes, the first 64 bytes of the area are written to the SMF data set. The exit has exclusive control of modifying this area. The area is only used as input to SMF processing upon completion of KGUP.

The SET statement

Use the SET control statements to specify data to send to a KGUP installation exit. For a more detailed description of the SET statement, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

The installation data field in KGXP (offset +396) contains the address of the data SET statement specifies. Data that is specified on a SET statement can be especially useful if you alter key entries. You may want to keep track of the entries you change by putting the original data and the changed data in the installation data area.

Return codes

You can pass a return code back to KGUP in the KGXP control block (offset +8). The exit can use the return code to cause KGUP to reject control statements or to end KGUP. Return code values, in decimal, for record pre- or postprocessing exit calls are:

Return Code Description

- 0**
Normal, continue processing.
- 4**
Reject control statement, but do not end KGUP.
- 8**
End KGUP.

All other return codes are not valid and cause KGUP to end.

Return code values, in decimal, for the KGUP pre- or postprocessing invocations are:

Return Code Description

- 0**
Normal, continue processing.
- >0**
End KGUP.

Chapter 6. Installation-defined Callable Services

This topic contains Programming Interface information.

ICSF provides callable services that perform cryptographic functions. For example, the ICSF encipher callable service enciphers data. You call and pass parameters to a callable service from an application program. See [z/OS Cryptographic Services ICSF Application Programmer's Guide](#) for a description of the ICSF callable services.

Besides the callable services that ICSF provides, you can write your own callable services; these are known as *installation-defined callable services*.

Important: Only an experienced system programmer should attempt to write an installation-defined callable service. The writing and installation of such a service require a thorough knowledge of system programming in an z/OS environment. If, without having this knowledge, you attempt to write or to install installation-defined callable services, you run the risk of seriously degrading the performance of your system and causing complete system failure.

To write an installation-defined callable service, you must first write the callable service and link-edit it into a load module. Then define the service in the installation options data set. Use the SERVICE installation option keyword to specify a number to identify the service and the load module that contains the service.

You must also write a service stub. To run an installation-defined callable service, you call a service stub from your application program. The service stub connects the application program with the installation-defined callable service. In the service stub, you specify the service number that identifies the callable service.

During ICSF startup, ICSF loads the load module that contains the service into the ICSF address space with the ICSF callable services. ICSF binds the service with the service number that you specified in the installation options data set.

This topic describes how to perform these tasks:

- Write a callable service.
- Define a callable service.
- Write a service stub.

Writing a callable service

An installation-defined callable service receives parameters from the application program when the program calls the service stub that is associated with the service. An installation-defined service can also access information in the secondary parameter block (SPB). The address of the SPB is passed in register 0. See [“Secondary parameter block” on page 183](#) for a description of the SPB.

The service receives control with these characteristics.

- Supervisor state
- Key 0
- APF authorized
- TCB or SRB mode
- Cross memory mode
- AR mode
- AMODE(31) or AMODE(64)
- RMODE(ANY)

The service can change the characteristics during their processing. However, the service must return to its caller with the same characteristics as on entry.

You must write the services in assembler, because you are in Access Register and cross memory mode, and the addresses of some of the parameters you may access are ALET-qualified. In particular, parameters passed into a callable service are in the user's address space, which you can access with an ALET of 1. See *z/OS MVS Programming: Extended Addressability Guide* for information about cross memory and AR mode.

Contents of registers

The contents of the registers on entry to the callable service are:

Register 0

Address of the secondary parameter block (SPB)

Register 1

Address of the parameter list

Register 2–13

Unpredictable

Register 14

Return address

Register 15

Service entry point address

The contents of the registers on exit from the callable service are:

Register 0

Reason code

Register 1–14

Same as on entry

Register 15

Return code

[Figure 5 on page 215](#) shows an example of entry and exit code for a generic service.

```

MYSERV    CSECT
MYSERV    AMODE    31
MYSERV    RMODE    ANY
          USING    *,15
          B         PROLOG                Branch around header text
          DC        C'some text'
          DC        C'compile date/time'
PROLOG     EQU      *
          DROP      15
          BSM       R14,0
          BAKR      14,0                Save callers info on stack
          LAE       12,0                Clear access register 12
          LR        12,15                Load reg 15 into 12
PROGSTR    EQU      *
          USING     MYSERV,12            Set up base register
          *                                     addressability
          .
          .
          .
          Get dynamic area for program
          .. STORAGE OBTAIN or CELLPOL or own scheme ...
          .
          .
          Free dynamic area for program
          .
          .
          .
RETURN     L        0,REASON_CODE        Put reason code in reg 0
          L        15,RETURN_CODE       Put return code in reg 15
          PR

```

Figure 5. Example of a service entry and exit

The example uses the instructions BAKR and PR to replace standard linkage. With these instructions, you no longer need to pass the save area in a register.

If the callable service ends abnormally, ICSF takes a system dump. The ICSF service functional recovery routine (FRR) PROTECTS an installation-defined service. You can, however, write your own recovery routine.

Security access control checking

For the ICSF-defined services, ICSF performs security access control checking to determine if the caller is authorized to access the service and the results of the authorization check can be logged in SMF. This checking is not performed by ICSF for installation-defined services or UDXs. Any security access control checking must be performed by the installation-defined service or UDX itself.

Checking the parameters

For the ICSF-defined services, ICSF checks the integrity of user-passed parameters. An error in a parameter that causes a system abend does not cause a system dump. For an installation-defined callable service, you must perform your own integrity checking of parameters. An error in a user parameter that results in a system abend causes a system dump. You can suppress the system dump by setting a bit on in the SPB. To suppress the dump, set the bit on before you check the integrity of the parameters. This bit (the SPBTERM bit) is the third bit of the flag byte at offset 16 in the SPB.

Link-editing the callable service

After you write the callable service, you need to link-edit it into a load module, and install the load module into an APF authorized library. ICSF uses this normal search order to locate the service:

- Job pack area
- Steplib (if one exists)
- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

Defining a callable service

Use the SERVICE keyword in the installation options data set to specify information about the callable service. ICSF uses this information at ICSF startup to enable the service. See [“Steps to create the installation options data set”](#) on page 24 for more information about ICSF installation options.

The SERVICE keyword has this syntax:

SERVICE(service-number,load-module-name,FAIL(fail-option))

The service-number is a number that identifies the service to ICSF. The valid service numbers are 1 through 32767, inclusive. The load-module-name is the name of the module that contains the service your installation wrote. During ICSF startup, ICSF loads the module and binds it to the service number you specified.

Using the fail-option, you specify the action ICSF takes if the loading of the service ends abnormally. ICSF loads all installation-defined services at ICSF startup.

Specify one of these values for the fail-option:

YES

ICSF abends if your service cannot be loaded.

NO

ICSF continues to start if your service cannot be loaded.

If the callable service ends abnormally while it is processing, ICSF does not end.

This SERVICE installation option statement identifies a specific installation-defined service to ICSF:

```
SERVICE(50,KSUST,FAIL(NO))
```

When ICSF starts, it binds the service number 50 to the load module KSUST, which contains the callable service you wrote. Because the fail option is NO, if your service cannot be loaded, ICSF continues to start anyway.

Writing a service stub

Besides writing the callable service itself, you must write a service stub, which is the connection between the application program and the installation-defined service. In an application program, you call the service stub, which accesses the installation-defined service. The service stub can be any name you choose to call it.

The service stub must:

- Check that ICSF is active.
- Place the service number for the installation-defined callable service into register 0.
- Call the IBM-supplied processing routine, CSFAPRPC.

CSFAPRPC is used to access the callable services on ICSF. In the service stub, you must call CSFAPRPC. ICSF stores the address of the CSFAPRPC entry point in the CCVTPRPC field of the ICSF cryptographic communication vector table (CCVT). If running in a CICS address space, then, after you call CSFVCCPP, the system calls the callable service that corresponds to the service number in register 0. [“The Cryptographic Communication Vector Table \(CCVT\)”](#) on page 400 describes the format of the CCVT.

The contents of the registers on entry to the service stub are:

Register 0

Unpredictable

Register 1

Address of the parameter list

Register 2–13

Unpredictable

Register 14

Return address

Register 15

Service stub entry point address

The contents of the registers on exit from the service stub are:

Register 0

Reason code

Register 1–14

Same as on entry

Register 15

Return code

To run an installation-defined callable service, an application program calls the service stub. You must link-edit the service stub with the application program that calls the service stub. Any application program that calls a service stub must be link-edited with the service stub.

To call an installation-defined service from an application program, use this statement:

```
CALL <service-stub-name> <service-parameters>
```

The service-stub-name is the name of the service stub for the installation-defined callable service. The service-parameters are the parameters you want to pass to the installation-defined service. You supply the parameters according to the syntax of the programming language that you use to write the application program.

Example of a service stub

[Figure 6 on page 218](#) through [Figure 10 on page 222](#) show an example of a service stub for an installation-defined callable service.

```

**** START OF SPECIFICATIONS ****
*
*   MODULE NAME = CSFGEN
*   DESCRIPTIVE NAME = SERVICE STUB
*
*   FUNCTION =
*   THIS IS A SAMPLE SERVICE STUB. IT IS MEANT TO BE LINKEDITED
*   WITH THE APPLICATION AND ENTERED VIA A CALL CSFGEN. THIS STUB
*   CAUSES THE EXECUTION OF THE SERVICE WITH SERVICE NUMBER = 50
*   (DECIMAL).
*   MODULE TYPE = ASSEMBLER
*   PROCESSOR = ASSEMBLER
*   MODULE SIZE = ONE BASE REGISTER
*
**** END OF SPECIFICATIONS ****
CSFGEN  START 0
GENSNUM EQU 50
CSFGEN  CSECT
CSFGEN  AMODE 31
CSFGEN  RMODE ANY
MAINENT DS 0H
        USING *,R15
        LAE R15,0(R15,0)
        L   R15,=A(CICSTEST)
        BAKR 0,R15          PR from CICSTEST will restore GPRs
        LTR R15,R15
        BC 2,NOCICS

*
YESCICS DS 0H
        SAC 0
        STM R14,R12,12(R13)
        LR R12,R15
        DROP R15
        USING MAINENT,R12
        LR R3,R0
        B NORMAL

*
        NOCICS DS 0H
        USING MAINENT,R12
        BSM R14,0
        BAKR R14,0
        LAE R12,0
        LR R12,R15
        SLR R13,R13

*****
* At this point, R0 must contain the service number.
* If we are to call the TRUE, R13 is non-zero
* R1 points to the caller's parameter list.
*****
NORMAL DS 0H
        LA R0,GENSNUM          R0 gets service number
        SLR R10_ZERO,R10_ZERO
        LR RC,R10_ZERO
        L R2,CVTPTR
        USING CVT,R2
        L R2,CVTABEND

```

Figure 6. Example of a service stub (1 of 5)

```

        CLR    R2,R10_ZERO
        BC     8,NOICSF
        USING  SCVTSECT,R2
        L      R2,SCVTCCVT
        CLR    R2,R10_ZERO
        BC     8,NOICSF
        USING  CCVT,R2
        TM     CCVTSFG1,B'00110000' IS ICSF ACTIVE
NOICSF  BC     1,YESICSF
        LA     RC,12          Set return code to 12 decimal
        L      R7,RETURN_CODE_PTR(,R1)
        ST     RC,RETURN_CODE(,R7)
        SLR    R0,R0
        L      R7,REASON_CODE_PTR(,R1)
        ST     R0,REASON_CODE(,R7)
        B      FINISHED
YESICSF DS     0H
*****
* Note that, if we're in CICS, the prolog code pointed R3 at the AFCB
* and R13 at the caller's savearea--they're still pointing. Also, R0
* contains the service number, with the high order bit ON if the TRUE
* has been tried and found wanting. In this last case, CSFVCCPP will
* check the high order bit and not attempt to call the TRUE.
* If R13 is zero, we're using the linkage stack. That means we can
* call CSFAPRPC.
* If R13 is not zero, we're using non-stack linkage. That means the
* caller's savearea will be used. CSFVCCPP uses this kind of linkage.
* But note that CSFVCCPP will not return here. Instead, it will return
* directly to the caller--that is, to the owner of the only save
* area around.
*****
        CLR    R13,R10_ZERO
        BC     8,EXECPRPC
        L      R15,CCVTPRPD
        BALR   R14,R15
LR      RC,R15
        B      FINISHED
EXECPRPC L      R15,CCVTPRPC
        BALR   R14,R15
        LR     RC,R15
FINISHED DS     0H
*
*****
* This routine uses the linkage stack to save the caller's regs
* if this is not a CICS environment. In CICS, it uses the save
* area pointed to by register 13. So the epilog code takes one
* of two forms. If this is CICS (i.e. if R13 is non-zero),
* return is via LM and BR 14. If this is not CICS, return is
* via PR.
*
* On return, the PR of ESA linkage does not restore registers
* 0, 1, 14 and 15. In the LM of normal BR 14 linkage, however,
* everything but 13 gets restored. Since this routine has no
* autodata, there's no way to pass back return and reason codes
* unless we leave 0 and 15 intact. The solution is to deviate
* slightly from normal BR 14 linkage and restore only registers
* 1 through 12 and 14.
*****
        LTR    R13,R13
        BC     8,ENDNOCICS

```

Figure 7. Example of a service stub (2 of 5)

```

ENDCICS    LR    R15,RC
           L     R14,SAVE14(,R13)
           LM    R1,R12,24(R13)
           BR    R14

*
EDNOCICS   DS    0H
           LR    R15,RC
           LA    R7,12
           CR    R15,R7
           BNE   ENDSVC
           LA    R7,16
           CR    R0,R7
           BNE   ENDSVC
           L     R7,RETURN_CODE_PTR(,R1)
           ST    R15,RETURN_CODE(,R7)
           L     R7,REASON_CODE_PTR(,R1)
           ST    R0,REASON_CODE(,R7)
ENDSVC     LR    R15,RC
           PR

*****
*****
**  CICSTEST:  Decides whether this is a CICS environment
*****
*****
CICSTEST DS    0H
           LAE   R12,0                Clear AR 12
           LR    R12,R15              Addressability via R12
           USING CICSTEST,R12
           L     R15,=A(CSFGEN)       R15 gets caller's base reg
           L     R2,CVTPTR            GET CVT POINTER
           USING CVT,R2
           L     R2,CVTABEND          AND SECONDARY CVT POINTER
           USING SCVTSECT,R2
           L     R2,SCVTCCVT         POINT TO CSF CCVT
           LTR   R2,R2               IS CRYPTO INSTALLED?
           BZ    RETRN               IF NOT, GO HOME
           USING CCVT,R2
           TM    CCVTSFG1,B'00110000' IS ICSF ACTIVE
           BNO   RETRN              IF NOT , GO HOME

* Check for wait list routine
*
           TM    CCVTCICS,B'10000000' Q. CCVTPRPA ON?
           BZ    RETRN               no---No CICS capability
           TM    CCVTCICS,B'01000000' Q. CCVTCKWL ON?
           BZ    CKWLHERE            no---use imbedded routine
                                     yes--use installed routine
*
           LA    R0,GENSNUM          R0 gets service number
           LR    R3,R1               R3 saves R1
           LR    R4,R14              R4 saves R14
           LR    R5,R15              R5 saves R15
           L     R15,CCVTCKWL        R15 gets routine address
           BALR  R14,R15             Go check for CICS
           LR    R0,R15              Save return code in R0
           LR    R15,R5              Restore R15
           LR    R14,R4              Restore R14
           LR    R1,R3              Restore R1
           LTR   R0,R0               Q. CICS?
           BZ    RETRN               no---return
                                     yes--pass info along
*
           O     R15,M_CICS          Enable high bit of R15 to CICS
           B     RETRN              Return

```

Figure 8. Example of a service stub (3 of 5)


```

* Cannot use installed routine.  Use imbedded routine
*
CKWLHERE DS      0H          Imbedded check for TRUE routine
      SLR      R0,R0          Init R0 to 0
      CPYA     R8,R12         Zero AR 8
      SLR      R8,R8          Init R8 to 0
      USING    PSA,R8
      L        R8,PSATOLD     R8->TCB
      USING    TCB,R8
      LTR      R8,R8          Q.  Is there a TCB?
      BC       8,RETRN        no---return
*                               yes--check state and key
      CPYA     R11,R12        Zero AR 11
      LA       R11,1          Get PSW state and key in R6
      ESTA     R6,R11
      LR       R7,R6          Copy of state & key in R7
      N        R7,M_KEY       Q.  problem key?
      BZ       RETRN          no---return
*                               yes--check state
      N        R6,M_STATE     Q.  problem state?
      BZ       RETRN          no---return
*                               yes--get the CICS eye-catcher
      LA       R6,2           Set ARs 6 and 8 to home
      SAR      R6,R6
      SAR      R8,R6
      L        R8,TCBEXT2     R8->TCB extension
      USING    TCBXTNT2,R8
      ICM      R4,B'1111',TCBCAUF  R4 gets AFCX address
*                               Q.  Address there?
      BZ       RETRN          no---return
*                               yes--check eye-catch
      CLC      0(4,R4),CICS_EYE  Q.  CICS?
      BNE      RETRN          no---return
*                               yes--pass info along
      LR       R0,R4          R0 gets the AFCX pointer
      O        R15,M_CICS     Enable high order bit of R15
RETRN  DS      0H
      DROP     R12           Free R12
      PR                               Return from CICSTEST subroutine
*
      LTORG
      DS      0D
*
GENSDATA DS      0F
R10_ZERO EQU     10
RC        EQU     05
R0        EQU     0
R1        EQU     1
R2        EQU     2
R3        EQU     3
R4        EQU     4
R5        EQU     5
R6        EQU     6
R7        EQU     7
R8        EQU     8
R9        EQU     9
R10       EQU     10
R11       EQU     11
R12       EQU     12
R13       EQU     13
R14       EQU     14
R15       EQU     15
*

```

Figure 9. Example of a service stub (4 of 5)

Chapter 7. Converting a CKDS from fixed length to variable length record format

Note: ICSF recommends converting your CKDS to the common record format to support variable-length symmetric key tokens. Additional functionality is available with the common record format.

ICSF provides a CKDS conversion program, CSFCNV2, that converts a fixed-length record format CKDS to the variable-length record format. There will be no changes to the key token in the CKDS record. Only the format of the record will be changed.

Note: The CSFCNV2 utility converts a fixed-length format CKDS to a variable-length format. To convert a fixed-length or variable-length format CKDS to the KDSR format, see [“Migrating to the common record format \(KDSR\) key data set”](#) on page 102.

You can also use the CSFCNV2 utility to rewrap encrypted DES values in the CKDS. For more information on this capability of the CSFCNV2 utility, refer to [z/OS Cryptographic Services ICSF Administrator's Guide](#).

There is no conversion from variable length to fixed length records.

You run the conversion utility program by submitting a batch job. On the EXEC statement, specify PGM=CSFCNV2.

This example is a JCL that runs the conversion program:

```
//CKDSCNV2 EXEC PGM=CSFCNV2,PARM='FORMAT,OLD.CKDS,NEW.CKDS'
```

Where:

OLD.CKDS

The fixed length record format CKDS to be converted. This is the source CKDS for the conversion.

NEW.CKDS

An empty disk copy of a variable length record format CKDS. This is the CKDS into which the conversion utility writes the converted records. The data set must be defined and empty before you run the conversion program.

ICSF no longer ships a sample to create a CKDS in variable length record format (maximum LRECL = 1024). It is recommended to convert to common record format.

The CSFV0560 message in the joblog will indicate the results of processing.

Return Code

Meaning

0

Process successful.

4

Minor error occurred.

8

SAF authorization check failed.

12

Process unsuccessful.

60 or 92

CKDS processing has failed. A return code 60 indicates the error was detected in the new KDS. A return code 92 indicates the error was detected with the old KDS.

When the program is invoked from another program, the invoking program receives the reason code in General Register 0 along with the return code in General Register 15. The following list describes the meaning of the reason codes. If a particular reason code is not listed, refer to the listing of ICSF and TSS return and reason codes in the [z/OS Cryptographic Services ICSF Application Programmer's Guide](#).

Return code 0 has this reason code:

Reason Code	Meaning
-------------	---------

36132

CKDS reencipher/Change MK processed only tokens encrypted under the DES master key.

Return code 4 has these reason codes:

Reason Code	Meaning
-------------	---------

0

Parameters are incorrect.

4004

Rewrapping is not allowed for one or more keys.

36112

CKDS conversion completed successfully but some tokens could not be rewrapped because the control vector prohibited rewapping from the enhanced wrapping method.
--

36164

Input CKDS is already in the variable-length record format. No conversion is necessary.

Return code 8 has this reason code:

Reason Code	Meaning
-------------	---------

16000

Invoker has insufficient SAF access authority to perform function.
--

Return code 12 has these reason codes:

Reason Code	Meaning
-------------	---------

0

ICSF has not been started

11060

The required cryptographic coprocessor was not active or the master key has not been set
--

36000

Unable to change master key. Check hardware status.

36008

Crypto master key register or registers in improper state.
--

36020

Input CKDS is empty or not initialized (authentication pattern in the control record is invalid).

36036

The new master key register for Coprocessor 1 (C1) is not full, but C0 is ready and the current master key is valid.
--

36040

The new master key register for C0 is not full, but C1 is ready and the current master key is valid.
--

36044

The master key authentication pattern for the CKDS does not match the authentication pattern of the coprocessors, which are not equal.
--

36048

The master key authentication pattern for the CKDS does not match the authentication pattern of either of the coprocessors, which are not equal.
--

36052

A valid new master key is present in C0, but its authentication pattern does not match that of C1 or the CKDS, which are equal.

36056

A valid new master key is present in C1, but its authentication pattern does not match that of C0 or the CKDS, which are equal.

36060

The new master key register or registers are not full.

36064

Both new master key registers are full but not equal.

36068

The input KDS is not enciphered under the current master key.

36076

The new master key register for C0 is not full, but the CPUs are online.

36080

The new master key register for C1 is not full, but the CPUs are online.

36084

The master key register cannot be changed since ICSF is running in compatibility mode.

36104

Option not available. There were no Cryptographic Coprocessors available to perform the service that was attempted.

36108

PKA callable services are enabled, and the PKDS is the active PKDS as specified in the options data set.

36120

The CKDS is unusable. The CKDS does not support record level authentication.

36124

The CKDS is unusable. The CKDS only supports encrypted AES keys and encrypted DES support is required.

36128

The CKDS is unusable. The CKDS does not support encrypted DES keys which is required.

36160

The attempt to reencipher the CKDS failed because there is an enhanced token in the CKDS.

36168

A CKDS has an invalid LRECL value for the requested function. For wrapping, the input and output CKDS LRECLs must be the same.

36172

The level of hardware required to perform the operation is not available.

Return code 60 or 92 has these reason codes:

Reason Code**Meaning****3078**

The CKDS was created with an unsupported LRECL.

5896

The CKDS does not exist.

6008

A service routine has failed.

The service routines that may be called are:

CSFMGN

MAC generation

CSFMVR

MAC verification

CSFMKVR

Master key verification

6012

The single-record, read-write installation exit (CSFSRRW) returned a return code greater than 4.

6016

An I/O error occurred reading or writing the CKDS.

6020

The CSFSRRW installation exit abended and the installation options EXIT keyword specifies that the invoking service should end.

6024

The CSFSRRW installation exit abended and the installation options EXIT keyword specifies that ICSF should end.

6028

The CKDS access routine could not establish the ESTAE environment.

6040

The CSFSRRW installation exit could not be loaded and is required.

6044

Information necessary to set up CSFSRRW installation exit processing could not be obtained.

6048

The system keys cannot be found while attempting to write a complete CKDS data set.

6052

For a write CKDS record request, the current master key verification pattern (MKVP) does not match the CKDS header record MKVP.

6056

The output CKDS is not empty.

Note: It is possible that you will receive MVS reason codes rather than ICSF reason codes, for example, if the reason code indicates a dynamic allocation failure. For an explanation of Dynamic Allocation reason codes, see [*z/OS MVS Programming: Authorized Assembler Services Guide*](#).

Chapter 8. Migration from PCF to z/OS ICSF

If your installation uses the cryptographic product, Programmed Cryptographic Facility (PCF), ICSF helps you migrate PCF applications to ICSF. You can run PCF applications on ICSF to gain the enhanced performance and availability of ICSF and to test ICSF. Eventually, you should convert these applications to use ICSF services, rather than the PCF macros.

During migration, you can run PCF applications on ICSF because ICSF continues to support the PCF macros (GENKEY, RETKEY, EMK, and CIPHER). If GENKEY or RETKEY macro exits exist, you should reevaluate their applicability to ICSF. If an exit performs a necessary function, you need to rewrite the exit for ICSF. Exits exist for the compatibility services on ICSF.

If a PCF application uses a key in the PCF cryptographic key data set, you must convert the key to an ICSF cryptographic key data set before you run the PCF application on ICSF. ICSF provides a program to make this conversion.

Running PCF and z/OS ICSF on the same system

You can run PCF and ICSF simultaneously on the same z/OS system or separately in three different modes. You can run ICSF in compatibility, coexistence, or noncompatibility mode.

In compatibility mode, you can run either PCF or ICSF, but you cannot run them simultaneously on the same z/OS system. You can continue to run PCF applications on PCF or you can run PCF applications on ICSF. ICSF supports the PCF macros that the PCF applications call. However, you cannot run the PCF key generator utility program (KGUP) on ICSF. You do not have to reassemble PCF applications to run the applications on ICSF.

In coexistence mode, you can run PCF and ICSF simultaneously on the same z/OS system. You can continue to run a PCF application on PCF or you can reassemble the PCF application to run on ICSF. In this mode, ICSF supports the PCF macros when a reassembled PCF application calls these macros.

In noncompatibility mode, you can run PCF and ICSF simultaneously and independently on the same z/OS system. You can run PCF applications on PCF and ICSF applications on ICSF. You cannot run PCF applications on ICSF, because ICSF does not support the PCF macros in this mode.

You can run PCF simultaneously and independently in coexistence and noncompatibility mode. Therefore, in these modes, you can run PCF KGUP on PCF while running ICSF. The PCF KGUP updates keys on a PCF CKDS.

The ICSF installation option COMPAT(YES, COEXIST or NO) allows you to specify which mode you want ICSF to run in. You specify COMPAT(YES) for compatibility mode, COMPAT(COEXIST) for coexistence mode, and COMPAT(NO) for noncompatibility mode. See [“Steps to create the installation options data set” on page 24](#) for information about creating the installation options data set and [“Parameters in the installation options data set” on page 35](#) for details about these options.

Running in compatibility mode

In compatibility mode, you can run a PCF application on ICSF without reassembling the application. A PCF application running on ICSF can still use PCF macros, because ICSF supports these macros. The PCF application gains the enhanced performance, reliability, and availability of ICSF.

You cannot run PCF and ICSF simultaneously on the same z/OS system in compatibility mode. If you start PCF, you must stop PCF before you can start ICSF. If you start ICSF, you must stop ICSF before you can start PCF.

A PCF application may have used keys on the PCF cryptographic key data set (CKDS). When you run the application on ICSF, these keys must be in the ICSF CKDS. The format of a key entry on the PCF CKDS differs from the format of a key entry on the ICSF CKDS. Therefore, you need to run a conversion program

to convert the PCF CKDS entries and place the entries in the ICSF CKDS. See [“Converting a PCF CKDS to ICSF format” on page 230](#) for a description of how to convert a PCF CKDS.

For encryption, ICSF supports the Data Encryption Standard (DES).

PCF macros receive identical error return codes if they run on ICSF or PCF, with one exception. If a key is installed on the ICSF CKDS with the correct label but with the wrong key type, an attempt to use that key by RETKEY or GENKEY results in a return code of 8 from PCF. This indicates that the key was not of the correct type. ICSF issues return code 12, indicating that it could not find the key. Ensure that PCF LOCAL or CROSS 1 keys are installed in the ICSF CKDS as EXPORTER keys. Also, ensure that REMOTE and CROSS 2 keys are installed in the ICSF CKDS as IMPORTER keys.

In compatibility mode, the safest method for changing the master key is to re-IPL the system. To change the master key in compatibility mode, see [“Changing the DES master key in compatibility or coexistence mode” on page 229](#).

Note: To use AMS REPRO encryption, you need to run ICSF in compatibility mode.

Running in coexistence mode

In coexistence mode, you can run ICSF and PCF simultaneously on the same z/OS system and run a PCF application on PCF or on ICSF. A PCF application running on ICSF gains the enhanced performance, reliability, and availability of ICSF.

A PCF application running on ICSF can still use PCF macros, because ICSF supports these macros. ICSF ships changed PCF macros in SAMPLIB that run only on ICSF. Because these changed PCF macros already exist unchanged on PCF, the changed PCF macros shipped with ICSF are named differently.

On ICSF, in SAMPLIB:

- The changed PCF EMK macro is named CSFEMK.
- The changed PCF CIPHER macro is named CSFCIPH.
- The changed PCF RETKEY macro is named CSFRKY.
- The changed PCF GENKEY macro is named CSFGKY.

You can rename these macros to the PCF names when you want to run a PCF application on ICSF.

To run a PCF application on ICSF, you must:

- Rename the changed PCF macro shipped in ICSF SAMPLIB to the appropriate PCF name.
- Place the macro in the appropriate macro library.
- Reassemble the PCF application against the changed PCF macro.

Then the application can run only on ICSF. To run a PCF application on PCF, just run the application without reassembling the application.

During migration, you can start ICSF and start PCF so that both products are running simultaneously. If you want to run a PCF application using the PCF macros on PCF, do not reassemble the application. If you want to run a PCF application using the changed PCF macros on ICSF, reassemble the application against the changed macros. Coexistence mode enables you to run the products simultaneously and choose whether to run a PCF application on PCF or ICSF.

A PCF application can use keys on the PCF CKDS. When you run the application on ICSF, those keys must be in the ICSF CKDS. The format of a key entry on the PCF CKDS differs from the format of a key entry on the ICSF CKDS. Therefore, you need to run a conversion program to convert the PCF CKDS entries and place the entries in the ICSF CKDS. See [“Converting a PCF CKDS to ICSF format” on page 230](#) for a description of how to convert a PCF CKDS.

In coexistence mode, the safest method for changing the master key is to re-IPL the system. See [“Changing the DES master key in compatibility or coexistence mode” on page 229](#) for a description of the process used to change the master key in coexistence mode.

Changing the DES master key in compatibility or coexistence mode

In compatibility and coexistence modes, the safest way to activate the DES master key after changing it is to re-IPL the system. This process is different from the usual process for entering and activating a master key. For information about changing the master key, see [*z/OS Cryptographic Services ICSF Administrator's Guide*](#).

A re-IPL ensures that a program does not access a cryptographic service with a key that is encrypted under a different master key. If a program is using an operational key, the program either re-creates the key or imports the key again.

In compatibility or coexistence mode, the ICSF administrator can use the ICSF panels to enter the key value into the new master key register. However, the master key cannot be *activated* using the panels in compatibility or coexistence mode. The value entered remains in the new master key register until you re-IPL the system. (In noncompatibility mode, the ICSF administrator can use the ICSF panels to enter the key value into the new master key register and to activate the master key.)

If the new master key is different than the current master key, the ICSF administrator must reencipher the CKDS under this new master key. To do this, choose the REENCIPHER CKDS option on the master key management panel. This reenciphers a CKDS under the master key in the new master key register. Reencipher all the disk copies of the CKDSs, and leave the ICSF panels without changing the master key.

Then re-IPL the system and restart ICSF. In the installation options data set, the CKDSN installation option must specify a disk copy of the CKDS that is reenciphered under the new master key. When ICSF starts again, it detects that the current master key is not the one that enciphered the CKDS that is specified in the installation options data set. ICSF detects that the CKDS is enciphered under the new master key and makes that master key active.

If your installation requires 24-hour availability and it is not possible to re-IPL the system, an alternative method is to stop all cryptographic applications, especially those using PCF macros. This helps eliminate inadvertent use of operational keys that are encrypted under the old master key. After you restart CSF, applications using an operational key can either re-create or reimport the key.

Running in noncompatibility mode

In noncompatibility mode PCF and ICSF can run simultaneously and independently. You can run both ICSF and PCF at the same time. Just start one and then the other. Both ICSF and PCF run completely separate from each other. Each has its own applications and each uses its own services and CKDS.

You cannot run a PCF application on ICSF, even if you reassemble it. If you run an application on ICSF that calls a PCF macro, the application ends abnormally, because ICSF does not support the PCF macros in noncompatibility mode.

Because each product runs separately, neither product loses any function in exchange for compatibility. When ICSF is in compatibility or coexistence mode, you can no longer change the master key dynamically. In noncompatibility mode, this function is still possible. Therefore, except for when your installation is migrating to ICSF, you probably want to run ICSF in noncompatibility mode.

Note: When you initialize ICSF for the first time, noncompatibility mode must be active.

Specifying compatibility modes during migration

The process and duration to migrate from PCF to ICSF depend on your installation. You can use different modes in different stages of migration. To change modes, change the COMPAT option in the installation options data set and restart ICSF. When you complete migration to ICSF, you can run in noncompatibility mode to use the full function of ICSF.

When you first install an ICSF system, you can continue to run PCF for production and just test ICSF. Because you are running the products separately but simultaneously on the same z/OS system, you can run in noncompatibility or coexistence mode. To run in compatibility mode, you need more than one z/OS system. You can run the test applications on ICSF on one z/OS system while you run your production on PCF on another z/OS system.

When you begin testing ICSF, you can run existing applications in either compatibility mode or coexistence mode to test the PCF macros on ICSF. After you run the test applications, you may want to bring up production using PCF applications on ICSF. When you bring over PCF applications to ICSF, you can run in coexistence mode. In this mode, you can run an application on PCF and then reassemble the application to run the application on ICSF.

While, or after, you bring PCF applications into production on ICSF, you can run test applications that call ICSF services. You can then convert the applications that call PCF macros to applications that call the ICSF services. The ICSF services provide enhanced key separation, performance, and function. After you convert all your PCF applications to ICSF applications, you can activate noncompatibility mode and have the full function of ICSF.

Converting a PCF CKDS to ICSF format

During migration, you may need to convert a PCF CKDS into ICSF CKDS format if:

- PCF applications running on ICSF use keys stored in a PCF CKDS.
- Your installation uses the PCF key generator utility program to create keys and uses ICSF for other cryptographic operations. To use the keys in ICSF applications, you must convert the PCF CKDS.

ICSF provides a PCF conversion program, CSFCONV, that converts a PCF CKDS into an ICSF CKDS. The conversion program runs with certain defaults. The program converts all the entries in a PCF CKDS and converts the PCF key types into certain corresponding ICSF key types. You can use the conversion program override file to instruct the conversion program not to convert certain entries. You can also tell the conversion program to convert a PCF key type into a different ICSF key type than the default.

These topics describe how:

- The conversion program runs with certain defaults
- To use the override file to make it run differently
- To run the conversion program

How the PCF conversion program runs

You can run the PCF conversion program only after you initialize the master key and CKDS for ICSF.

When the conversion program processes a PCF CKDS, the program duplicates the single length key values to create double length keys.

The conversion program merges the PCF CKDS with an input ICSF CKDS. The input ICSF CKDS is an existing disk copy of an ICSF CKDS. The input ICSF CKDS must contain a header record. For information about initializing an ICSF CKDS, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

The PCF conversion program places the input ICSF CKDS entries and the converted PCF entries into an output CKDS. You must create an empty VSAM data set to be the output CKDS before running the conversion program. See [“Steps to create the CKDS” on page 12](#) for information about creating the data set.

The PCF conversion program converts all the entries in a PCF CKDS. When you run the PCF conversion program, the program does these conversions of PCF key types into ICSF key types:

- Converts each PCF local key entry into an ICSF NOCV exporter key-encrypting key entry.
- Converts each PCF remote key entry into an ICSF NOCV importer key-encrypting key entry.
- Converts each PCF cross key entry into two ICSF key entries: an NOCV exporter key-encrypting key and an NOCV importer key-encrypting key.

You use the override file to not convert all the entries in a PCF CKDS or to convert a PCF key into a different key type than the default key type.

When the PCF conversion program converts a PCF entry, the program places any installation data from the installation data field of the PCF entry into the ICSF entry. You can use the override file to place different installation data into the ICSF entry.

Note: ICSF copies any installation data in the input CSF CKDS header record into the output ICSF CKDS header record.

As the conversion program reads the PCF CKDS, the input ICSF CKDS, and the override file, the program places key entries into a virtual image of the output ICSF CKDS. When the virtual image CKDS is complete, the conversion program reenciphers the key values of the PCF entries from under the PCF master key to under the ICSF master key. The conversion program places the reenciphered entries into the actual output CKDS.

As the conversion program creates the virtual image ICSF CKDS, the conversion program takes information from the PCF entry and possibly the override file. For each PCF entry, the conversion program checks if its key label exists in the override file. If the label does exist in the override file, the conversion program takes the action that is specified in the override file. The program either converts or bypasses the entry. If the key label does not exist in the override file, ICSF converts the entry.

The conversion program compares the converted PCF entries by label and type with the ICSF entries that already exist in the input ICSF CKDS. If there is a match, the conversion program replaces the key value from the converted entry of the PCF source into the virtual image CKDS. If there is not a match, the conversion program converts each PCF entry after checking the override file. If the label matches and the type does not, the conversion program checks to see if the type requires a unique label. If a unique label is not required, the conversion program converts the PCF entry after checking the override file. If a unique label is required, the conversion program does not convert the PCF entry and issues an error message. If the record type is DATA, DATAXLAT, MAC, MACVER, or NULL the CKDS record requires a unique label. The CKDS record also requires a unique label if the record has ever been updated by the dynamic CKDS update callable services. The conversion program also places all the input ICSF CKDS entries into the virtual image CKDS.

Calling installation exits during conversion

You can call two installation exits during conversion program processing: the conversion program exit (CSFCONVX) and the single-record, read-write exit (CSFSRRW). The conversion program calls the exit at three different times: before, during, and after conversion program processing. See [Chapter 5, “Installation exits,”](#) on page 165 for a description of the conversion program and single-record, read-write exit control blocks.

The conversion program calls the CSFCONVX exit after you submit the conversion program job, but before the program actually begins processing. At this point, you can use the exit to change the output ICSF CKDS header record installation data field.

The conversion program also calls the CSFCONVX exit during processing as the conversion program completes the virtual image ICSF CKDS, but before the conversion program reenciphers the key values. The conversion program calls the exit as it writes each record to the virtual image ICSF CKDS. At this point, you can use the exit to specify that the conversion program not place an entry into the output ICSF CKDS.

The conversion program also calls the CSFCONVX exit after the conversion program completes processing. At this point, you can use the exit to change the output ICSF CKDS header record installation data field.

As the conversion program reads the records from the virtual image ICSF CKDS to the actual output ICSF CKDS, it calls the single-record, read-write exit. The conversion program calls the single-record, read-write exit as it writes each record to the output ICSF CKDS. You can use this exit to specify that the conversion program not place an entry into the output ICSF CKDS.

The conversion program writes every entry from the PCF CKDS and input ICSF CKDS into the output ICSF CKDS unless an override record or installation exit indicates that the conversion program should bypass the entry from the PCF CKDS.

Using the conversion program override file

The conversion program converts all entries in a PCF CKDS into ICSF entries. The conversion program also converts each type of PCF key into a specific ICSF key type. If you want the conversion program to bypass

certain key entries or convert a specific key or key type differently than it does by default, use the override file.

By specifying override records, you can have the conversion program:

- Bypass conversion of key entries.
- Include information in key entries.
- Convert key types differently than it does by default.

These actions can relate to entries explicitly identified with a key label or entries that are identified globally.

You specify information in certain fields in an override record and leave other fields blank, depending on the action you want the conversion program to take. You can specify a global record affecting more than one PCF CKDS entry or a record that affects only one PCF CKDS entry.

All the override data set records should be in ascending sequence by key label and old key type. If you use global entries, they must be the initial entries in the override record. [Table 30 on page 232](#) shows the syntax of a record in the override file.

Note: All the fields should contain character values and be left-justified.

If you specify a key label in an override record, the conversion program processes the key entry identified by that key label. If you do not specify a key label in an override record, you are using a global override record. The conversion program processes all the key labels that pertain to the information specified by the override file.

You can use a global override record to affect all the entries in a CKDS and then use override records to explicitly affect entries you did not want to have that global override record affect.

Table 30. Format of Records in the Override File		
Column	Length	Description
1	8	Key Label The key label of the PCF entry you want to convert The field can have these values: <ul style="list-style-type: none">• Blanks• A key label existing in the PCF CKDS that you want to convert
9	1	This field must be blank.
10	8	Old Key Type The key type of the key entry you want to convert in the PCF CKDS. The field can have these values: <ul style="list-style-type: none">• Blanks• LOCAL• REMOTE
18	1	This field must be blank.

Table 30. Format of Records in the Override File (continued)

Column	Length	Description
19	8	<p>New Key Type</p> <p>The key type that you want the converted key entry to be in the ICSF CKDS. The master key variant for the key type enciphers the key in the ICSF CKDS entry that the conversion program creates.</p> <p>The field can have these values:</p> <ul style="list-style-type: none"> • Blanks • OPINENC • EXPORTER • IPINENC • IMPORTER
27	1	This field must be blank.
28	8	<p>Ignored</p> <p>In ICSF/MVS Version 1 Release 1, this field contained the key qualifier. The CKDS for ICSF/MVS Version 1 Release 2 or higher does not support key qualifiers. If your installation has a PCF conversion program override file created with ICSF/MVS Version 1 Release 1, you can still use it with z/OS ICSF. Any key qualifier entries are ignored.</p>
36	1	This field must be blank.
37	1	<p>Bypass Flag</p> <p>Used to indicate that an input CKDS entry is not to be included in the new ICSF CKDS. If you set this field to Y, the conversion program does not convert the entry.</p> <p>The field can have these values:</p> <ul style="list-style-type: none"> • Blank (same as N) • N • Y
38	1	This field must be blank.
39	52	<p>Installation Data</p> <p>Any additional information your installation records about a key. The information appears in the installation data field of the new ICSF CKDS.</p> <p>The field can contain any value.</p>

Bypassing conversion of entries

Using an override record, you can bypass a PCF entry so it is not converted and placed in the ICSF CKDS. You can use a global override record to bypass all the entries in the data set and then use explicit override records to convert certain entries. You can also convert most of a PCF CKDS and just bypass certain entries using explicit override records.

These are some examples of override records for bypassing conversion.

Example 1

This example shows an override record specifying that the conversion program not convert any PCF CKDS entry with a certain key label.

EXTOATM3	Y
----------	---

The conversion program bypasses any PCF CKDS entry with the label EXTOATM3.

Example 2

This example shows an override record specifying that the conversion program not convert any PCF CKDS entry with a certain key label and key type.

CRLABEL4 REMOTE	Y
-----------------	---

The conversion program bypasses any PCF CKDS entry with the label CRLABEL4 and key type REMOTE.

Example 3

This example shows a global override record specifying that the conversion program bypass all the entries in a PCF CKDS.

	Y
--	---

The conversion program does not convert any of the entries in the PCF CKDS.

After you specify this global override record, you can use explicit override records to convert certain entries in the PCF CKDS. For example, you can use an override record like this one to explicitly convert PCF entries with a certain label.

ATM03	N
-------	---

In this example, the conversion program converts any PCF CKDS entry with the label ATM03.

Example 4

This example shows a global override record specifying that the conversion program bypass all the entries with a certain PCF key type in a PCF CKDS.

REMOTE	Y
--------	---

The conversion program does not convert any of the entries with a key type of REMOTE in the PCF CKDS. After you specify this global override record, you can use explicit override records to convert specific entries with a key type of REMOTE in the PCF CKDS.

Including information in a key entry

An ICSF key entry contains an installation data field that an installation can use to further identify a key. The installation data field contains any information that an installation wants to supply about a key.

PCF records contain an installation data field. The conversion program places the information in the field into the installation data field of the converted entry in the output ICSF CKDS. You can use an override record to specify installation data information for the converted entry in the output ICSF CKDS. The installation data information supplied in the override record overrides any information from the PCF installation data field. If you do not use an override record, the conversion program places any installation data from the PCF entry into the leftmost 8 bytes of the ICSF entry.

These are examples of override records for including key information.

Example 1

This example shows an override record providing the conversion program with installation data information to place in the ICSF CKDS for any converted PCF entry with a certain key label.

ATMKEY12	CONVERTED FROM CUSP1.CKDS 10/01/98
----------	------------------------------------

When the conversion program converts an entry that is labeled ATMKEY12, it places CONVERTED FROM CUSP1.CKDS 10/01/98 in the installation data field for the converted entry.

Example 2

This example shows an override record providing the conversion program with installation data information to place in the ICSF CKDS for any converted PCF entry with a certain key label and key type.

LOCAL890 LOCAL	CONVERTED FROM PCF12.CKDS
----------------	---------------------------

When the conversion program converts an entry that is labeled LOCAL890 with a key type of LOCAL, it places CONVERTED FROM PCF12.CKDS in the installation data field for the converted entry.

Example 3

This example shows a global override record that provides the conversion program with installation data information to place in the ICSF CKDS for all converted entries.

CONVERTED FROM PCF10.CKDS

When the conversion program converts the PCF CKDS, it places CONVERTED FROM PCF10.CKDS in the installation data field. The information is placed into every converted key entry. After you specify this global override record, you can use explicit override records to provide different information for specific entries in the PCF CKDS.

Converting key types

By default, the conversion program converts PCF keys into certain ICSF key types. You can use the override file to override the defaults. For example:

- Instead of automatically converting a PCF local key into a NOCV exporter key-encrypting key, you can convert the local key into an output PIN-encrypting key.
- Instead of automatically converting a PCF remote key into a NOCV importer key-encrypting key, you can convert the remote key into an input PIN-encrypting key.
- Instead of automatically converting a PCF cross key into a NOCV exporter key-encrypting key and a NOCV importer key-encrypting key, you can convert the cross key into an output PIN-encrypting key and an input PIN-encrypting key.

You can use a global override record to convert all keys of a certain type into a type other than the conversion program default key type. Then using an explicit override record, you can specify that the conversion program convert a specific record into a the default key type. For example, you can use a global override record to convert all remote keys into input PIN-encrypting keys, and then use an override record to convert specific remote entries into importer key-encrypting keys.

These are some examples of override records for key type conversion.

Example 1

This example shows an override record specifying that the conversion program convert a local key to an output PIN-encrypting key for any PCF CKDS entry with a certain key label. The override record also provides the conversion program with installation data.

CRLABEL1	LOCAL	OPINENC	OPINENC FOR ATM123
----------	-------	---------	--------------------

When the conversion program converts any PCF entry labeled CRLABEL1 with a key type of local, it converts the key from a local key type to an output PIN-encrypting key type. The program also places OPINENC FOR ATM123 in the installation data field.

If you did not specify this override record, the conversion program would automatically convert the entry from a local key type to an exporter key-encrypting key type.

Example 2

This example shows an override record specifying that the conversion program convert a remote key to an input PIN-encrypting key for any PCF CKDS entry with a certain key label. The override record also provides the conversion program with installation data.

CRLABEL2	REMOTE	IPINENC	IPINENC FOR ATM123
----------	--------	---------	--------------------

When the conversion program converts any PCF CKDS entry labeled CRLABEL2 with a key type of remote, it converts the key from a remote key type to an input PIN-encrypting key type. The program also places IPINENC FOR ATM123 in the installation data field.

If you did not specify this override record, the conversion program would automatically convert the entry from a remote key type to an importer key-encrypting key type.

Example 3

This example shows an override record specifying that the conversion program convert a local key to an exporter key-encrypting key for any PCF CKDS entry with a certain key label. The override record also provides the conversion program with installation data.

LOLABEL1	LOCAL	EXPORTER	EXPORTER CONVERTED FROM CUSP12.CKDS
----------	-------	----------	-------------------------------------

The conversion program automatically converts a local key to an exporter key-encrypting key. You can use this override record if you previously submitted an override record that had the conversion program convert all the local key types to output PIN-encrypting keys. You can use this override record to explicitly convert the key entry that is labeled LOLABEL1 from a local key type to an exporter key-encrypting key type.

With the example override record, when the conversion program converts any PCF entry labelled LOLABEL1 with a key type of local, it converts the key from a local key type to an exporter key-encrypting key type. The program also places EXPORTER CONVERTED FROM CUSP12.CKDS in the installation data field.

Example 4

This example shows an override record specifying that the conversion program convert a remote key to an importer key-encrypting key for any PCF CKDS entry with a certain key label. The override record also provides the conversion program with installation data.

RECKDS12	REMOTE	IMPORTER	IMPORTER CONVERTED FROM CUSP12.CKDS
----------	--------	----------	-------------------------------------

The conversion program automatically converts remote keys to importer key-encrypting keys. You can use this override record if you supplied an override record to convert all the remote key types to input key-encrypting keys. Use this override record to explicitly convert key entries labeled RECKDS12 from remote key types to importer key-encrypting key types.

With the example override record, when the conversion program converts any PCF entry labeled RECKDS12 with a key type of remote, it converts the key from a remote key type to an importer key-encrypting key type. The program also places IMPORTER CONVERTED FROM CUSP12.CKDS in the installation data field.

Example 5

This example shows a global override record specifying that the conversion program convert a local key to an output PIN-encrypting key for any PCF CKDS entry with a key type of local. The override record also provides the conversion program with installation data.

```
LOCAL  OPINENC          OPINENC FROM CUSP.PIN12.CKDS
```

When the conversion program converts any PCF entry with a key type of local, the program converts the key from a local key type to an output PIN-encrypting key type. The program also places OPINENC FROM CUSP.PIN12.CKDS in the installation data field. After you specify this global override record, you can use explicit override records to place different installation data in the ICSF CKDS entries.

Example 6

This example shows a global override record specifying that the conversion program convert a remote key to an input PIN-encrypting key for any PCF CKDS entry with a key type of remote. The override record also provides the conversion program with installation data.

```
REMOTE IPINENC          IPINENC FROM CUSP.PIN12.CKDS
```

When the conversion program converts any CUSP/PCF entry with a key type of remote, it converts the key from a remote key type to an input PIN-encrypting key type. The program also places IPINENC FROM CUSP.PIN12.CKDS in the installation data field for the entry in the ICSF CKDS. After you specify this global override record, you can use explicit override records to place different installation data information in the ICSF CKDS entries.

Running the conversion program

You can run the conversion program only after you initialize the master key and CKDS for ICSF. The CKDS you specify at ICSF startup must be initialized to contain NOCV-enablement keys. For information about defining keys on ICSF, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

If the PCF master key and the ICSF master key are not the same, you must define the PCF master key in the input ICSF CKDS. Define the PCF master key as an importer key-encrypting key in the input ICSF CKDS. You define the key by entering the key through the key entry hardware, or by importing the key using the ICSF key generator utility program. For information about direct key entry through the key entry hardware and the key generator utility program, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

Note: Be careful defining the PCF master key in the input ICSF CKDS, because there is no programmed way to determine its validity.

You run the conversion program by submitting a batch job. On the EXEC statement, specify PGM=CSFCONV. If the PCF master key and ICSF master key are not the same in the PARM= field on the EXEC statement, specify the label of the PCF master key entry in the input ICSF CKDS. If you do not specify the parameter, the conversion program assumes that the PCF master key and ICSF master key are the same.

This example is a JCL that runs the conversion program:

```
//CKDSCONV EXEC PGM=CSFCONV,PARM='CUSPMKEY'  
//CSFVSRC DD DSN=PROD.CUSP.CKDS,DISP=SHR  
//CSFVINP DD DSN=TEST.CSF.CKDS,DISP=SHR  
//CSFVOVR DD DSN=OVERRIDE.DATA,DISP=OLD  
//CSFVNEW DD DSN=MERGED.CSF.CKDS,DISP=OLD  
//CSFVRPT DD SYSOUT=A  
//
```

In the example, CUSPMKEY is the label of the entry in the input ICSF CKDS for the PCF master key in importer key-encrypting key form. All the data sets necessary to run the conversion program are specified using DD statements.

The conversion program uses these data sets:

CSFVSR

The PCF CKDS containing entries that you want to convert into ICSF format and place in the output ICSF CKDS. This is the source CKDS for the conversion. For a description of the PCF CKDS record format, see *OS/VS1 and OS/VS2 MVS Programmed Cryptographic Facility*.

CSFVINP

A disk copy of the input ICSF CKDS. The input CKDS should already contain the header record and the ICSF system keys and can contain other ICSF key entries. If the CKDS does not contain NOCV-enablement keys, the output ICSF CKDS will not contain NOCV-enablement keys. For more information about NOCV-enablement keys, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

Note: The input ICSF CKDS does not have to be the CKDS you specify when you start ICSF.

CSFVOVR

The override file with information specifying how you want the conversion program to process PCF key entries. If no override data is required, this data set is optional. However, you must code a dummy DD statement in the JCL.

This JCL is an example of a dummy DD statement for an override file:

```
//CSFVOVR DD DUMMY,DCB=(RECFM=FB,LRECL=90,BLKSIZE=3600)
```

See [“Using the conversion program override file” on page 231](#) for a description of when and how to use the override file.

CSFVNEW

An empty disk copy of an ICSF CKDS. This is the ICSF CKDS into which the conversion program places key entries. The conversion program places key entries from the input ICSF CKDS and the PCF CKDS into the output ICSF CKDS. The data set must be defined and empty before you run the conversion program.

CSFVRPT

The activity report that the conversion program creates. The report describes any override records and gives a summary of CKDS entries that were affected by the conversion program.



Attention: If a conversion program run ends prematurely, the results of the job are unpredictable. You should not read a CKDS involved in the conversion into storage for use. For a description of the conversion program return codes, see the explanation of message CSFV0026 in [z/OS Cryptographic Services ICSF Messages](#).

When you run the conversion program, the program produces information about the conversion in an activity report. The activity report lists each override entry, the action each override entry applies to the input PCF CKDS, and any error messages. The activity report also lists the data sets that were used in the conversion and a summary of processing. The summary of processing contains totals that apply to CKDS entries in the conversion program job.

Example of a Conversion Initial Activity Report

[Figure 11 on page 239](#) is an example of an activity report with five explicit override records and no global override records.

```

CRYPTOGRAPHIC CONVERSION ACTIVITY REPORT                                DATE: 2001/06/01 (YYYY/MM/DD) TIME: 10:13:09 PAGE: 1
OVERRIDE--> CRLABEL3 LOCAL    OPINENC                               Used in transfers to Main Office.
>>>CSFV0192 TYPE FOR KEY ENTRY CRLABEL3 LOCAL CONVERTED TO OPINENC.
>>>CSFV0232 INSTALLATION DATA FOR KEY ENTRY CRLABEL3 OPINENC SET TO Used in transfers to Main Office

OVERRIDE--> CRLABEL3 REMOTE    IPINENC                               Used in receiving from the Main Office
>>>CSFV0192 TYPE FOR KEY ENTRY CRLABEL3 REMOTE CONVERTED TO IPINENC.
>>>CSFV0232 INSTALLATION DATA FOR KEY ENTRY CRLABEL3 IPINENC SET TO Used in receiving from the Main Office.

OVERRIDE--> KGLABEL1 LOCAL    OPINENC                               Used for sending encrypted PINs
>>>CSFV0292 NO KEY ENTRY FOUND FOR KGLABEL1 LOCAL.

OVERRIDE--> LOLABEL2                                Valid for January 2001
>>>CSFV0232 INSTALLATION DATA FOR KEY ENTRY LOLABEL2 EXPORTER SET TO Valid for January 2001.

OVERRIDE--> ZZZZ1      LOCAL                                Y Eliminate Key from output CKDS
>>>CSFV0382 ADD/CHANGE SPECIFICATIONS IGNORED ON OVERRIDE ENTRY. BYPASS_FLAG VALUE IS "Y".
>>>CSFV0292 NO KEY ENTRY FOUND FOR ZZZZ1 LOCAL.

>>>CSFV0012 CONVERSION PROCESSING COMPLETED. RETURN CODE = 4.

```

```

CRYPTOGRAPHIC CONVERSION ACTIVITY REPORT                                DATE: 2001/06/01 (YYYY/MM/DD) TIME: 10:13:09 PAGE: 2

CKDS DDNAME      Data Set Name
-----
CSFVSR           PROD.CUSP.CKDS
CSFVINP          TEST.CSF.CKDS
CSFVNEW          MERGED.CSF.CKDS

PROCESSING SUMMARY

Source CKDS Entries      Converted Entries      ICSF Entries
-----
LOCAL                     4      * Candidates          16      + Changed Input Entries      2
REMOTE                     4      Bypassed by Overrides    ( 0)      + Unchanged Input Entries    13
CROSS                      4
-----
* TOTAL Source Entries    12      TOTAL Converted Entries    16      + TOTAL ICSF Input Entries   15
                                           + Entries Added from Source   14
                                           + Entries Bypassed by Exit    ( 0)
                                           -----
                                           TOTAL Output ICSF Entries    29

* One Source CKDS CROSS entry converts to two Candidates.
+ Total Converted Entries = Changed Input Entries + Entries Added from Source.

```

Figure 11. Example of a Conversion Initial Activity Report

In the report, the first override record specifies that when the conversion program converts a PCF entry labeled CRLABEL3 with a key type of local, the program should convert the entry into an output PIN-encrypting key. The conversion program also places the information 'Used in transfers to Main Office' in the installation data field of the output ICSF CKDS entry.

The second override record specifies that when the conversion program converts a PCF entry labeled CRLABEL3 with a key type of remote, the program should convert the key into an input PIN-encrypting key. The conversion program places the information 'Used in receiving from the Main Office' in the installation data field of the output ICSF CKDS entry.

The label specified by the third override record does not exist in the PCF CKDS. Therefore, the conversion program ignores this override record.

The fourth override record specifies that when the conversion program converts a PCF entry labelled LOLABEL2, the program should place the information 'Valid for January 2001' in the installation data field of the output ICSF CKDS record.

The label specified by the fifth override record does not exist on the PCF CKDS that the conversion program is converting. Therefore, the conversion program ignores this override record.

The message that the conversion processing has been completed is followed by a return code. Return codes are listed under message CSFV0026 in *z/OS Cryptographic Services ICSF Messages*.

After describing the five override records, the conversion report lists the data sets the conversion program used in the conversion. PROD.CUSP.CKDS is the PCF CKDS that the program converted. TEST.CSF.CKDS is the input ICSF CKDS containing the ICSF entries input during the conversion. MERGED.CSF.CKDS is the output ICSF CKDS where the conversion program placed the converted entries.

Then the activity report lists totals pertaining to the conversion. The PCF CKDS has a total of 12 entries: four with a key type of local, four with a key type of remote, and four with a key type of cross. Because the conversion of each cross key entry results in two ICSF entries, the total ICSF entries that are candidates for conversion from the PCF is 16. None of these candidates was bypassed because of an override record, so 16 PCF entries were converted.

There were 15 entries in the input ICSF CKDS, and two of these entries were updated because they had identical key labels in the PCF CKDS. Fourteen new output ICSF CKDS entries were added from the PCF CKDS. The total number of entries in the output ICSF CKDS is 29. This includes the 15 entries in the input ICSF CKDS and the 14 entries added from the PCF CKDSN. No entries were bypassed because of the conversion program exit.

Example of a Conversion Update Activity Report

Figure 12 on page 240 is an example of an activity report with a global override record that has the conversion program bypass all the entries in the PCF CKDS. Then two override records are used to convert specific entries.

CRYPTOGRAPHIC CONVERSION ACTIVITY REPORT

DATE: 2001/06/01 (YYYY/MM/DD) TIME: 10:13:09 PAGE: 1

OVERRIDE-->

Y

>>>CSFV0172 ALL ENTRIES BYPASSED.

OVERRIDE--> CRLABEL3 LOCAL OPINENC Used in transfers to Main Office

>>>CSFV0222 KEY ENTRY CRLABEL3 LOCAL NOT BYPASSED.

>>>CSFV0192 TYPE FOR KEY ENTRY CRLABEL3 LOCAL CONVERTED TO OPINENC.

>>>CSFV0232 INSTALLATION DATA FOR KEY ENTRY CRLABEL3 OPINENC SET TO Used in transfers to Main Office.

OVERRIDE--> LOLABEL2 Valid for January 2001

>>>CSFV0222 KEY ENTRY LOLABEL2 LOCAL NOT BYPASSED.

>>>CSFV0232 INSTALLATION DATA FOR KEY ENTRY LOLABEL2 EXPORTER SET TO Valid for January 2001.

>>>CSFV0012 CONVERSION PROCESSING COMPLETED. RETURN CODE = 0.

CRYPTOGRAPHIC CONVERSION ACTIVITY REPORT

DATE: 2001/06/01 (YYYY/MM/DD) TIME: 10:13:09 PAGE: 2

CKDS DDNAME Data Set Name

CSFVSRC PROD.PCF.CKDS

CSFVINP INTEST.CSF.CKDS

CSFVNEW NEWTEST.CSF.CKDS

PROCESSING SUMMARY

Source CKDS Entries

Converted Entries

ICSF Entries

LOCAL 4 * Candidates 16 + Changed Input Entries 1

REMOTE 4 Bypassed by Overrides (14) Unchanged Input Entries 27

CROSS 4

* TOTAL Source Entries 12 TOTAL Converted Entries 2 + TOTAL ICSF Input Entries 28

+ Entries Added from Source 1

Entries Bypassed by Exit (0)

TOTAL Output ICSF Entries 29

* One Source CKDS CROSS entry converts to two Candidates.

+ Total Converted Entries = Changed Input Entries + Entries Added from Source.

Figure 12. Example of a Conversion Update Activity Report

The first override record specifies that the conversion program bypass all the entries in the PCF CKDS. The second override record specifies that the conversion program convert a PCF entry labeled CRLABEL3 with a key type of local into an output PIN-encrypting key. This second override record also instructs the conversion program to place the phrase 'Used in transfers to Main Office' in the installation data field of the output ICSF CKDS entry. The third override record specifies that the conversion program convert a PCF entry labeled LOLABEL2 and place 'Valid for January 2001' in the installation data field of the output ICSF CKDS entry.

After describing the three override records, the conversion report lists the data sets the conversion program used in the conversion. PROD.PCF.CKDS is the PCF CKDS that the program converted. INTEST.CSF.CKDS is the input ICSF CKDS that contains the ICSF entries input containing the ICSF entries input during the conversion. NEWTEST.CSF.CKDS is the output ICSF CKDS where the conversion program placed the converted entries.

Then the activity report lists totals pertaining to the conversion. The PCF CKDS has a total of 12 entries: four with a key type of local, four with a key type of remote, and four with a key type of cross. Because the conversion of each cross key entry results in two ICSF entries, the total ICSF records that are candidates for conversion from PCF is 16. Fourteen of those 16 entries were bypassed because of the global override record.

There were 28 entries in the input ICSF CKDS, and one of these entries was updated because it had an identical key label in the PCF CKDS. The total number of entries in the output ICSF CKDS is 29. This includes the 28 entries in the input ICSF CKDS plus the one added from the PCF CKDS. No entries were bypassed because of the conversion program exit.

Appendix A. Diagnosis reference information

This appendix contains Diagnosis, Modification, or Tuning Information.

This appendix contains descriptions of the cryptographic key data set (CKDS), the public key data set (PKDS), PKA key tokens, the Cryptographic Communication Vector Table (CCVT), and Cryptographic Communication Vector Table Extension (CCVE) data areas.

For more information about key tokens, refer to [*z/OS Cryptographic Services ICSF Application Programmer's Guide*](#).

Cryptographic Key Data Set (CKDS) formats

There are four formats of the CKDS:

- The large common record format (KDSRL) which supports all operational symmetric key tokens along with metadata and allows ICSF to track key usage if so configured. This format is referred to as KDSRL format in most places, but may be referred to as KDSR when it need not be differentiated from the original common record format.
- The common record format (KDSR) which supports all operational symmetric key tokens and metadata and allows ICSF to track key usage if so configured.
- The variable-length record format which supports all symmetric key tokens.
- The fixed-length record format which supports fixed-length symmetric key tokens.

Format of the CKDS header record

Table 31. Cryptographic Key Data Set Header Record Format			
Offset (Dec)	Number of Bytes	Field Name	Description
0	72	<i>Constant</i>	The field is set to binary zeros and is not used for the header record.
72	8	<i>Creation date</i>	The date the CKDS was initialized in the format <i>yyyymmdd</i> .
80	8	<i>Creation time</i>	The initial time the CKDS was created in the format <i>hhmmssst</i> .
88	8	<i>Last update date</i>	The most recent date the CKDS was updated, in the format <i>yyyymmdd</i> . This field is no longer updated when a record is updated.
96	8	<i>Last update time</i>	The most recent time the CKDS was updated, in the format <i>hhmmssst</i> . This field is no longer updated when a record is updated.
104	2	<i>Sequence number</i>	Initially zero in binary. Incremented each time the data set is processed. This field is no longer updated.

Table 31. Cryptographic Key Data Set Header Record Format (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
106	2	<i>CKDS header flag bytes</i>	<p>Flag bytes.</p> <p>Bit Meaning When Set On</p> <p>0 The DES master key verification pattern is valid.</p> <p>1 Reserved.</p> <p>2 The AES master key verification pattern is valid.</p> <p>3–7 Reserved.</p> <p>8 Record level authentication is disabled.</p> <p>9 The record format is variable. Set on for either variable length record format or KDSR record format.</p> <p>10 CKDS not completely written, missing records.</p> <p>11–15 Reserved.</p> <p>Note: After the bits are set on, the given values remain constant in ICSF.</p>
108	8	<i>DES master key verification pattern</i>	The system DES master key verification pattern.
116	8		Reserved.
124	8	<i>AES master key verification pattern.</i>	The AES master key verification pattern.
132	4	<i>Record length</i>	Length of the record in bytes. X'00000000' for fixed length record format. X'000000FC' for either variable length record format or KDSR record format.
136	1	<i>Record version</i>	Version number of the CKDS in binary. Set to X'00' for fixed length record format or variable length record format. Set to X'02' or greater for KDSR record format.
137	8	<i>DES MKVP date</i>	The date that the CKDS header DES master key verification pattern was stored in the CKDS in STCKE format, high 8 bytes. Low bit in Byte 5 represents one second.
145	8	<i>AES MKVP date</i>	The date that the CKDS header AES master key verification pattern was stored in the CKDS in STCKE format, high 8 bytes. Low bit in Byte 5 represents one second.

Table 31. Cryptographic Key Data Set Header Record Format (continued)			
Offset (Dec)	Number of Bytes	Field Name	Description
153	1	AES/ DES MKVP date flags	<p>Bit Meaning When Set On</p> <p>0 The DES MKVP date was initialized as part of the initialization or an update of the KDS when the DES KDS MKVP value was initialized.</p> <p>1 The DES MKVP date was changed as part of a reencipher during coordinated change master key or when a reencipher of the KDS was done outside of a coordinated change master key.</p> <p>2-3 Reserved.</p> <p>4 The AES MKVP date was initialized as part of the initialization or an update of the KDS when the DES KDS MKVP value was initialized.</p> <p>5 The AES MKVP date was changed as part of a reencipher during coordinated change master key or when a reencipher of the KDS was done outside of a coordinated change master key.</p> <p>6-7 Reserved.</p>
154	42		Reserved.
196	52	Installation data	Installation data associated with the CKDS record, as supplied by an installation exit.
248	4	Authentication code	The code generated by the authentication process that ensures that the CKDS record has not been modified since the last update. The authentication code is placed in the CKDS header record when the CKDS is initialized. ICSF verifies the CKDS header record authentication code whenever a CKDS is reenciphered, refreshed, or converted from PCF to ICSF format. This field is not used when the record level authentication flag is set in the CKDS header flag bytes field of the CKDS header record.

Format of the fixed-length CKDS record

Table 32. Cryptographic Key Data Set Record Format			
Offset (Dec)	Number of Bytes	Field Name	Description
0	64	Key label	The key label specified by the KGUP control statement or Clear Key Input panel when the record was created. When using KGUP and the callable services, you can specify the label to identify the record. The key label is the first field of the key index.

Table 32. Cryptographic Key Data Set Record Format (continued)			
Offset (Dec)	Number of Bytes	Field Name	Description
64	8	Key type	The type of key the record contains. The master key variant for the key type enciphers the key. A KGUP control statement or Clear Key Input panel specifies the key type when the record is created. The key type is the second field of the key index.
72	8	Creation date	The initial date the CKDS record was created in the format <i>yyyymmdd</i> .
80	8	Creation time	The initial time the CKDS record was created in the format <i>hhmmssst</i> .
88	8	Last update date	The most recent date the CKDS record was updated in the format <i>yyyymmdd</i> .
96	8	Last update time	The most recent time the CKDS record was updated in the format <i>hhmmssst</i> .
104	64	Key token	The internal key token. A key token contains the key value. The value in byte four of the internal key token indicates whether the key is AES or DES. Refer to “AES internal fixed-length key token” on page 282 and “DES fixed-length key token” on page 284 for the format of the internal key token.
168	2	CKDS flag bytes	<p>Flag bytes.</p> <p>Bit Meaning When Set On</p> <p>0 The key within the key token field (offset 104) is a partial key. The key is unusable.</p> <p>1 Reserved.</p> <p>2 CKDS label must be unique.</p> <p>3–7 Reserved.</p>
170	26		Reserved.
196	52	Installation data	Installation data associated with the CKDS record as supplied by an installation exit.
248	4	Authentication code	The code generated by the authentication process that ensures the CKDS record has not been modified since the last update. The authentication code is placed in the CKDS record when the record is created. When you refresh, reencipher, or convert a CKDS, ICSF verifies each CKDS record as ICSF performs the action. This field is not used when the record level authentication flag is set in the CKDS header flag bytes field of the CKDS header record.

Format of the variable-length CKDS record

The following table presents the format of each variable-length data set record.

Table 33. Variable-Length Cryptographic Key Data Set Record Format			
Offset (Dec)	Number of Bytes	Field Name	Description
0	64	Key label	The label or name of this CKDS record. The key label is the first field of the key index.
64	8	Key type	The type of key the record contains. The key type is the second field of the key index.
72	8	Creation date	The initial date the CKDS record was created in the format <i>yyyymmdd</i> .
80	8	Creation time	The initial time the CKDS record was created in the format <i>hhmmssst</i> .
88	8	Last update date	The most recent date the CKDS record was updated in the format <i>yyyymmdd</i> .
96	8	Last update time	The most recent time the CKDS record was updated in the format <i>hhmmssst</i> .
104	4	Record length	Length of the entire record including the key token.
108	60		Reserved.
168	2	CKDS flag bytes	Flag bytes. Bit Meaning When Set On 0 The key within the key token field is a partial key. The key is unusable. 1 Reserved. 2 CKDS label must be unique. 3 The record format is variable — always 1 4–7 Reserved.
170	26		Reserved.
196	52	Installation data	
248	20	Authentication code	The record authentication code.
268	variable	Key token	The key token.

Format of KDSR CKDS record

See “Common record format (KDSR)” on page 279 for more information on this CKDS record.

Public Key Data Set (PKDS) format

The PKDS record includes the PKDS header and the PKA key token. These tables show the format of each of these records.

Format of the PKDS header record

Table 34. Public Key Data Set Header Record Format			
Offset (Dec)	Number of Bytes	Field Name	Description
0	64	PKHVKEY	VSAM key of the PKDS header.
64	8		Reserved.
72	8	PKHCRDTE	The date the PKDS was created in the format <i>yyyymmdd</i> .
80	8	PKHCRTIM	The initial time the PKDS was created in the format <i>hhmmssst</i> .
88	8	PKHUPDTE	The most recent date the PKDS header was updated, in the format <i>yyyymmdd</i> . This field is no longer updated when a record is updated.
96	8	PKHUPTIM	The most recent time the PKDS header was updated, in the format <i>hhmmssst</i> . This field is no longer updated when a record is updated.
104	4	PKHRLN	Length of the PKDS header entry.
108	8	PKHKMKHP PKHECCDT	The date that the PKDS header ECC master key verification pattern was stored in the PKDS in STCKE format, high 8 bytes. Low bit in Byte 5 represents one second.
116	8		Reserved.
124	16	PKHRSAVP	The verification pattern of the RSA MK.
140	8	PKHEMKVP	The verification pattern of the ECC MK.
148	8	PKHRSADT	The date that the PKDS header RSA master key verification pattern was stored in the PKDS in STCKE format, high 8 bytes. Low bit in Byte 5 represents one second.

Table 34. Public Key Data Set Header Record Format (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
156	1	PKHMKDTF	<p>ECC/RSA MKVP date flags.</p> <p>Bit Meaning When Set On</p> <p>0 The ECC MKVP date was initialized as part of the initialization or an update of the KDS when the DES KDS MKVP value was initialized.</p> <p>1 The ECC MKVP date was changed as part of a reencipher during coordinated change master key or when a reencipher of the KDS was done outside of a coordinated change master key.</p> <p>2-3 Reserved.</p> <p>4 The RSA MKVP date was initialized as part of the initialization or an update of the KDS when the RSA KDS MKVP value was initialized.</p> <p>5 The ECC MKVP date was changed as part of a coordinated change master key or a reencipher of the KDS when the ECC KDS MKVP value was changed.</p> <p>6-7 Reserved.</p>
157	1		Reserved.
158	1	PKHVER	<p>Version number of the PKDS in binary.</p> <ul style="list-style-type: none"> Set to X'00' for PKDS record format. Set to X'02' or greater for KDSR record format.
159	1		<p>Flag bytes.</p> <p>Bit Meaning When Set On</p> <p>0 PKDS not completely written, missing records.</p> <p>1-7 Reserved.</p>
160	20	PKHAUTH	PKDS header authentication code.

Format of the PKDS record

Table 35. Public Key Data Set Record Format

Offset (Dec)	Number of Bytes	Field Name	Description
0	64	PKDLABEL	Label or name of this PKDS entry.

Table 35. Public Key Data Set Record Format (continued)			
Offset (Dec)	Number of Bytes	Field Name	Description
64	8		Reserved.
72	8	PKDCRDTE	The date this PKDS record was created in the format <i>yyyymmdd</i> .
80	8	PKDCRTIM	The initial time this PKDS record was created in the format <i>hhmmssst</i> .
88	8	PKDUPDTE	The most recent date this PKDS record was updated, in the format <i>yyyymmdd</i> .
96	8	PKDUPTIM	The most recent time this PKDS record was updated, in the format <i>hhmmssst</i> .
104	4	PKDRLEN	Length of the entire PKDS record entry.
108	52	PKDUDATA	User data.
160	20	PKDAUTH	The entry authentication code.
180	1868	PKDTOKEN	The public or private key token.

Token data set (TKDS) format

A z/OS PKCS #11 token represents a virtual cryptographic device, and can contain multiple objects. The token data set (TKDS) contains definitions of z/OS PKCS #11 tokens and token objects.

The token data set includes a header record and records for each of the individual z/OS PKCS #11 tokens and token objects. Each object associated with a particular z/OS PKCS #11 token has the token's name in its handle. The records are variable length records, and contain a length field specifying the total length of the record.

Format of the header record of the token data set

There is one header record for the token data set.

Table 36. Format of the header record of the token data set		
Offset (decimal)	Length of field (bytes)	Description
0	72	VSAM key of the TKDS header. Bytes 0-39: Binary zeros. Bytes 40-43: EBCDIC 'THDR'. Bytes 44-71: Binary zeros.
72	8	The date that the TKDS header P11 master key verification pattern was stored in the TKDS in STCKE format, high 8 bytes. Low bit in Byte 5 represents one second.

Table 36. Format of the header record of the token data set (continued)

Offset (decimal)	Length of field (bytes)	Description
80	8	The date that the TKDS was created, in the format <i>yyyymmdd</i> .
88	8	The time that the TKDS was created, in the format <i>hhmmssst</i> .
96	8	The most recent date that the TKDS header was updated, in the format <i>yyyymmdd</i> . This field is no longer updated when a record is updated.
104	8	The most recent time that the TKDS header was updated, in the format <i>hhmmssst</i> . This field is no longer updated when a record is updated.
112	4	Length of the TKDS header record.
116	16	P11 MKVP.
132	16	Reserved.
148	1	<p>P11 MKVP date flags.</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0</p> <p>The P11 MKVP date was initialized as part of the initialization or an update of the KDS when the P11 KDS MKVP value was initialized.</p> <p>1</p> <p>The P11 MKVP date was changed as part of a reencipher during coordinated change master key or when a reencipher of the KDS was done outside of a coordinated change master key.</p> <p>3-7</p> <p>Reserved.</p>
149	5	Reserved.
154	1	<p>Version number of the TKDS in binary.</p> <ul style="list-style-type: none"> • Set to X'00' for fixed length record format or variable length record format. • Set to X'02' or greater for KDSR record.
155	1	<p>Flag bytes.</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0</p> <p>TKDS not completely written, missing records.</p> <p>1-7</p> <p>Reserved.</p>

Format of the token and object records

Each z/OS PKCS #11 token record and token object record begins with the same 188 bytes of data. The remainder of the record is specific to the token or object.

Common section of the token and object records

Every record in the token data set, with the exception of the header record, begins with these 188 bytes of data.

Table 37. Format of the common section of the token and object records		
Offset (decimal)	Length of field (bytes)	Description
0	72	Handle of token or object Bytes 0-31: Token name Bytes 32-39: Sequence number Byte 40: Character "T" for clear token object Character "Y" for secure token object Bytes 41-43 Blank characters Bytes 44-71: Binary zeros
72	8	Reserved for IBM's use
80	8	The date that this record was created, in the format <i>yyyymmdd</i>
88	8	The time that this record was created, in the format <i>hhmmssst</i>
96	8	The most recent date that this record was updated, in the format <i>yyyymmdd</i>
104	8	The most recent time that this record was updated, in the format <i>hhmmssst</i>
112	4	Length of the entire TKDS record entry
116	20	Reserved for IBM's use
136	52	User data
188	variable	The TKDS token or object (see mappings)

Format of the token-specific section of the token record

Each z/OS PKCS #11 token record begins with the 188 bytes. The remainder of the record contains the contents of the token. The mapping of the record shows the data beginning at offset 0, which is its offset into the token-specific portion of the record; however, that portion of the record is at an offset of 188 into the entire record.

Table 38. Format of the unique section of the token record		
Offset (decimal) 188 +	Length of field (bytes)	Description
0	4	Eye catcher for token: "TOKN"
4	2	Version number of structure: EBCDIC '00'
6	2	Length of structure in bytes

Table 38. Format of the unique section of the token record (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
8	4	Reserved for IBM's use. Must be zeros.
12	8	Last assigned sequence number
20	32	Manufacturer identification
52	16	Model
68	16	Serial number
84	8	Date of the most recent update to this token, expressed as Coordinated Universal Time (UTC) in the format <i>yyyymmdd</i> . This includes any update to token information or to a token object.
92	8	Time of the most recent update to this token, expressed as Coordinated Universal Time (UTC) in the format <i>hhmmssst</i> . This includes any update to token information or to a token object.
100	44	Reserved for IBM's use
144		End of token

Format of the object-specific sections of the token object records

The following classes of objects can be associated with a z/OS PKCS #11 token:

- Certificate
- Public key
- Private key
- Secret key
- Data objects
- Domain parameters

The token object record for each begins with the common section described “Common section of the token and object records” on page 252, followed by a section specific to the class of object. Each of the object-specific sections begins with a 12-byte header record, followed by a variable-length section. Each 12-byte header contains a 4-byte flag field that has the same mapping for all classes of objects.

This 4-byte flag field occurs in the object header section of each token object record.

Table 39. Format of the token object flags

Flag bytes	Field name	Description
Flag byte 1		
Bit 0	OBJ_IS_TOKOBJ	When on, the object is a token object. When off, the object is a session object.
Bit 1	OBJ_IS_PRIVOBJ	When on, the object is a private object. When off, the object is a public object.
Bit 2	OBJ_IS_MODOBJ	When on, the object is modifiable.
Bit 3	KEY_DERIVE	When on, the key supports key derivation.

Table 39. Format of the token object flags (continued)

Flag bytes	Field name	Description
Bit 4	KEY_LOCAL	When on, the key was generated locally.
Bit 5	KEY_ENCRYPT	When on, the key supports encryption.
Bit 6	KEY_DECRYPT	When on, the key supports decryption.
Bit 7	KEY_VERIFYA	When on, the key supports verification where the signature is an appendix to the data.
Flag byte 2		
Bit 0	KEY_VERIFYR	When on, the key supports verification where the data is recovered from the signature
Bit 1	KEY_SIGA	When on, the key supports signatures where the signature is an appendix to the data.
Bit 2	KEY_SIGR	When on, the key supports signatures where the data is recovered from the signature.
Bit 3	KEY_WRAP	When on, the key supports wrapping.
Bit 4	KEY_UNWRAP	When on, the key supports unwrapping.
Bit 5	KEY_EXTRACT	When on, the key is extractable.
Bit 6	KEY_IS_SENSITIVE	When on, the key is sensitive.
Bit 7	KEY_IS_ALWAYS_SENSITIVE	When on, the SENSITIVE attribute (KEY_IS_SENSITIVE) is always true.
Flag byte 3		
Bit 0	KEY_NEVER_EXTRACT	When on, the EXTRACTABLE attribute (KEY_EXTRACT) is never true. When off, the EXTRACTABLE attribute (KEY_EXTRACT) can be true.
Bit 1	OBJ_IS_TRUSTED	When on, the certificate can be trusted for the application for which it was created.
Bit 2	CERT_IS_DEFAULT	When on, this is the default certificate.
Bit 3	FIPS140	When on, key is only to be used in a FIPS 140-2 compliant manner.
Bit 4	KEY_IS_SECURE	When on, key is a secure PKCS #11 key.
Bit 5	KEY_ATTRBOUND	When on, key is attribute bound.
Bit 6	WRAP_WITH_TRUSTED	When on, key may only be wrapped with another key marked OBJ_IS_TRUSTED
Bit 7	KEY_IS_ALWAYS_SECURE	When on, KEY_IS_SECURE is always true.
Flag byte 4		
Bits 0-7		Reserved for IBM's use.

Table 40. Format of the token certificate object

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for certificate object: "CERT"
4	2	Version: EBCDIC '00'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 39 on page 253)
Object type-specific section		
12	4	TYPE attribute: X'00000000': CKC_X_509
16	4	Certificate category 0 Undefined 1 Token user 2 Certificate authority 3 Other entity
20	8	Reserved for IBM's use
28	32	Reserved for IBM's use
60	2	Length of SUBJECT attribute in bytes (<i>aa</i>)
62	2	Length of ID attribute in bytes (<i>bb</i>)
64	2	Length of ISSUER attribute in bytes (<i>cc</i>)
66	2	Length of SERIAL_NUMBER attribute in bytes (<i>dd</i>)
68	2	Length of VALUE attribute in bytes (<i>ee</i>)
70	2	Length of LABEL attribute in bytes (<i>ff</i>)
72	2	Length of APPLICATION attribute in bytes (<i>gg</i>)
74	22	Reserved for IBM's use
96	4	Offset of SUBJECT attribute in bytes
100	4	Offset of ID attribute in bytes
104	4	Offset of ISSUER attribute in bytes
108	4	Offset of SERIAL_NUMBER attribute in bytes
112	4	Offset of VALUE attribute in bytes
116	4	Offset of LABEL attribute in bytes
120	4	Offset of APPLICATION attribute in bytes
124	44	Reserved for IBM's use

Table 40. Format of the token certificate object (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
168	$aa + bb + cc + dd + ee + ff + gg$	Certificate attributes (variable length)
168 + $aa + bb + cc + dd + ee + ff + gg$		End of certificate object

Table 41. Format of the token public key object (Version 0)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for public key object: "PUBK"
4	2	Version: EBCDIC '00'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 39 on page 253)
Object type-specific section		
12	4	TYPE attribute: CKK_RSA
16	8	Start date for the key, in the format $yyyymmdd$
24	8	End date for the key, in the format $yyyymmdd$
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	36	Reserved
72	4	Length in bits of modulus n
76	256	Modulus n
332	256	Reserved
588	256	Public exponent e
844	256	Reserved
1100	2	Length of SUBJECT attribute in bytes (aa)
1102	2	Length of ID attribute in bytes (bb)
1104	2	Length of LABEL attribute in bytes (cc)
1106	2	Length of APPLICATION attribute in bytes (dd)
1108	20	Reserved
1128	4	Offset of SUBJECT attribute in bytes
1132	4	Offset of ID attribute in bytes

Table 41. Format of the token public key object (Version 0) (continued)		
Offset (decimal) 188 +	Length of field (bytes)	Description
1136	4	Offset of LABEL attribute in bytes
1140	4	Offset of APPLICATION attribute in bytes
1144	40	Reserved
1184	$aa+bb+cc+dd$	Public key attributes (variable length)
1184 + $aa+bb+cc+dd$		End of public key object

Table 42. Format of the token public key object (Version 1)		
Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for public key object: "PUBK"
4	2	Version: EBCDIC '01'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 39 on page 253)
Object type-specific section		
12	4	TYPE attribute: CKK_RSA, CKK_DSA, CKK_EC, or CKK_DH
16	8	Start date for the key, in the format $yyyymmdd$
24	8	End date for the key, in the format $yyyymmdd$
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	36	Reserved
Algorithm-specific section (RSA)		
72	4	Length in bits of modulus n
76	512	Modulus n
588	512	Public exponent e
Algorithm-specific section (DSA)		
72	4	Length in bits of prime p
76	128	Reserved
204	128	Prime p
332	128	Reserved
460	128	Base g

Table 42. Format of the token public key object (Version 1) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
588	128	Reserved
716	128	Value y
844	20	Reserved
864	20	Subprime q
884	216	Reserved
Algorithm-specific section (DH)		
72	4	Length in bits of prime p
76	256	Prime p
332	256	Base g
588	256	Value y
844	256	Reserved
Algorithm-specific section (EC)		
72	4	EC params curve constant – x'00000001' secp192r1 - { 1 2 840 10045 3 1 1 } x'00000002' secp224r1 - { 1 3 132 0 33 } x'00000003' secp256r1 - { 1 2 840 10045 3 1 7 } x'00000004' secp384r1 - { 1 3 132 0 34 } x'00000005' secp521r1 - { 1 3 132 0 35 } x'00000006' brainpoolP160r1 - { 1 3 36 3 3 2 8 1 1 1 } x'00000007' brainpoolP192r1 - { 1 3 36 3 3 2 8 1 1 3 } x'00000008' brainpoolP224r1 - { 1 3 36 3 3 2 8 1 1 5 } x'00000009' brainpoolP256r1 - { 1 3 36 3 3 2 8 1 1 7 } x'0000000A' brainpoolP320r1 - { 1 3 36 3 3 2 8 1 1 9 } x'0000000B' brainpoolP384r1 - { 1 3 36 3 3 2 8 1 1 11 } x'0000000C' brainpoolP512r1 - { 1 3 36 3 3 2 8 1 1 13 }
76	128	Reserved
204	136	EC point Q (DER encoded)
340	760	Reserved
Variable length attribute section		

Table 42. Format of the token public key object (Version 1) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
1100	2	Length of SUBJECT attribute in bytes (<i>aa</i>)
1102	2	Length of ID attribute in bytes (<i>bb</i>)
1104	2	Length of LABEL attribute in bytes (<i>cc</i>)
1106	2	Length of APPLICATION attribute in bytes (<i>dd</i>)
1108	20	Reserved
1128	4	Offset of SUBJECT attribute in bytes
1132	4	Offset of ID attribute in bytes
1136	4	Offset of LABEL attribute in bytes
1140	4	Offset of APPLICATION attribute in bytes
1144	40	Reserved
1184	<i>aa+bb+cc+dd</i>	Public key attributes (variable length)
1184 + <i>aa+bb+cc+dd</i>		End of public key object

Table 43. Format of the token public key object (Version 2)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for public key object: "PUBK"
4	2	Version: EBCDIC '02'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 39 on page 253)
Object type-specific section		
12	4	TYPE attribute: CKK_RSA, CKK_DSA, CKK_EC, or CKK_DH
16	8	Start date for the key, in the format <i>yyyymmdd</i>
24	8	End date for the key, in the format <i>yyyymmdd</i>
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	36	Reserved
Algorithm-specific section (RSA)		
72	4	Length in bits of modulus <i>n</i>
76	512	Modulus <i>n</i>

Table 43. Format of the token public key object (Version 2) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
588	512	Public exponent e
Algorithm-specific section (DSA)		
72	4	Length in bits of prime p
76	256	Prime p
332	256	Base g
588	256	Value y
844	8	Reserved
852	32	Subprime q
884	216	Reserved
Algorithm-specific section (DH)		
72	4	Length in bits of prime p
76	256	Prime p
332	256	Base g
588	256	Value y
844	256	Reserved
Algorithm-specific section (EC)		
72	4	EC params curve constant – x'00000001' secp192r1 - { 1 2 840 10045 3 1 1 } x'00000002' secp224r1 - { 1 3 132 0 33 } x'00000003' secp256r1 - { 1 2 840 10045 3 1 7 } x'00000004' secp384r1 - { 1 3 132 0 34 } x'00000005' secp521r1 - { 1 3 132 0 35 } x'00000006' brainpoolP160r1 - { 1 3 36 3 3 2 8 1 1 1 } x'00000007' brainpoolP192r1 - { 1 3 36 3 3 2 8 1 1 3 } x'00000008' brainpoolP224r1 - { 1 3 36 3 3 2 8 1 1 5 } x'00000009' brainpoolP256r1 - { 1 3 36 3 3 2 8 1 1 7 } x'0000000A' brainpoolP320r1 - { 1 3 36 3 3 2 8 1 1 9 } x'0000000B' brainpoolP384r1 - { 1 3 36 3 3 2 8 1 1 11 } x'0000000C' brainpoolP512r1 - { 1 3 36 3 3 2 8 1 1 13 }

Table 43. Format of the token public key object (Version 2) (continued)		
Offset (decimal) 188 +	Length of field (bytes)	Description
76	128	Reserved
204	136	EC point Q (DER encoded)
340	760	Reserved
Variable length attribute section		
1100	2	Length of SUBJECT attribute in bytes (<i>aa</i>)
1102	2	Length of ID attribute in bytes (<i>bb</i>)
1104	2	Length of LABEL attribute in bytes (<i>cc</i>)
1106	2	Length of APPLICATION attribute in bytes (<i>dd</i>)
1108	20	Reserved
1128	4	Offset of SUBJECT attribute in bytes
1132	4	Offset of ID attribute in bytes
1136	4	Offset of LABEL attribute in bytes
1140	4	Offset of APPLICATION attribute in bytes
1144	40	Reserved
1184	<i>aa+bb+cc+dd</i>	Public key attributes (variable length)
1184 + <i>aa+bb+cc+dd</i>		End of public key object

Table 44. Format of the token public key object (Version 3)		
Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for public key object: "PUBK"
4	2	Version: EBCDIC '03'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 39 on page 253)
Object type-specific section		
12	4	TYPE attribute: CKK_RSA, CKK_DSA, CKK_EC, or CKK_DH
16	8	Start date for the key, in the format <i>yyyymmdd</i>
24	8	End date for the key, in the format <i>yyyymmdd</i>
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION

Table 44. Format of the token public key object (Version 3) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
36	2	Reserved
38	2	Length of secure key material in bytes (ee)
40	4	Offset to secure key material in bytes
44	28	Reserved
Algorithm-specific section (RSA)		
72	4	Length in bits of modulus n
76	512	Modulus n
588	512	Public exponent e
Algorithm-specific section (DSA)		
72	4	Length in bits of prime p
76	256	Prime p
332	256	Base g
588	256	Value y
844	8	Reserved
852	32	Subprime q
884	216	Reserved
Algorithm-specific section (DH)		
72	4	Length in bits of prime p
76	256	Prime p
332	256	Base g
588	256	Value y
844	256	Reserved
Algorithm-specific section (EC)		

Table 44. Format of the token public key object (Version 3) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
72	4	EC params curve constant – x'00000001' secp192r1 - { 1 2 840 10045 3 1 1 } x'00000002' secp224r1 - { 1 3 132 0 33 } x'00000003' secp256r1 - { 1 2 840 10045 3 1 7 } x'00000004' secp384r1 - { 1 3 132 0 34 } x'00000005' secp521r1 - { 1 3 132 0 35 } x'00000006' brainpoolP160r1 - { 1 3 36 3 3 2 8 1 1 1 } x'00000007' brainpoolP192r1 - { 1 3 36 3 3 2 8 1 1 3 } x'00000008' brainpoolP224r1 - { 1 3 36 3 3 2 8 1 1 5 } x'00000009' brainpoolP256r1 - { 1 3 36 3 3 2 8 1 1 7 } x'0000000A' brainpoolP320r1 - { 1 3 36 3 3 2 8 1 1 9 } x'0000000B' brainpoolP384r1 - { 1 3 36 3 3 2 8 1 1 11 } x'0000000C' brainpoolP512r1 - { 1 3 36 3 3 2 8 1 1 13 }
76	128	Reserved
204	136	EC point Q (DER encoded)
340	760	Reserved
Variable length attribute section		
1100	2	Length of SUBJECT attribute in bytes (<i>aa</i>)
1102	2	Length of ID attribute in bytes (<i>bb</i>)
1104	2	Length of LABEL attribute in bytes (<i>cc</i>)
1106	2	Length of APPLICATION attribute in bytes (<i>dd</i>)
1108	20	Reserved
1128	4	Offset of SUBJECT attribute in bytes
1132	4	Offset of ID attribute in bytes
1136	4	Offset of LABEL attribute in bytes
1140	4	Offset of APPLICATION attribute in bytes
1144	40	Reserved
1184	<i>aa+bb+cc+dd+ee</i>	Public key attributes (variable length)

Table 44. Format of the token public key object (Version 3) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
1184 +aa+bb+cc+dd+ee		End of public key object

Table 45. Format of the token private key object (Version 0)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for private key object: "PRIV"
4	2	Version: EBCDIC '00'
6	2	Length of object (in bytes)
8	4	Flags (see Table 39 on page 253)
Object type-specific section		
12	4	Type attribute: CKK_RSA
16	8	Start date for the key (in the format <i>yyyymmdd</i>)
24	8	End date for the key (in the format <i>yyyymmdd</i>)
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	36	Reserved
72	4	Length in bits of modulus n
76	256	Modulus: modulus n
332	256	Reserved
588	256	Public exponent e
844	256	Reserved
1100	32	Reserved
1132	256	Private exponent d
1388	256	Reserved
1644	136	Prime p
1780	128	Reserved
1908	128	Prime q
2036	128	Reserved
2172	136	Private exponent d modulo p-1
2300	128	Reserved

Table 45. Format of the token private key object (Version 0) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
2428	128	Private exponent d modulo q-1
2556	128	Reserved
2684	136	CRT coefficient q-1 mod p
2820	128	Reserved
2948	2	Length of SUBJECT attribute in bytes (xx)
2950	2	Length of ID attribute in bytes (yy)
2952	2	Length of LABEL attribute in bytes (zz)
2954	2	Length of APPLICATION attribute in bytes (ww)
2956	20	Reserved
2976	4	Offset of SUBJECT attribute in bytes
2980	4	Offset of ID attribute in bytes
2984	4	Offset of LABEL attribute in bytes
2988	4	Offset of APPLICATION attribute in bytes
2992	40	Reserved
3032	xx+yy+zz+ww	Private key attributes (variable length)
3032 +xx+yy+zz+ww		End of private key object

Table 46. Format of the token private key object (Version 1)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for private key object: "PRIV"
4	2	Version: EBCDIC '01'
6	2	Length of object (in bytes)
8	4	Flags (see Table 39 on page 253)
Object type-specific section		
12	4	Type attribute: CKK_RSA, CKK_DSA, CKK_EC, or CKK_DH
16	8	Start date for the key (in the format <i>yyyymmdd</i>)
24	8	End date for the key (in the format <i>yyyymmdd</i>)
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION

Table 46. Format of the token private key object (Version 1) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
36	36	Reserved
Algorithm-specific section (RSA)		
72	4	Length in bits of modulus n
76	512	Modulus: modulus n
588	512	Public exponent e
1100	32	Reserved
1132	512	Private exponent d
1644	264	Prime p
1908	256	Prime q
2164	264	Private exponent d modulo $p-1$
2428	256	Private exponent d modulo $q-1$
2684	264	CRT coefficient $q-1 \bmod p$
Algorithm-specific section (DSA)		
72	4	Length in bits of prime p
76	128	Reserved
204	128	Prime p
332	128	Reserved
460	128	Base g
588	236	Reserved
824	20	Value x
844	20	Reserved
864	20	Subprime q
884	2064	Reserved
Algorithm-specific section (DH)		
72	4	Length in bits of prime p
76	256	Prime p
332	256	Base g
588	236	Reserved
824	20	Value x
844	2104	Reserved
Algorithm-specific section (EC)		

Table 46. Format of the token private key object (Version 1) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
72	4	EC params curve constant – x'00000001' secp192r1 - { 1 2 840 10045 3 1 1 } x'00000002' secp224r1 - { 1 3 132 0 33 } x'00000003' secp256r1 - { 1 2 840 10045 3 1 7 } x'00000004' secp384r1 - { 1 3 132 0 34 } x'00000005' secp521r1 - { 1 3 132 0 35 } x'00000006' brainpoolP160r1 - { 1 3 36 3 3 2 8 1 1 1 } x'00000007' brainpoolP192r1 - { 1 3 36 3 3 2 8 1 1 3 } x'00000008' brainpoolP224r1 - { 1 3 36 3 3 2 8 1 1 5 } x'00000009' brainpoolP256r1 - { 1 3 36 3 3 2 8 1 1 7 } x'0000000A' brainpoolP320r1 - { 1 3 36 3 3 2 8 1 1 9 } x'0000000B' brainpoolP384r1 - { 1 3 36 3 3 2 8 1 1 11 } x'0000000C' brainpoolP512r1 - { 1 3 36 3 3 2 8 1 1 13 }
76	64	Reserved
140	66	Value <i>d</i>
206	2742	Reserved
Variable length attribute section		
2948	2	Length of SUBJECT attribute in bytes (xx)
2950	2	Length of ID attribute in bytes (yy)
2952	2	Length of LABEL attribute in bytes (zz)
2954	2	Length of APPLICATION attribute in bytes (ww)
2956	20	Reserved
2976	4	Offset of SUBJECT attribute in bytes
2980	4	Offset of ID attribute in bytes
2984	4	Offset of LABEL attribute in bytes
2988	4	Offset of APPLICATION attribute in bytes
2992	40	Reserved
3032	xx+yy+zz+ww	Private key attributes (variable length)

Table 46. Format of the token private key object (Version 1) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
3032 +xx+yy+zz+ww		End of private key object

Table 47. Format of the token private key object (Version 2)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for private key object: "PRIV"
4	2	Version: EBCDIC '02'
6	2	Length of object (in bytes)
8	4	Flags (see Table 39 on page 253)
Object type-specific section		
12	4	Type attribute: CKK_RSA, CKK_DSA, CKK_EC, or CKK_DH
16	8	Start date for the key (in the format <i>yyyymmdd</i>)
24	8	End date for the key (in the format <i>yyyymmdd</i>)
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	36	Reserved
Algorithm-specific section (RSA)		
72	4	Length in bits of modulus <i>n</i>
76	512	Modulus: modulus <i>n</i>
588	512	Public exponent <i>e</i>
1100	32	Reserved
1132	512	Private exponent <i>d</i>
1644	264	Prime <i>p</i>
1908	256	Prime <i>q</i>
2164	264	Private exponent <i>d</i> modulo <i>p</i> -1
2428	256	Private exponent <i>d</i> modulo <i>q</i> -1
2684	264	CRT coefficient <i>q</i> -1 mod <i>p</i>
Algorithm-specific section (DSA)		
72	4	Length in bits of prime <i>p</i>
76	256	Prime <i>p</i>

Table 47. Format of the token private key object (Version 2) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
332	256	Base g
588	224	Reserved
812	32	Value x
844	8	Reserved
852	32	Subprime q
884	2064	Reserved
Algorithm-specific section (DH)		
72	4	Length in bits of prime p
76	256	Prime p
332	256	Base g
588	256	Value x
844	4	Length in bits of value x
848	2100	Reserved
Algorithm-specific section (EC)		
72	4	EC params curve constant – x'00000001' secp192r1 - { 1 2 840 10045 3 1 1 } x'00000002' secp224r1 - { 1 3 132 0 33 } x'00000003' secp256r1 - { 1 2 840 10045 3 1 7 } x'00000004' secp384r1 - { 1 3 132 0 34 } x'00000005' secp521r1 - { 1 3 132 0 35 } x'00000006' brainpoolP160r1 - { 1 3 36 3 3 2 8 1 1 1 } x'00000007' brainpoolP192r1 - { 1 3 36 3 3 2 8 1 1 3 } x'00000008' brainpoolP224r1 - { 1 3 36 3 3 2 8 1 1 5 } x'00000009' brainpoolP256r1 - { 1 3 36 3 3 2 8 1 1 7 } x'0000000A' brainpoolP320r1 - { 1 3 36 3 3 2 8 1 1 9 } x'0000000B' brainpoolP384r1 - { 1 3 36 3 3 2 8 1 1 11 } x'0000000C' brainpoolP512r1 - { 1 3 36 3 3 2 8 1 1 13 }
76	64	Reserved
140	66	Value d

Table 47. Format of the token private key object (Version 2) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
206	2742	Reserved
Variable length attribute section		
2948	2	Length of SUBJECT attribute in bytes (xx)
2950	2	Length of ID attribute in bytes (yy)
2952	2	Length of LABEL attribute in bytes (zz)
2954	2	Length of APPLICATION attribute in bytes (ww)
2956	20	Reserved
2976	4	Offset of SUBJECT attribute in bytes
2980	4	Offset of ID attribute in bytes
2984	4	Offset of LABEL attribute in bytes
2988	4	Offset of APPLICATION attribute in bytes
2992	40	Reserved
3032	xx+yy+zz+ww	Private key attributes (variable length)
3032 +xx+yy+zz+ww+ee		End of private key object

Table 48. Format of the token private key object (Version 3)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for private key object: "PRIV"
4	2	Version: EBCDIC '03'
6	2	Length of object (in bytes)
8	4	Flags (see Table 39 on page 253)
Object type-specific section		
12	4	Type attribute: CKK_RSA, CKK_DSA, CKK_EC, or CKK_DH
16	8	Start date for the key (in the format yyyyymmdd)
24	8	End date for the key (in the format yyyyymmdd)
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	2	Reserved
38	2	Length of secure key material (ee)

Table 48. Format of the token private key object (Version 3) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
40	4	Offset to secure key material in bytes
44	28	Reserved
Algorithm-specific section (RSA)		
72	4	Length in bits of modulus n
76	512	Modulus: modulus n
588	512	Public exponent e
1100	32	Reserved
1132	512	Private exponent d
1644	264	Prime p
1908	256	Prime q
2164	264	Private exponent d modulo $p-1$
2428	256	Private exponent d modulo $q-1$
2684	264	CRT coefficient $q-1 \bmod p$
Algorithm-specific section (DSA)		
72	4	Length in bits of prime p
76	256	Prime p
332	256	Base g
588	224	Reserved
812	32	Value x
844	8	Reserved
852	32	Subprime q
884	2064	Reserved
Algorithm-specific section (DH)		
72	4	Length in bits of prime p
76	256	Prime p
332	256	Base g
588	256	Value x
844	4	Length in bits of value x
848	2100	Reserved
Algorithm-specific section (EC)		

Table 48. Format of the token private key object (Version 3) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
72	4	EC params curve constant – x'00000001' secp192r1 - { 1 2 840 10045 3 1 1 } x'00000002' secp224r1 - { 1 3 132 0 33 } x'00000003' secp256r1 - { 1 2 840 10045 3 1 7 } x'00000004' secp384r1 - { 1 3 132 0 34 } x'00000005' secp521r1 - { 1 3 132 0 35 } x'00000006' brainpoolP160r1 - { 1 3 36 3 3 2 8 1 1 1 } x'00000007' brainpoolP192r1 - { 1 3 36 3 3 2 8 1 1 3 } x'00000008' brainpoolP224r1 - { 1 3 36 3 3 2 8 1 1 5 } x'00000009' brainpoolP256r1 - { 1 3 36 3 3 2 8 1 1 7 } x'0000000A' brainpoolP320r1 - { 1 3 36 3 3 2 8 1 1 9 } x'0000000B' brainpoolP384r1 - { 1 3 36 3 3 2 8 1 1 11 } x'0000000C' brainpoolP512r1 - { 1 3 36 3 3 2 8 1 1 13 }
76	64	Reserved
140	66	Value <i>d</i>
206	2742	Reserved
Variable length attribute section		
2948	2	Length of SUBJECT attribute in bytes (xx)
2950	2	Length of ID attribute in bytes (yy)
2952	2	Length of LABEL attribute in bytes (zz)
2954	2	Length of APPLICATION attribute in bytes (ww)
2956	20	Reserved
2976	4	Offset of SUBJECT attribute in bytes
2980	4	Offset of ID attribute in bytes
2984	4	Offset of LABEL attribute in bytes
2988	4	Offset of APPLICATION attribute in bytes
2992	40	Reserved
3032	xx+yy+zz+ww+ee	Private key attributes (variable length)

Table 48. Format of the token private key object (Version 3) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
3032 +xx+yy+zz+ww+ee		End of private key object

Table 49. Format of the token secret key object (Version 0)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for secret key object: "SECK"
4	2	Version: EBCDIC '00'
6	2	Length of the object in bytes
8	4	Flags (see Table 39 on page 253)
Object type-specific section		
12	4	Type of key: CKK_DES, CKK_DES2, CKK_DES3, CKK_AES
16	8	Start date for the key (in the format <i>yyyymmdd</i>)
24	8	End date for the key (in the format <i>yyyymmdd</i>)
32	4	Key generate mechanism CK_UNAVAILABLE_INFORMATION
36	2	Length of the key in bytes
38	32	Reserved
70	64	VALUE: value of the key
134	538	Reserved
672	4	Usage counter field
676	2	Reserved
678	2	Length of LABEL attribute in bytes (xx)
680	2	Length of APPLICATION attribute in bytes (yy)
682	2	Length of the ID attribute in bytes (zz)
684	20	Reserved
704	4	Offset of LABEL attribute in bytes
708	4	Offset of APPLICATION attribute in bytes
712	4	Offset of the ID attribute in bytes
716	40	Reserved
756	xx+yy+zz	Secret key attributes (variable length)

Table 49. Format of the token secret key object (Version 0) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
756 +xx+yy+zz		End of secret key object

Table 50. Format of the token secret key object (Version 1)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for secret key object: "SECK"
4	2	Version: EBCDIC '01'
6	2	Length of the object in bytes
8	4	Flags (see Table 39 on page 253)
Object type-specific section		
12	4	Type of key: CKK_DES, CKK_DES2, CKK_DES3, CKK_BLOWFISH, CKK_RC4, CKK_GENERIC_SECRET, and CKK_AES.
16	8	Start date for the key (in the format <i>yyyymmdd</i>)
24	8	End date for the key (in the format <i>yyyymmdd</i>)
32	4	Key generate mechanism CK_UNAVAILABLE_INFORMATION
36	2	Length of the key in bytes
38	32	Reserved
70	256	VALUE: value of the key
326	346	Reserved
672	4	Usage counter field
676	2	Reserved
678	2	Length of LABEL attribute in bytes (xx)
680	2	Length of APPLICATION attribute in bytes (yy)
682	2	Length of the ID attribute in bytes (zz)
684	20	Reserved
704	4	Offset of LABEL attribute in bytes
708	4	Offset of APPLICATION attribute in bytes
712	4	Offset of the ID attribute in bytes
716	40	Reserved

Table 50. Format of the token secret key object (Version 1) (continued)		
Offset (decimal) 188 +	Length of field (bytes)	Description
756	xx+yy+zz	Secret key attributes (variable length)
756 +xx+yy+zz		End of secret key object

Table 51. Format of the token secret key object (Version 3)		
Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for secret key object: "SECK"
4	2	Version: EBCDIC '03'
6	2	Length of the object in bytes
8	4	Flags (see Table 39 on page 253)
Object type-specific section		
12	4	Type of key: CKK_DES, CKK_DES2, CKK_DES3, CKK_BLOWFISH, CKK_RC4, CKK_GENERIC_SECRET, and CKK_AES.
16	8	Start date for the key (in the format <i>yyyymmdd</i>)
24	8	End date for the key (in the format <i>yyyymmdd</i>)
32	4	Key generate mechanism CK_UNAVAILABLE_INFORMATION
36	2	Length of the key in bytes
38	2	Length of secure key material (ee)
40	4	Offset to secure key material in bytes
44	26	Reserved
70	256	VALUE: value of the key
326	342	Reserved
668	1	Key field flags: Bit 0 Key check value present Bits 1-7 Reserved for IBM's use
672	4	Usage counter field
676	2	Reserved
678	2	Length of LABEL attribute in bytes (xx)

Table 51. Format of the token secret key object (Version 3) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
680	2	Length of APPLICATION attribute in bytes (yy)
682	2	Length of the ID attribute in bytes (zz)
684	20	Reserved
704	4	Offset of LABEL attribute in bytes
708	4	Offset of APPLICATION attribute in bytes
712	4	Offset of the ID attribute in bytes
716	40	Reserved
756	xx+yy+zz+ee	Secret key attributes (variable length)
756 +xx+yy+zz+ee		End of secret key object

Table 52. Format of the token domain parameters object (Version 1)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for token domain object: "DOMP"
4	2	Version: EBCDIC '01'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 39 on page 253)
Object type-specific section		
12	4	TYPE attribute: CKK_DSA or CKK_DH
16	28	Reserved
Algorithm-specific section (DSA)		
44	4	Length in bits of prime p
48	128	Reserved
176	128	Prime p
304	128	Reserved
432	128	Base g
560	20	Reserved
580	20	Subprime q
600	636	Reserved
Algorithm-specific section (DH)		
44	4	Length in bits of prime p

Table 52. Format of the token domain parameters object (Version 1) (continued)		
Offset (decimal) 188 +	Length of field (bytes)	Description
48	4	Reserved
52	256	Prime p
308	256	Reserved
564	256	Base g
820	416	Reserved
Variable length attribute section		
1236	2	Length of LABEL attribute in bytes (aa)
1238	2	Length of APPLICATION attribute in bytes (bb)
1240	20	Reserved
1260	4	Offset of LABEL attribute in bytes
1264	4	Offset of APPLICATION attribute in bytes
1268	40	Reserved
1308	$aa+bb$	Domain parameters attributes (variable length)
1308 + $aa+bb$		End of domain parameters object

Table 53. Format of the token domain parameters object (Version 2)		
Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for token domain object: "DOMP"
4	2	Version: EBCDIC '02'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 39 on page 253)
Object type-specific section		
12	4	TYPE attribute: CKK_DSA
16	28	Reserved
Algorithm-specific section (DSA)		
44	4	Length in bits of prime p
48	256	Prime p
304	256	Base g
560	8	Reserved
568	32	Subprime q

Table 53. Format of the token domain parameters object (Version 2) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
600	636	Reserved
Variable length attribute section		
1236	2	Length of LABEL attribute in bytes (<i>aa</i>)
1238	2	Length of APPLICATION attribute in bytes (<i>bb</i>)
1240	20	Reserved
1260	4	Offset of LABEL attribute in bytes
1264	4	Offset of APPLICATION attribute in bytes
1268	40	Reserved
1308	<i>aa+bb</i>	Domain parameters attributes (variable length)
1308 + <i>aa+bb</i>		End of domain parameters object

Table 54. Format of the token data object

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for data object: "DATA"
4	2	Version: EBCDIC '00'
6	2	Length of object, in bytes
8	4	Flags (see Table 39 on page 253)
Object type-specific section		
12	4	Reserved for IBM's use
16	28	Reserved for IBM's use
44	2	Length of VALUE attribute in bytes (<i>aa</i>)
46	2	Length of OBJECT_ID attribute in bytes (<i>bb</i>)
48	2	Length of LABEL attribute in bytes (<i>cc</i>)
50	2	Length of APPLICATION attribute in bytes (<i>dd</i>)
52	2	Length of ID attribute in bytes (<i>ee</i>)
54	22	Reserved for IBM's use
76	4	Offset of VALUE attribute in bytes
80	4	Offset of OBJECT_ID attribute in bytes
84	4	Offset of LABEL attribute in bytes
88	4	Offset of APPLICATION attribute in bytes

Table 54. Format of the token data object (continued)		
Offset (decimal) 188 +	Length of field (bytes)	Description
92	4	Offset of ID attribute in bytes
96	44	Reserved for IBM's use
140	$aa + bb + cc + dd + ee$	Data attributes (variable length)
$140 + aa + bb$ $+ cc + dd + ee$		End of data object

Common record format (KDSR)

The common record format (KDSR) is a record format for all KDS types (CKDS, PKDS, and TKDS) that allows for reference date tracking. KDSR format records were introduced in ICSF FMID HCR77A1. Version X'02' of the KDSR records have three distinct sections: a 140-byte fixed area, a variable length area that contains the cryptographic key material (key token), and a variable length metadata area that is used to store reference dates and other data. The large common record format (KDSRL) is fundamentally identical to KDSR except for the larger maximum record size for the CKDS and PKDS.

Format of the KDSR format record (Version X'02')

KDSR record sections:

- Fixed data area – 140 bytes
- Cryptographic key material (key token) – variable length
- Metadata area – variable length

Table 55. Format of the KDSR record fixed data area

Offset (Decimal)	Number of bytes	Field name	Description
0	72	VSAM Key	CKDS Bytes 0-63 Key Label. Bytes 64-71 Key Type. PKDS Bytes 0-63 Key Label. Bytes 64-71 Reserved. TKDS Bytes 0-31 Token name. Bytes 32-39 Sequence number. Byte 40 Blank for token. Character "T" for clear token object. Character "Y" for secure token object. Bytes 41-43 Blank characters. Bytes 44-71 Binary zeros.
72	8		Reserved.
80	1	Record Version	Version of the KDSR record format.
81	1	KDS Type	1=CKDS, 2=PKDS, 3=TKDS.
82	1	KDS Flags	Bit Meaning when set On 0 The key within the key material field is a partial key. (CKDS only). 1 Label must be unique. (CKDS only). 2 Preactive -> Active state audited. 3 Active -> Deactivated state audited.
83	1	KFP Count	Count of key fingerprints.

Table 55. Format of the KDSR record fixed data area (continued)

Offset (Decimal)	Number of bytes	Field name	Description
84	4	KDS Length	Length of the entire KDS record including key material and metadata.
88	8	Creation Date	The initial date the KDS record was created in the format <i>yyyymmdd</i> .
96	8	Creation Time	The initial time the KDS record was created in the format <i>hhmmssst</i> .
104	8	Update Date	The most recent date that this record was updated, in the format <i>yyyymmdd</i> or binary zero if the record has not been updated since creation.
112	8	Update Time	The most recent time that this record was updated, in the format <i>hhmmssst</i> or binary zero if the record has not been updated since creation.
120	4	Key Material Length	Length of the key material portion of the record.
124	4	Key Material Offset	Offset of the key material portion of the record, which is calculated from the start of the record.
128	4	Metadata Length	Length of the metadata area.
132	4	Metadata Offset	Offset of the metadata area in the record, which is calculated from the start of the record.
136	4	Reserved	Reserved.

Table 56. Format of KDSR metadata area

Offset (Decimal)	Number of bytes	Field name	Description
0	1	KDSR_MD_VERSION	
1	7		Reserved for IBM use.
8	8	KDSR_MD_REFDATE_STCKE	Reference date in STCKE format, high 8 bytes. Low bit in Byte 5 represents one second.
16	8	KDSR_MD_REFDATE	Reference date in the format <i>yyyymmdd</i> .
24	8	KDSR_MD_STARTDATE	Key material validity start date in the format <i>yyyymmdd</i> .
32	8	KDSR_MD_ENDDATE	Key material validity end date in the format <i>yyyymmdd</i> .
40	Variable		Reserved for IBM use.

<i>Table 57. Format of KDSR variable-length metadata block</i>			
Offset (Decimal)	Number of bytes	Field name	Description
0	2	KDSR_MD_TLV_TAG	Tag for block.
2	2	KDSR_MD_TLV_LEN	Length of the block, which includes the length of the tag and length fields.
4	Variable	KDSR_MD_DATA	Data.

AES key token format

AES internal fixed-length key token

Fixed-length AES key tokens are 64 bytes and consist of an internal key token identifier and a token version number, reserved fields, a flag byte containing various flag bits, and a token validation value.

Depending on the flag byte, the key token either contains an encrypted key, a clear key, or the key is absent. An encrypted key is encrypted under an AES master key that is identified by a master-key verification pattern (MKVP) in the key token. The key token contains a two-byte integer that specifies the length of the clear-key value in bits, valued to 0, 128, 192, or 256, and a two-byte integer that specifies the length of the encrypted-key value in bytes, valued to 0 or 32. An LRC checksum byte of the clear-key value is also in the key token.

All keys in fixed-length AES key tokens are DATA keys. If the flag byte indicates that a control vector (CV) is present, it must be all binary zeros. An all-zero CV represents the CV value of an AES DATA key. If a key is present without a control vector in a key token, that is accepted and the key is interpreted as an AES DATA key.

The AES internal key token is the structure that is used to hold AES keys that are either encrypted with the AES master-key or in clear text format.

Table 58 on page 282 shows the format for an AES internal key token.

<i>Table 58. AES internal fixed-length key token format</i>		
Offset (Dec)	Length of field (Bytes)	Description
00	1	X'01' (flag indicating that this is an internal key token)
01	3	Implementation-dependent bytes (X'000000' for ICSF)
04	1	Key token version number (X'04')
05	1	Reserved - must be set to X'00'

Table 58. AES internal fixed-length key token format (continued)		
Offset (Dec)	Length of field (Bytes)	Description
06	1	<p>Flag byte</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0 Encrypted key and master key verification pattern (MKVP) are present. Off for a clear key token. On for an encrypted key token.</p> <p>1 Control vector (CV) value in this token has been applied to the key.</p> <p>2 No key is present or the AES MKVP is not present if the key is encrypted.</p> <p>3- 7 Reserved. Must be set to 0.</p>
07	1	1-byte LRC checksum of clear key value.
08	8	<p>Master key verification pattern (MKVP).</p> <p>(For a clear AES key token, this value is hex zeros.)</p>
16	32	<p>Key value, if present. Contains either:</p> <ul style="list-style-type: none"> • A 256-bit encrypted-key value. The clear key value is padded on the right with binary zeros, and the entire 256-bit value is encrypted under the AES master-key using AES CBC mode with an initialization vector of binary zeros. • A 128-bit, 192-bit, or 256-bit clear-key value left-aligned and padded on the right with binary zeros for the entire 256-bit field.
48	8	<p>8-byte control vector.</p> <p>(For a clear AES key token, this value is hex zeros.)</p>
56	2	2-byte integer that specifies the length in bits of the clear key value.
58	2	<p>2-byte integer that specifies the length in bytes of the encrypted key value.</p> <p>(For a clear AES key token, this value is hex zeros.)</p>
60	4	Token validation value (TVV).

Token validation value

ICSF uses the *token validation value (TVV)* to verify that a token is valid. The TVV prevents a key token that is not valid or that is overlaid from being accepted by ICSF. It provides a checksum to detect a corruption in the key token.

When an ICSF callable service generates a key token, it generates a TVV and stores the TVV in bytes 60-63 of the key token. When an application program passes a key token to a callable service, ICSF checks the TVV. To generate the TVV, ICSF performs a twos complement ADD operation (ignoring carries and overflow) on the key token, operating on four bytes at a time, starting with bytes 0-3 and ending with bytes 56-59.

DES key token formats

DES fixed-length key token

Fixed-length DES key tokens are 64 bytes and consist of a DES-enciphered key, a control vector, various flag bits, a token identifier and version number, reserved fields, and a token-validation value. An internal key-token also includes a master-key verification pattern.

If an internal fixed-length DES key-token has a key present, it contains a key enciphered by a DES master key. If an external fixed-length DES key-token has a key present, it contains a key enciphered by a key-encrypting key.

Version X'00' tokens are single-length, double-length, and triple-length keys for all key types. DATA key tokens with zero control vectors are version X'00' for single-length keys and version X'01' for double-length and triple-length keys.

Table 59 on page 284 shows the format for a DES internal key token.

Table 59. DES internal fixed-length key token format		
Offset (Dec)	Length of field (Bytes)	Description
00	1	X'01' (flag indicating this is an internal key token).
01	3	Implementation-dependent bytes (X'000000' for ICSF).
04	1	Key token version number (X'00' or X'01').
05	1	Reserved (X'00').
06	1	<div>Flag byte</div> <div>Bit</div> <div>Meaning When Set On</div> <div>0 Encrypted key and master key verification pattern (MKVP) are present.</div> <div>1 Control vector (CV) value is present.</div> <div>2 Key is used for no control vector (NOCV) processing. Valid for transport keys only.</div> <div>3-6 Reserved.</div> <div>7 Export prohibited.</div>

Table 59. DES internal fixed-length key token format (continued)

Offset (Dec)	Length of field (Bytes)	Description
07	1	<p>Flag byte</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0-2 Key value encryption method.</p> <ul style="list-style-type: none"> • 000 - The key is encrypted by using the original CCA method (ECB). • 001 - The encrypted key is wrapped using the enhanced method and SHA-1 (WRAP-ENH). • 010 - The encrypted key is wrapped using the enhanced method and SHA-256 (WRAPENH2). Requires CV bit ENH-ONLY to be enabled. Only valid with version X'00' tokens. • 011 - The encrypted key wrapped using the enhanced method and SHA-256 and TDES-CMAC authentication code (WRAPENH3). Requires CV bit ENH-ONLY to be enabled. Only valid with version X'00' tokens. <p>These bits are ignored if the token contains a clear key.</p> <p>3-7 Reserved.</p>
08	8	<p>When the compliant-tag bit is off (bit 58 in the CV): Master Key Verification Pattern (MKVP).</p> <p>When the compliant-tag bit is on (bit 58 in the CV):</p> <p>Offset Length</p> <p>Description</p> <p>0 5 Truncated MKVP.</p> <p>5 2 Reserved.</p> <p>7 1 Key Derivation Function (KDF).</p> <p>When KDF is X'01' or X'02', the token is not considered compliant-tagged. Throughout the publications, they are referred to as DES KDF 01 or KDF 02 tokens. Only key tokens with a KDF higher than X'02' are referred to as compliant-tagged.</p>
16	8	A single-length key, the left half of a double-length key, or Part A of a triple-length key. The value is encrypted under the master key when flag bit 0 is on. Otherwise, it is in the clear.
24	8	<p>X'0000000000000000' if a single-length key, or the right half of a double-length key, or Part B of a triple-length key. The right half of the double-length key or Part B of the triple-length key is encrypted under the master key when flag bit 0 is on. Otherwise, it is in the clear.</p> <p>For WRAPENH3, this field will always hold ciphertext in order to obfuscate the length of the key.</p>

Table 59. DES internal fixed-length key token format (continued)		
Offset (Dec)	Length of field (Bytes)	Description
32	8	The control vector (CV) for a single-length key or the left half of the control vector for a double-length key. For WRAPENH3, this field will contain the control vector with key form bits (40-42) indicating a triple-length key.
40	8	X'0000000000000000' if a single-length key or the right half of the control vector for a double-length operational key. For WRAPENH3, this field will hold the 8-byte TDES-CMAC over the entire key token, with this field set to hex zero before the calculation of the TDES-CMAC.
48	8	X'0000000000000000' if a single-length key or double-length key, or Part C of a triple-length key. Part C of a triple-length key is encrypted under the master key when flag bit 0 is on. Otherwise, it is in the clear. For WRAPENH3, this field will always hold ciphertext in order to obfuscate the length of the key.
56	3	Reserved (X'000000').
59	1	Key length for zero CV DATA keys: Value Meaning B'00000000' Single-length key (version 0 only). B'00010000' Double-length key (version 1 only). B'00100000' Triple-length key (version 1 only). All other values are reserved and undefined.
60	4	Token validation value.

Note: A fixed-length key token that is stored in a non-KDSR CKDS will not have an MKVP or TVV. Before such a key token is used, the MKVP is copied from the CKDS header record, and the TVV is calculated and placed in the token.

Table 60 on page 286 shows the format for a DES external fixed-length key token.

Table 60. DES external fixed-length key token format		
Offset (Dec)	Length of field (Bytes)	Description
00	1	X'02' (flag indicating an external key token).
01	3	Implementation-dependent bytes (X'000000' for ICSF).
04	1	Key token version number (X'00' or X'01').
05	1	Reserved (X'00').

Table 60. DES external fixed-length key token format (continued)

Offset (Dec)	Length of field (Bytes)	Description
06	1	<p>Flag byte.</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0 Encrypted key and master key verification pattern (MKVP) are present.</p> <p>1 Control vector (CV) value is present.</p> <p>2 Key is used for no control vector (NOCV) processing. Valid for transport keys only.</p> <p>Other bits are reserved and are binary zeros.</p>
07	1	<p>Flag byte.</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0-2 Key value encryption method.</p> <ul style="list-style-type: none"> • 000 - The key is encrypted by using the original CCA method (ECB). • 001 - The encrypted key is wrapped using the enhanced method and SHA-1 (WRAP-ENH). • 010 - The encrypted key is wrapped using the enhanced method and SHA-256 (WRAPENH2). Requires CV bit ENH-ONLY to be enabled. Only valid with version X'00' tokens. • 011 - The encrypted key wrapped using the enhanced method and SHA-256 and TDES-CMAC authentication code (WRAPENH3). Requires CV bit ENH-ONLY to be enabled. Only valid with version X'00' tokens. <p>These bits are ignored if the token contains a clear key.</p> <p>3-7 Reserved.</p>
08	8	Reserved (X'0000000000000000').
16	8	Single-length key or left half of a double-length key, or Part A of a triple-length key. The value is encrypted under a transport key-encrypting key when flag bit 0 is on. Otherwise, it is in the clear.
24	8	<p>X'0000000000000000' if a single-length key or right half of a double-length key, or Part B of a triple-length key. The right half of a double-length key or Part B of a triple-length key is encrypted under a transport key-encrypting key when flag bit 0 is on. Otherwise, it is in the clear.</p> <p>For WRAPENH3, this field will always hold ciphertext in order to obfuscate the length of the key.</p>
32	8	<p>Control vector (CV) for single-length key or left half of CV for double-length key.</p> <p>For WRAPENH3, this field will contain the control vector with key form bits (40-42) indicating a triple-length key.</p>

Table 60. DES external fixed-length key token format (continued)		
Offset (Dec)	Length of field (Bytes)	Description
40	8	X'0000000000000000' if single-length key or right half of CV for double-length key. For WRAPENH3, this field will hold the 8-byte TDES-CMAC over the entire key token, with this field set to hex zero before the calculation of the TDES-CMAC.
48	8	X'0000000000000000' if a single-length key, double-length key, or Part C of a triple-length key. This key part is encrypted under a transport key-encrypting key when flag bit 0 is on. Otherwise, it is in the clear. For WRAPENH3, this field will always hold ciphertext in order to obfuscate the length of the key.
56-58	4	Reserved (X'000000').
59	1	Key length for zero CV DATA keys. Value Meaning B'00000000' Single-length key (version 0 only). B'00010000' Double-length key (version 1 only). B'00100000' Triple-length key (version 1 only). All other values are reserved and undefined.
60-63	4	Token validation value.

External RKX DES key token

Table 61 on page 288 defines an external DES key-token called an *RKX key-token*. An RKX key-token is a special token used exclusively by the Remote Key Export (CSNDRKX) and DES key-storage callable services (for example, Key Record Write). No other callable services use or reference an RKX key-token or key-token record.

Note: Callable services other than CSNDRKX and the DES key-storage do not support RKX key tokens or RKX key token records.

As can be seen in the table, RKX key tokens are 64 bytes in length, have a token identifier flag (X'02'), a token version number (X'10'), and room for encrypted keys like normal CCA DES key tokens. Unlike normal CCA DES key-tokens, RKX key tokens do not have a control vector, flag bits, and a token-validation value. In addition, they have a confounder value, a MAC value, and room for a third encrypted key.

Table 61. External RKX DES key-token format, version X'10'		
Offset	Length	Meaning
00	1	X'02' (a token identifier flag that indicates an external key-token)
01	3	Reserved, binary zero
04	1	The token version number (X'10')
05	2	Reserved, binary zero
07	1	Key length in bytes, including confounder

Table 61. External RKX DES key-token format, version X'10' (continued)		
Offset	Length	Meaning
08	8	Confounder
16	8	Key left
24	8	Key middle (binary zero if not used)
32	8	Key right (binary zero if not used)
40	8	<p>Rule ID</p> <p>The trusted block rule identifier used to create this key token. A subsequent call to Remote Key Export (CSNDRKX) can use this token with a trusted block rule that references the rule ID that must have been used to create this token. The trusted block rule can be compared with this rule ID for verification purposes.</p> <p>The Rule ID is an 8-byte string of ASCII characters, left justified and padded on the right with space characters. Acceptable characters are A...Z, a...z, 0...9, - (X'2D'), and _ (X'5F'). All other characters are reserved for future use.</p>
48	8	Reserved, binary zero
56	8	<p>MAC value</p> <p>ISO 16609 TDES CBC-mode MAC, computed over the 56 bytes starting at offset 0 and including the encrypted key value and the rule ID using the same MAC key that is used to protect the trusted block itself.</p> <p>This MAC value guarantees that the key and the rule ID cannot be modified without detection, providing integrity and binding the rule ID to the key itself. This MAC value must verify with the same trusted block used to create the key, thus binding the key structure to that specific trusted block.</p>

Note:

1. A fixed, randomly derived variant is exclusive-ORed with the MAC key before it is used to encipher the generated or exported key and confounder.
2. The MAC key is located within a trusted block (internal format) and can be recovered by decipherment under a variant of the PKA master key.
3. The trusted block is originally created in external form by the CSNDTBC callable service and then converted to internal form by the CSNDPKI callable service prior to the CSNDRKX call.

DES null key token

Table 62 on page 289 shows the format for a fixed length DES null key token.

Table 62. Format of Null Key Tokens	
Bytes	Description
0	X'00' (flag indicating this is a null key token).
1–15	Reserved (set to binary zeros).
16–23	Single-length encrypted key, or left half of double-length encrypted key, or Part A of triple-length encrypted key.
24–31	X'0000000000000000' if a single-length encrypted key, the right half of double-length encrypted key, or Part B of triple-length encrypted key.

Table 62. Format of Null Key Tokens (continued)	
Bytes	Description
32–39	X'0000000000000000' if a single-length encrypted key or double-length encrypted key.
40–47	Reserved (set to binary zeros).
48–55	Part C of a triple-length encrypted key.
56–63	Reserved (set to binary zeros).

Variable-length symmetric key token formats

Variable-length symmetric key token

The following table presents the format for a variable-length symmetric key token. The length of the token depends on the key type and algorithm.

Table 63. Variable-length symmetric key token		
Offset (Dec)	Length of Field (Bytes)	Description
		Header
0	1	Token flag X'00' for null tokens X'01' for internal tokens X'02' for external tokens
1	1	Reserved (X'00')
2	2	Length of the token in bytes
4	1	Token version number X'05' (May be X'00' for null tokens)
5	3	Reserved (X'000000')
		Wrapping information
8	1	Key material state. X'00' no key present (internal or external) X'01' key is clear (internal) X'02' key is encrypted under a key-encrypting key (external) X'03' key is encrypted under the master key (internal)

Table 63. Variable-length symmetric key token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
9	1	<p>Key verification pattern (KVP) type.</p> <p>X'00' No KVP</p> <p>X'01' AES master key verification pattern</p> <p>X'02' key-encrypting key verification pattern</p> <p>X'03' Truncated AES master key verification pattern with compliance information</p>
10	16	<p>Non-compliant tagged token: Verification pattern of the key used to wrap the payload.</p> <p>Compliant-tagged token: 5 bytes of the AES MKVP followed by 3 bytes of internal compliance information.</p> <p>Values are left justified.</p>
26	1	<p>Wrapping method - This value indicates the wrapping method used to protect the data in the encrypted section.</p> <p>X'00' key is in the clear</p> <p>X'02' AESKW</p> <p>X'03' PKOAEP2</p>
27	1	<p>Hash algorithm used in wrapping algorithm.</p> <ul style="list-style-type: none"> For wrapping method X'00' <p>X'00' None. For clear key tokens.</p> For wrapping method X'02' <p>X'02' SHA-256</p> For wrapping method X'03' <p>X'01' SHA-1</p> <p>X'02' SHA-256</p> <p>X'04' SHA-384</p> <p>X'08' SHA-512</p>

Table 63. Variable-length symmetric key token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
28	1	Payload version X'00' Variable-length payload X'01' Fixed-length payload All other values are reserved and must not be used.
29	1	Reserved (X'00')
		Associated data section
30	1	Associated data version (X'01')
31	1	Reserved (X'00')
32	2	Length of the associated data in bytes: <i>adl</i>
34	1	Length of the key name in bytes: <i>kl</i>
35	1	Length of the IBM extended associated data in bytes: <i>iead</i>
36	1	Length of the installation-definable associated data in bytes: <i>uad</i>
37	1	Reserved (X'00')
38	2	Length of the payload in bits: <i>pl</i>
40	1	Reserved (X'00')
41	1	Type of algorithm for which the key can be used X'01' DES X'02' AES X'03' HMAC

Table 63. Variable-length symmetric key token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
42	2	<p>Key type:</p> <p>For algorithm AES:</p> <p>X'0001' CIPHER</p> <p>X'0002' MAC</p> <p>X'0003' EXPORTER</p> <p>X'0004' IMPORTER</p> <p>X'0005' PINPROT</p> <p>X'0006' PINCALC</p> <p>X'0007' PINPRW</p> <p>X'0009' DKYGENKY</p> <p>X'000A' SECMSG</p> <p>X'000B' KDKGENKY</p> <p>For algorithm HMAC:</p> <p>X'0002' MAC</p> <p>For algorithm DES:</p> <p>X'0008' DESUSECV</p>
44	1	<p>Key-usage field count (<i>kuf</i>) - (1 byte)</p> <p>Key-usage field information defines restrictions on the use of the key.</p>

Table 63. Variable-length symmetric key token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
45	$kuf * 2$	<p>Key-usage fields ($kuf * 2$ bytes)</p> <ul style="list-style-type: none"> • For HMAC algorithm keys, refer to Table 65 on page 295. • For AES algorithm Key-Encrypting keys (Exporter or Importer), refer to Table 72 on page 309. • For AES algorithm CIPHER keys, refer to Table 73 on page 311. • For AES algorithm MAC keys, refer to Table 66 on page 297. • For AES algorithm PINCALC keys, refer to Table 67 on page 298. • For AES algorithm PINPROT keys, refer to Table 68 on page 299. • For AES algorithm PINPRW keys, refer to Table 69 on page 301. • For AES algorithm DKYGENKY keys, refer to Table 70 on page 303. • For AES algorithm SECMSG keys, refer to Table 71 on page 308. • For AES algorithm KDKGENKY keys, refer to Table 76 on page 318. • For DESUSECV keys, refer to Table 64 on page 295.
45 + $kuf * 2$	1	<p>Key-management field count (kmf) - (1 byte):</p> <ul style="list-style-type: none"> • For AES and HMAC keys: 2 (no pedigree information) or 3 (has pedigree information) • For DESUSECV keys: 1 <p>Key-management field information describes how the data is to be managed or helps with management of the key material.</p>
46 + $kuf * 2$	$kmf * 2$	<p>Key-management fields ($kmf * 2$ bytes):</p> <ul style="list-style-type: none"> • For AES and HMAC algorithm keys, refer to Table 74 on page 313. • For DESUSECV keys, refer to Table 75 on page 318.
46 + $kuf * 2 + kmf * 2$	kl	Key name (in ASCII)
46 + $kuf * 2 + kmf * 2 + kl$	$iead$	IBM extended associated data
46 + $kuf * 2 + kmf * 2 + kl + iead$	uad	Installation-defined associated data
		Clear key or encrypted payload

Table 63. Variable-length symmetric key token (continued)		
Offset (Dec)	Length of Field (Bytes)	Description
30 + <i>adl</i>	$(pl+7)/8$	<p>Encrypted AESKW payload (internal keys): The encrypted AESKW payload is created from the unencrypted AESKW payload which is made up of the ICV/pad length/hash options and hash length/hash options/hash of the associated data/key material/padding. See unencrypted AESKW payload.</p> <p>Encrypted PKOAEP2 payload (external keys): The encrypted PKOAEP2 payload is created using the PKCS #1 v1.2 encoding method for a given hash algorithm. The message (M) inside the encoding contains: [2 bytes: bit length of key] [clear HMAC key]. M is encoded using OAEP and then encrypted with an RSA public key according to the standard.</p> <p>Clear key payload: When the key is clear, only the key material will be in the payload padded to the nearest byte with binary zeros.</p>

Table 64. DESUSECV key-usage fields		
Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (<i>kuf</i>): 1
45	2	<p>Key-usage field 1</p> <p>High-order byte:</p> <p>B'0000 0000' Reserved</p> <p>All unused bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p>B'0000 0000' Reserved</p> <p>All unused bits are reserved and must be zero.</p>

Table 65. HMAC algorithm key-usage fields		
Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (<i>kuf</i>): 2

Table 65. HMAC algorithm key-usage fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
45	2	<p>Key-usage field 1</p> <p>High-order byte:</p> <p>1xxx xxxx Key can be used for generate.</p> <p>x1xx xxxx Key can be used for verify.</p> <p>All unused bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p>xxxx 1xxx The key can only be used in UDXs (used in KGN, KIM, KEX).</p> <p>xxxx 0xxx The key can be used in both UDXs and CCA.</p> <p>xxxx xuuu Reserved for UDXs, where uuu are UDX-defined bits.</p> <p>All unused bits are reserved and must be zero.</p>
47	2	<p>Key-usage field 2</p> <p>High-order byte:</p> <p>1xxx xxxx SHA-1 hash method is allowed for the key.</p> <p>x1xx xxxx SHA-224 hash method is allowed for the key.</p> <p>xx1x xxxx SHA-256 hash method is allowed for the key.</p> <p>xxx1 xxxx SHA-384 hash method is allowed for the key.</p> <p>xxxx 1xxx SHA-512 hash method is allowed for the key.</p> <p>All unused bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p>All bits are reserved and must be zero.</p>

Table 66. AES algorithm MAC key associated data

Offset (Dec)	Length of Field (Bytes)	Description
44	1	<p>Key-usage field count (kuf): 2 – 3 Count is based on whether the key is DK enabled or not:</p> <p>kuf</p> <p>DK enabled</p> <p>2 No</p> <p>3 Yes</p>
45	2	<p>Key-usage field 1</p> <p>High-order byte:</p> <p>B'00xx xxxx' Undefined.</p> <p>B'01xx xxxx' Key cannot be used for generate; key can be used for verify.</p> <p>B'10xx xxxx' Key can be used for generate; key cannot be used for verify.</p> <p>B'11xx xxx*' Key can be used for generate and verify. Not valid if offset 50 is X'01'.</p> <p>All unused bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p>xxxx 1xxx The key can only be used in UDXs (used in KGN, KIM, KEX).</p> <p>xxxx 0xxx The key can be used in both UDXs and CCA.</p> <p>xxxx xuuu Reserved for UDXs, where <i>uuu</i> are UDX-defined bits.</p> <p>All unused bits are reserved and must be zero.</p>
47	2	<p>Key-usage field 2</p> <p>High-order byte:</p> <p>X'01' CMAC mode.</p> <p>All unused bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p>B'0xxx xxxx' Key cannot be used by CSNBPTR2 to verify authentication data using NIST SP 800-38B CMAC for ISO-4 to ISO-4 PAN change. Only valid with key usage VERIFY.</p> <p>B'1xxx xxxx' Key can be used by CSNBPTR2 to verify authentication data using NIST SP 800-38B CMAC for ISO-4 to ISO-4 PAN change. Only valid with key usage VERIFY.</p> <p>All bits are reserved and must be zero.</p>

Table 66. AES algorithm MAC key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
49	2	<p>Key-usage field 3</p> <p>High-order byte when DK enabled:</p> <p>X'01' PIN_OP (DKPINOP)</p> <p>X'03' PIN_ADMIN1 (DKPINAD1)</p> <p>X'04' PIN_ADMIN2 (DKPINAD2)</p> <p>All unused values are reserved and must not be used.</p> <p>Low-order byte:</p> <p>X'01' DK enabled.</p> <p>All unused values are reserved and must not be used.</p>

Table 67. AES algorithm PINCALC key associated data

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (<i>kuf</i>): 3
45	2	<p>Key-usage field 1</p> <p>High-order byte:</p> <p>B'00xx xxxx' Undefined.</p> <p>B'10xx xxxx' Key can be used for generate; key cannot be used for verify.</p> <p>All unused bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p>xxxx 1xxx The key can only be used in UDXs (used in KGN, KIM, KEX).</p> <p>xxxx 0xxx The key can be used in both UDXs and CCA.</p> <p>xxxx xuuu Reserved for UDXs, where <i>uuu</i> are UDX-defined bits.</p> <p>All unused bits are reserved and must be zero.</p>

Table 67. AES algorithm PINCALC key associated data (continued)		
Offset (Dec)	Length of Field (Bytes)	Description
47	2	<p>Key-usage field 2</p> <p>High-order byte:</p> <p>X'00'</p> <p>Key can be used for Cipher Block Chaining (CBC).</p> <p>All unused values are reserved and must not be used.</p> <p>Low-order byte:</p> <p>All bits are reserved and must be zero.</p>
49	2	<p>Key-usage field 3</p> <p>High-order byte when DK enabled:</p> <p>X'01'</p> <p>PIN_OP (DKPINOP)</p> <p>All unused values are reserved and must not be used.</p> <p>Low-order byte:</p> <p>X'01'</p> <p>DK enabled.</p> <p>All unused values are reserved and must not be used.</p>

Table 68. AES algorithm PINPROT key associated data		
Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key usage fields count (kuf): 3 if value at offset 50 = X'01' (DK enabled), or 4 if value at offset 50 = X'00' (no field format specification).
45	2	<p>Key-usage field 1</p> <p>High-order byte:</p> <p>B'00xx xxxx'</p> <p>Undefined.</p> <p>B'01xx xxxx'</p> <p>Key cannot be used for encryption; key can be used for decryption. This is an inbound PIN protection key.</p> <p>B'10xx xxxx'</p> <p>Key can be used for encryption; key cannot be used for decryption. This is an outbound PIN protection key.</p> <p>B'11xx xxxx'</p> <p>Undefined.</p> <p>All unused bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p>xxxx 1xxx</p> <p>The key can only be used in UDXs (used in KGN, KIM, KEX).</p> <p>xxxx 0xxx</p> <p>The key can be used in both UDXs and CCA.</p> <p>xxxx xuuu</p> <p>Reserved for UDXs, where <i>uuu</i> are UDX-defined bits.</p> <p>All unused bits are reserved and must be zero.</p>

Table 68. AES algorithm PINPROT key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
47	2	<p>Key-usage field 2</p> <p>High-order byte:</p> <p>X'00'</p> <p>Key can be used for Cipher Block Chaining (CBC).</p> <p>All unused values are reserved and undefined.</p>
		<p>Key-usage field 2</p> <p>Low-order byte:</p> <p>Inbound key (value at offset 45 is B'01xx xxxx')</p> <p>B'xxx1 xxxx'</p> <p>Key can be used to verify an encrypted PIN (EPINVER).</p> <p>B'xxx0 xxxx'</p> <p>Key cannot be used to verify an encrypted PIN.</p> <p>B'xxxx 1xxx'</p> <p>Key can be used to generate an alternate clear PIN (CPINGENA).</p> <p>B'xxxx 0xxx'</p> <p>Key cannot be used to generate an alternate clear PIN.</p> <p>B'xxxx x1xx'</p> <p>Key can be used to translate an encrypted PIN (PINXLATE).</p> <p>B'xxxx x0xx'</p> <p>Key cannot be used to translate an encrypted PIN.</p> <p>B'xxxx xx1x'</p> <p>Key can be used to reformat an encrypted PIN (REFORMAT).</p> <p>B'xxxx xx0x'</p> <p>Key cannot be used to reformat an encrypted PIN.</p> <p>B'xxxx xxx1'</p> <p>Key can be used to restrictively reformat an ISO-4 encrypted PIN to an ISO-1 encrypted PIN (RFMT4TO1).</p> <p>B'xxxx xxx0'</p> <p>Key cannot be used to reformat an ISO-4 encrypted PIN to an ISO-1 encrypted PIN.</p> <p>All unused bits are reserved and must be zero.</p> <p>Outbound key (value at offset 45 is B'10xx xxxx')</p> <p>B'xx1x xxxx'</p> <p>Key can be used to encrypt a clear PIN (CPINENC).</p> <p>B'xx0x xxxx'</p> <p>Key cannot be used to encrypt a clear PIN.</p> <p>B'xxx1 xxxx'</p> <p>Key can be used to generate an encrypted PIN (EPINGEN).</p> <p>B'xxx0 xxxx'</p> <p>Key cannot be used to generate an encrypted PIN.</p> <p>B'xxxx x1xx'</p> <p>Key can be used to translate an encrypted PIN (PINXLATE).</p> <p>B'xxxx x0xx'</p> <p>Key cannot be used to translate an encrypted PIN.</p> <p>B'xxxx xx1x'</p> <p>Key can be used to reformat an encrypted PIN (REFORMAT).</p> <p>B'xxxx xx0x'</p> <p>Key cannot be used to reformat an encrypted PIN.</p> <p>B'xxxx xxx1'</p> <p>Key can be used to restrictively reformat an ISO-1 encrypted PIN to an ISO-4 encrypted PIN (RFMT1TO4).</p> <p>B'xxxx xxx0'</p> <p>Key cannot be used to restrictively reformat an ISO-1 encrypted PIN to an ISO-4 encrypted PIN.</p> <p>All unused bits are reserved and must be zero.</p>

Table 68. AES algorithm PINPROT key associated data (continued)		
Offset (Dec)	Length of Field (Bytes)	Description
49	2	Key-usage field 3, high-order byte No field format specification (value at offset 50 is X'00')
		Value Meaning X'00' No field format specification (NOFLDFMT) All unused values are reserved and undefined. DK enabled (value at offset 50 is X'01')
		X'01' PIN_OP (DKPINOP)
		X'02' PIN_OPP (DKPINOPP) X'03' PIN_ADMIN1 (DKPINAD1) All unused values are reserved and undefined.
51	2	Key-usage field 3, low-order byte Field format identifier: X'00' No field format specification (NOFLDFMT) X'01' DK enabled (DKPINOP, DKPINOPP, DKPINAD1) All unused values are reserved and undefined.
		Key-usage field 4, high-order byte PIN block format usage: B'xxxx xxx1' Allow ISO-4 All undefined bits are reserved and must be zero.
		Key-usage field 4, low-order byte All bits are reserved and must be zero.

Table 69. AES algorithm PINPRW key associated data		
Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (kuf): 3

Table 69. AES algorithm PINPRW key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
45	2	<p>Key-usage field 1</p> <p>High-order byte:</p> <p>B'00xx xxxx' Undefined.</p> <p>B'01xx xxxx' Key cannot be used for generate; key can be used for verify.</p> <p>B'10xx xxxx' Key can be used for generate; key cannot be used for verify.</p> <p>B'11xx xxxx' Undefined.</p> <p>All unused bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p>xxxx 1xxx The key can only be used in UDXs (used in KGN, KIM, KEX).</p> <p>xxxx 0xxx The key can be used in both UDXs and CCA.</p> <p>xxxx xuuu Reserved for UDXs, where <i>uuu</i> are UDX-defined bits.</p> <p>All unused bits are reserved and must be zero.</p>
47	2	<p>Key-usage field 2</p> <p>High-order byte:</p> <p>X'01' CMAC mode</p> <p>All unused values are reserved and must not be used.</p> <p>Low-order byte:</p> <p>All bits are reserved and must be zero.</p>
49	2	<p>Key-usage field 3</p> <p>High-order byte when DK enabled:</p> <p>X'01' PIN_OP (DKPINOP)</p> <p>All unused values are reserved and must not be used.</p> <p>Low-order byte:</p> <p>X'01' DK enabled.</p> <p>All unused values are reserved and must not be used.</p>

Table 70. AES algorithm DKYGENKY key associated data

Offset (Dec)	Length of Field (Bytes)	Description
44	1	<p>Key-usage field count (kuf): 2, 4-51</p> <p>Count is based on the type of key to diversify (value of offset 45):</p> <p>Value at offset 45 Type of key to diversify / kuf count</p> <p>X'00' D-ALL / kuf count: 2</p> <p>X'01' D-CIPHER / kuf count: 4</p> <p>X'02' D-MAC / kuf count: 4 (not DK enabled) or 5 (DK enabled)</p> <p>X'03' D-EXP / kuf count: 6</p> <p>X'04' D-IMP / kuf count: 6</p> <p>X'05' D-PPROT / kuf count: 5</p> <p>X'06' D-PCALC / kuf count: 5</p> <p>X'07' D-PPRW / kuf count: 5</p> <p>X'08' D-SECMSG / kuf count: 4</p> <p>X'09' D-KDKGKY / kuf count: 13, 25, 37, 49</p> <p>Each key-usage field is 2 bytes in length. The value in this field indicates how many 2-byte key usage fields follow.</p>

Table 70. AES algorithm DKYGENKY key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
45	2	<p>Key-usage field 1</p> <p>High-order byte: Defines the key type to be generated.</p> <p>X'00' Any type listed below (D-ALL)</p> <p>X'01' CIPHER (D-CIPHER)</p> <p>X'02' MAC (D-MAC)</p> <p>X'03' EXPORTER (D-EXP)</p> <p>X'04' IMPORTER (D-IMP)</p> <p>X'05' PINPROT (D-PPROT)</p> <p>X'06' PINCALC (D-PCALC)</p> <p>X'07' PINPRW (D-PPRW)</p> <p>X'08' SECMSG (D-SECMSG)</p> <p>X'09' KDKGENKY (D-KDKGKY)</p> <p>All other values are reserved and undefined.</p> <p>Low-order byte:</p> <p>1xxx xxxx The key can be used as the base derivation key in the AES DUKPT key derivation algorithm.</p> <p>xxxx 1xxx The key can only be used in UDXs (used in KGN, KIM, KEX).</p> <p>xxxx 0xxx The key can be used in both UDXs and CCA.</p> <p>xxxx xuuu Reserved for UDXs, where <i>uuu</i> are UDX-defined bits.</p> <p>All unused bits are reserved and must be zero.</p>

Table 70. AES algorithm DKYGENKY key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
47	2	<p>Key-usage field 2: Indicates the key usage.</p> <p>High-order byte (key-usage field level of control):</p> <p>B'1xxx xxxx' The key usage fields of the key to be generated must be equal (KUF-MBE) to the related generated key usage fields that start with key usage field 3 below.</p> <p>B'0xxx xxxx' The key usage fields of the key identifier to be generated must be permitted (KUF-MBP) based on the related generated-key usage fields that start with key usage field 3 below. A key to be diversified is not permitted to have a higher level of usage than the related key usage fields permit. The key to be diversified is only permitted to have key usage that is less than or equal to the related key usage fields. The UDX-ONLY bit of the related key usage fields must always be equal in both the generating key and the generated key.</p> <p>B'x1xx xxxx' The key management fields of the generated key must be permitted (KMF-MBP) based on the related key management fields of the DKYGENKY generating key. The key to be diversified is not permitted to have a higher level of management than the related key management fields permit. The key to be diversified is only permitted to have key management that is less than or equal to the related key management fields.</p> <p>B'x0xx xxxx' The key management fields of the key to be generated does not have to be permitted by the related key management fields of the DKYGENKY generating key.</p> <p>B'xx1x xxxx' The key management fields of the key to be generated must be equal (KMF-MBE) to the related key management fields of the DKYGENKY generating key.</p> <p>B'xx0x xxxx' The key management fields of the key to be generated does not have to be equal to the related key management fields of the DKYGENKY generating key.</p> <p>Note: When both bits 1 and 2 are OFF (B'x0xx xxxx' and B'xx0x xxxx'), the DKYGENKY generating key does not present any restrictions on the key management fields of the generated key.</p> <p>Undefined when the value at offset 45 = X'00' (D-ALL). All other values are reserved and undefined.</p> <p>Low-order byte (key-derivation sequence level):</p> <p>X'00' DKYL0. Generate a key based on the key usage byte at offset 45.</p> <p>X'01' DKYL1. Generate a level 0 diversified key with key type DKYGENKY.</p> <p>X'02' DKYL2. Generate a level 1 diversified key with key type DKYGENKY.</p> <p>All other values are reserved and undefined.</p>

Table 70. AES algorithm DKYGENKY key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
49 (if defined)	2	<p>Key-usage field 3 (related generated key usage fields):</p> <p>These values determine allowable key usage of key to be generated.</p> <p>Meaning depends on value of offset 45:</p> <p>X'01' Same as key-usage field 1 of AES CIPHER key.</p> <p>X'02' Same as key-usage field 1 of AES MAC key.</p> <p>X'03' Same as key-usage field 1 of AES EXPORTER key.</p> <p>X'04' Same as key-usage field 1 of AES IMPORTER key.</p> <p>X'05' Same as key-usage field 1 of AES PINPROT key.</p> <p>X'06' Same as key-usage field 1 of AES PINCALC key.</p> <p>X'07' Same as key-usage field 1 of AES PINPRW key.</p> <p>X'08' Same as key-usage field 1 of AES SECMSG key.</p> <p>X'09' Same as key-usage field 1 of AES KDKGENKY key.</p>

Table 70. AES algorithm DKYGENKY key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
51 (if defined)	2	<p>Key-usage field 4 (related generated key usage fields):</p> <p>These values determine allowable key usage of key to be generated.</p> <p>Meaning depends on value of offset 45:</p> <p>X'01' Same as key-usage field 2 of AES CIPHER key.</p> <p>X'02' High-order byte:</p> <p>X'01' CMAC mode.</p> <p>All unused bits are reserved and must be zero.</p> <p>Low-order byte (key-derivation sequence level):</p> <p>B'0xxx xxxx' Key cannot be used by CSNBPTR2 to verify authentication data using NIST SP 800-38B CMAC for ISO-4 to ISO-4 PAN change. Only valid with key usage VERIFY.</p> <p>B'1xxx xxxx' Key can be used by CSNBPTR2 to verify authentication data using NIST SP 800-38B CMAC for ISO-4 to ISO-4 PAN change. Only valid with key usage VERIFY.</p> <p>B'x0xx xxxx' AES DKYGENKY D-MAC key cannot be used as k-base-1 by CSNBDKG2 in the M of N MAC scheme.</p> <p>B'x1xx xxxx' AES DKYGENKY D-MAC key can be used as k-base-1 by CSNBDKG2 in the M of N MAC scheme.</p> <p>B'xx0x xxxx' AES DKYGENKY D-MAC key cannot be used as k-base-2 by CSNBMMS in the M of N MAC scheme.</p> <p>B'xx1x xxxx' AES DKYGENKY D-MAC key can be used as k-base-2 by CSNBMMS in the M of N MAC scheme.</p> <p>X'03' Same as key-usage field 2 of AES EXPORTER key.</p> <p>X'04' Same as key-usage field 2 of AES IMPORTER key.</p> <p>X'05' Same as key-usage field 2 of AES PINPROT key.</p> <p>X'06' Same as key-usage field 2 of AES PINCALC key.</p> <p>X'07' Same as key-usage field 2 of AES PINPRW key.</p> <p>X'08' Same as key-usage field 2 of AES SECMSG key.</p> <p>X'09' Same as key-usage field 2 of AES KDKGENKY key (1st KUF of required 1st active/passive related key-usage field blocks).</p>

Table 70. AES algorithm DKYGENKY key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
53 (if defined)	2	<p>Key-usage field 5 (related generated key usage fields):</p> <p>These values determine allowable key usage of key to be generated.</p> <p>Meaning depends on value of offset 45:</p> <p>X'02' Same as key-usage field 3 of AES MAC key.</p> <p>X'03' Same as key-usage field 3 of AES EXPORTER key.</p> <p>X'04' Same as key-usage field 3 of AES IMPORTER key.</p> <p>X'05' Same as key-usage field 3 of AES PINPROT key.</p> <p>X'06' Same as key-usage field 3 of AES PINCALC key.</p> <p>X'07' Same as key-usage field 3 of AES PINPRW key.</p> <p>X'09' Same as key-usage field 3 of AES KDKGENKY key (1st KUF of required 1st active/passive related key-usage field blocks).</p>
55 (if defined)	2	<p>Key-usage field 6 (related generated key usage fields):</p> <p>These values determine allowable key usage of key to be generated.</p> <p>Meaning depends on value of offset 45:</p> <p>X'03' Same as key-usage field 4 of AES EXPORTER key.</p> <p>X'04' Same as key-usage field 4 of AES IMPORTER key.</p> <p>X'05' Same as key-usage field 4 of AES PINPROT key.</p> <p>X'09' Same as key-usage field 4 of AES KDKGENKY key (1st KUF of required 1st active/passive related key-usage field blocks).</p>

Table 71. AES algorithm SECMSG key associated data

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (<i>kuf</i>): 2

Table 71. AES algorithm SECMSG key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
45	2	<p>Key-usage field 1</p> <p>High-order byte: Secure message encryption enablement:</p> <p>Value Meaning</p> <p>X'00' Enable the encryption of PINs in an EMV secure message (SMPIN). All other values are reserved and undefined.</p> <p>Low-order byte:</p> <p>xxxx 1xxx The key can only be used in UDXs (used in KGN, KIM, KEX).</p> <p>xxxx 0xxx The key can be used in both UDXs and CCA.</p> <p>xxxx xuuu Reserved for UDXs, where <i>uuu</i> are UDX-defined bits. All unused bits are reserved and must be zero.</p>
47	2	<p>Key-usage field 2: Indicates the key usage.</p> <p>High-order byte: Service restriction:</p> <p>Value Meaning</p> <p>X'00' Any verb can use this key (ANY-USE).</p> <p>X'01' Only CSNBDPC can use this key (DPC-ONLY). All other values are reserved and undefined.</p> <p>Low-order byte (reserved). All unused bits are reserved and must be zero</p>

Table 72. AES algorithm KEK key-usage fields

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (<i>kuf</i>): 4

Table 72. AES algorithm KEK key-usage fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
45	2	<p>Key-usage field 1, high-order byte</p> <p>EXPORTER:</p> <p>1xxx xxx0 Key can be used for EXPORT.</p> <p>x1xx xxx0 Key can be used for TRANSLAT.</p> <p>xx1x xxx0 Key can be used for GEN-OPEX.</p> <p>xxx1 xxx0 Key can be used for GEN-IMEX.</p> <p>xxxx 1xx0 Key can be used for GEN-EXEX.</p> <p>xxxx x1x0 Key can be used for GEN-PUB.</p> <p>0000 0001 Key can wrap an AES or DES key using the Key Block Binding key wrapping method as defined in ISO/DIS 20038 (EXPTT31D).</p> <p>All unused bits are reserved and must be zero.</p> <p>IMPORTER:</p> <p>1xxx xxx0 Key can be used for IMPORT.</p> <p>x1xx xxx0 Key can be used for TRANSLAT.</p> <p>xx1x xxx0 Key can be used for GEN-OPIM.</p> <p>xxx1 xxx0 Key can be used for GEN-IMEX.</p> <p>xxxx 1xx0 Key can be used for GEN-IMIM.</p> <p>xxxx x1x0 Key can be used for GEN-PUB.</p> <p>0000 0001 Key can unwrap an AES or DES key using the Key Block Binding key wrapping method as defined in ISO/DIS 20038 (IMPTT31D).</p> <p>All unused bits are reserved and must be zero.</p>
		<p>Key-usage field 1, low-order byte</p> <p>UDX control:</p> <p>xxxx 1xxx The key can only be used in UDXs (used in KGN, KIM, KEX).</p> <p>xxxx 0xxx The key can be used in both UDXs and CCA.</p> <p>xxxx xuuu Reserved for UDXs, where uuu are UDX-defined bits.</p> <p>All unused bits are reserved and must be zero.</p>
47	2	<p>Key-usage field 2</p> <p>High-order byte:</p> <p>0000 0001 Key can wrap a TR-31 key block version "D" (VARDRV-D). Only valid if value at offset 45 is B'0000 0001' (EXPTT31D or IMPTT31D).</p> <p>1xxx xxx0 Key can wrap a TR-31 key block. Not valid if value at offset 45 is B'0000 0001' (EXPTT31D or IMPTT31D).</p> <p>All unused bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p>xxxx xxx1 This KEK can export a key in RAW format.</p> <p>All unused bits are reserved and must be zero.</p>

Table 72. AES algorithm KEK key-usage fields (continued)		
Offset (Dec)	Length of Field (Bytes)	Description
49	2	<p>Key-usage field 3</p> <p>High-order byte:</p> <p>1xxx xxxx Key can wrap DES keys.</p> <p>x1xx xxxx Key can wrap AES keys.</p> <p>xx1x xxxx Key can wrap HMAC keys.</p> <p>xxx1 xxxx Key can wrap RSA keys.</p> <p>xxxx 1xxx Key can wrap ECC keys.</p> <p>xxxx x1xx Key can wrap QSA keys.</p> <p>All unused bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p>All bits are reserved and must be zero.</p>
51	2	<p>Key-usage field 4</p> <p>High-order byte:</p> <p>1xxx xxxx Key can wrap DATA class keys.</p> <p>x1xx xxxx Key can wrap KEK class keys.</p> <p>xx1x xxxx Key can wrap PIN class keys.</p> <p>xxx1 xxxx Key can wrap DERIVATION class keys.</p> <p>xxxx 1xxx Key can wrap CARD class keys.</p> <p>xxxx x1xx Key can wrap CVAR class keys.</p> <p>All unused bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p>All bits are reserved and must be zero.</p>

Table 73. AES algorithm CIPHER key associated data		
Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (<i>kuf</i>): 2

Table 73. AES algorithm CIPHER key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
45	2	<p>Key-usage field 1</p> <p>High-order byte:</p> <p>1xxx xxxx Key can be used for encryption.</p> <p>x1xx xxxx Key can be used for decryption.</p> <p>xx1x xxxx Key can only be used for data translate.</p> <p>All unused bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p>xxxx 1xxx The key can only be used in UDXs (used in KGN, KIM, KEX).</p> <p>xxxx 0xxx The key can be used in both UDXs and CCA.</p> <p>xxxx xuuu Reserved for UDXs, where uuu are UDX-defined bits.</p> <p>All unused bits are reserved and must be zero.</p>

Table 73. AES algorithm CIPHER key associated data (continued)		
Offset (Dec)	Length of Field (Bytes)	Description
47	2	<p>Key-usage field 2</p> <p>High-order byte:</p> <p>X'00' Key can be used for Cipher Block Chaining (CBC).</p> <p>X'01' Key can be used for Electronic Code Book (ECB).</p> <p>X'02' Key can be used for Cipher Feedback (CFB).</p> <p>X'03' Key can be used for Output Feedback (OFB).</p> <p>X'04' Key can be used for Galois/Counter Mode (GCM)</p> <p>X'05' Key can be used for XEX-based Tweaked CodeBook Mode with CipherText Stealing (XTS)</p> <p>X'06' Key can be used for Format Preserving method FF1.</p> <p>X'07' Key can be used for Format Preserving method FF2.</p> <p>X'08' Key can be used for Format Preserving method FF2.1.</p> <p>X'FF' Key can be used for any mode of encryption</p> <p>All unused values are reserved and must not be used.</p> <p>Low-order byte:</p> <p>All bits are reserved and must be zero.</p>

Table 74. AES and HMAC algorithm key-management fields		
Offset (Dec)	Length of Field (Bytes)	Description
49	1	Key-management field count (kmf): 2 or 3.

Table 74. AES and HMAC algorithm key-management fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
50	2	<p>Key-management field 1.</p> <p>High-order byte:</p> <p>1xxx xxxx Allow export using symmetric key.</p> <p>x1xx xxxx Allow export using unauthenticated asymmetric key.</p> <p>xx1x xxxx Allow export using authenticated asymmetric key.</p> <p>xxx1 xxxx Allow export in RAW format.</p> <p>xxxx 1xxx Allow export to CPACF protected key format.</p> <p>xxxx xxx1 Compliant-tagged key. Applies to AES only.</p> <p>All other bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p>--symmetric--</p> <p>1xxx xxxx Prohibit export using DES key.</p> <p>x1xx xxxx Prohibit export using AES key.</p> <p>--asymmetric--</p> <p>xxxx 1xxx Prohibit export using RSA key.</p> <p>All other bits are reserved and must be zero.</p>

Table 74. AES and HMAC algorithm key-management fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
48 + kuf * 2	2	<p>Key-management field 2.</p> <p>High-order byte:</p> <p>11xx xxxx Key, if present, is incomplete. Key requires at least 2 more parts.</p> <p>10xx xxxx Key, if present, is incomplete. Key requires at least 1 more part.</p> <p>01xx xxxx Key, if present, is incomplete. Key can be completed or have more parts added.</p> <p>00xx xxxx Key, if present, is complete. No more parts can be added.</p> <p>All other bits are reserved and must be zero.</p> <p>Low-order byte (Security History):</p> <p>xxx1 xxxx Key was encrypted with an untrusted KEK.</p> <p>xxxx 1xxx Key was in a format without type/usage attributes.</p> <p>xxxx x1xx Key was encrypted with key weaker than itself.</p> <p>xxxx xx1x Key was in a non-CCA format.</p> <p>xxxx xxx1 Key was encrypted in ECB mode.</p> <p>All other bits are reserved and must be zero.</p>

Table 74. AES and HMAC algorithm key-management fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
50 + kuf * 2	2	<p>Key-management field 3 - Pedigree (this field may or may not be present). Indicates how key was originally created and how it got into the current system. High-order byte: Pedigree Original.</p> <p>X'00' Unknown (Key Token Build2, Key Translate2).</p> <p>X'01' Other - method other than those defined here, probably used in UDX.</p> <p>X'02' Randomly Generated (Key Generate2).</p> <p>X'03' Established by key agreement (ECC Diffie-Hellman).</p> <p>X'04' Created from cleartext key components (Key Part Import2).</p> <p>X'05' Entered as a cleartext key value (Key Part Import2, Secure Key Import2).</p> <p>X'06' Derived from another key.</p> <p>X'07' Cleartext keys or key parts that were entered at TKE and secured from there to the target card (operational key load).</p> <p>All unused values are reserved and undefined.</p>

Table 74. AES and HMAC algorithm key-management fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
50 + kuf * 2 (cont'd)	2 (cont'd)	<p>Low-order byte: Pedigree Current.</p> <p>X'00' Unknown (Key Token Build2).</p> <p>X'01' Other - method other than those defined here, probably used in UDX.</p> <p>X'02' Randomly Generated (Key Generate2).</p> <p>X'03' Established by key agreement (ECC Diffie-Hellman).</p> <p>X'04' Created from cleartext key components (Key Part Import2).</p> <p>X'05' Entered as a cleartext key value (Key Part Import2, Secure Key Import2).</p> <p>X'06' Derived from another key.</p> <p>X'07' Imported from a CCA 05 variable length token with pedigree field (Symmetric Key Import2).</p> <p>X'08' Imported from a CCA 05 variable length token with no pedigree field (Symmetric Key Import2).</p> <p>X'09' Imported from a CCA token that had a CV.</p> <p>X'0A' Imported from a CCA token that had no CV or a zero CV.</p> <p>X'0B' Imported from a TR-31 key block that contained a CCA CV (ATTR-CV option) (TR-31 Import).</p> <p>X'0C' Imported from a TR-31 key block that did not contain a CCA CV (TR-31 Import).</p> <p>X'0D' Imported using PKCS 1.2 RSA encryption (Symmetric Key Import2).</p> <p>X'0E' Imported using PKCS OAEP encryption (Symmetric Key Import2).</p> <p>X'0F' Imported using PKA92 RSA encryption (Symmetric Key Import2).</p> <p>X'10' Imported using RSA ZERO-PAD encryption (Symmetric Key Import2).</p> <p>X'11' Converted from a CCA token that had a CV (Key Translate2).</p> <p>X'12' Converted from a CCA token that had no CV or a zero CV (Key Translate2).</p> <p>X'13' Cleartext keys or key parts that were entered at TKE and secured from there to the target card (operational key load).</p>

Table 74. AES and HMAC algorithm key-management fields (continued)		
Offset (Dec)	Length of Field (Bytes)	Description
50 + kuf * 2 (cont'd)	2 (cont'd)	<p>Low-order byte: Pedigree Current.</p> <p>X'14' Exported from a CCA 05 variable length token with pedigree field (Symmetric Key Export).</p> <p>X'15' Exported from a CCA 05 variable length token with no pedigree field (Symmetric Key Export).</p> <p>X'16' Exported using PKCS OAEP encryption (Symmetric Key Export).</p> <p>All unused values are reserved and undefined.</p>

Table 75. DESUSECV key-management fields		
Offset (Dec)	Length of Field (Bytes)	Description
47	1	Key-management field count (<i>kmf</i>): 1
48	2	<p>Key-management field 1</p> <p>High-order byte:</p> <p>B'0000 0000' Reserved</p> <p>All unused bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p>B'0000 0000' Reserved</p> <p>All unused bits are reserved and must be zero.</p>

Table 76. AES algorithm KDKGENKY key-usage fields		
Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (<i>kuf</i>): 13, 25, 37, or 49. Each key-usage field is two bytes in length.

Table 76. AES algorithm KDKGENKY key-usage fields (continued)		
Offset (Dec)	Length of Field (Bytes)	Description
45	2	<p>Key-usage field 1, high-order byte</p> <p>Key diversification type:</p> <p>X'00' Entity type A (KDKTYPEA)</p> <p>X'01' Entity type B (KDKTYPEB)</p> <p>All other values are reserved and undefined.</p>
		<p>Key-usage field 1, low-order byte</p> <p>User-defined extension control:</p> <p>B'xxxx 1xxx' Key can only be used in UDXs (used in KGN, KIM, KEX).</p> <p>B'xxxx 0xxx' Key can be used in UDXs and CCA.</p> <p>B'xxxx xuuu' Reserved for UDXs, where <i>uuu</i> are UDX-defined bits.</p> <p>All unused bits are reserved and must be zero.</p>
47	$amb + acb + apb + awp$	<p>Key-usage field 2 (active/passive key-usage field block)</p> <p>For the format of an active/passive key-usage field block, see the 'AES DKYGENKY and AES KDKGENKY active/passive related key-usage field block' table in <i>z/OS Cryptographic Services ICSF Application Programmer's Guide</i>.</p> <p>At least one active/passive related KUF block is required and a maximum of four blocks is allowed. The sequence of the blocks is in ascending numerical order, based on the numeric value of the key type of the key to be derived or generated, found at offset 0 of the block header. This is the required order of the blocks:</p> <ol style="list-style-type: none"> 1. Active AES MAC/passive AES MAC related KUF block (<i>amb</i> bytes, where $amb = 0$ or 24). 2. Active AES CIPHER/passive AES CIPHER related KUF block (<i>acb</i> bytes, where $acb = 0$ or 24). 3. Active AES PINPROT/passive AES PINPROT related KUF block (<i>apb</i> bytes, where $apb = 0$ or 24). 4. Active AES EXPORTER/passive AES EXPORTER related KUF block (<i>awp</i> bytes, where $awp = 0$ or 24). <p>where $amb + acb + apb + awp = 24, 48, 72, \text{ or } 96$.</p>

Variable-length symmetric null key token

The following table shows the format for a variable-length symmetric null key token.

Table 77. Variable-length symmetric null token	
Bytes	Description
0	X'00' Token identifier (indicates that this is a null key token).

Table 77. Variable-length symmetric null token (continued)	
Bytes	Description
1	Version, X'00'.
2-3	X'0008' Length of the key token structure.
4-7	Ignored (zero).

X9.143 (TR-31) key block header and optional block data

X9.143 (TR-31) key blocks are supported by ICSF. This topic describes the format of the X9.143 key block header and the header values that ICSF supports. It also describes the TR-31 optional blocks that can be used by ICSF. See ANSI X9.143 Retail Financial Services Interoperable Secure Key Block Specification for the definition of a X9.143 key block.

X9.143 key block header

A X9.143 key block always begins with a key block header consisting of a required 16-byte fixed-length portion followed by from 0 - 99 variable-length optional blocks. Table 78 on page 320 shows the format of the required fixed-length portion of the header and only shows those values supported by ICSF.

Offset (bytes)	Length (bytes)	X9.143 key block header field name										
0	1	<p>Key block version ID.</p> <p>Identifies the version of the key block, which defines the method by which it is cryptographically protected and the content and layout of the block.</p> <p>The following key block version ID values defined by X9.143 are the only ones supported by ICSF:</p> <table><thead><tr><th>ASCII value</th><th>Meaning</th></tr></thead><tbody><tr><td>"A"</td><td>Key block protected by the Key Variant Binding Method 2005 Edition (maps to TR-31 Translate rule-array keyword VARXOR-A). This method is deprecated and should not be used for any new development.</td></tr><tr><td>"B"</td><td>Key block protected by the Key Derivation Binding Method 2010 Edition (maps TR-31 Translate rule-array keyword VARDRV-B).</td></tr><tr><td>"C"</td><td>Key block protected by the Key Variant Binding Method 2010 Edition (maps to TR-31 Translate rule-array keyword VARXOR-C).</td></tr><tr><td>"D"</td><td>Key block protected by the Key Derivation Binding Method 2017 Edition (maps to TR-31 Translate rule-array keyword VARDRV-D).</td></tr></tbody></table>	ASCII value	Meaning	"A"	Key block protected by the Key Variant Binding Method 2005 Edition (maps to TR-31 Translate rule-array keyword VARXOR-A). This method is deprecated and should not be used for any new development.	"B"	Key block protected by the Key Derivation Binding Method 2010 Edition (maps TR-31 Translate rule-array keyword VARDRV-B).	"C"	Key block protected by the Key Variant Binding Method 2010 Edition (maps to TR-31 Translate rule-array keyword VARXOR-C).	"D"	Key block protected by the Key Derivation Binding Method 2017 Edition (maps to TR-31 Translate rule-array keyword VARDRV-D).
ASCII value	Meaning											
"A"	Key block protected by the Key Variant Binding Method 2005 Edition (maps to TR-31 Translate rule-array keyword VARXOR-A). This method is deprecated and should not be used for any new development.											
"B"	Key block protected by the Key Derivation Binding Method 2010 Edition (maps TR-31 Translate rule-array keyword VARDRV-B).											
"C"	Key block protected by the Key Variant Binding Method 2010 Edition (maps to TR-31 Translate rule-array keyword VARXOR-C).											
"D"	Key block protected by the Key Derivation Binding Method 2017 Edition (maps to TR-31 Translate rule-array keyword VARDRV-D).											
1	4	<p>Key block length.</p> <p>Provides the key-block length after encoding. Length includes the entire block (header + encrypted confidential data + MAC).</p>										

Table 78. Format and supported values of the required header for a X9.143 key block (continued)

Offset (bytes)	Length (bytes)	X9.143 key block header field name
5	2	<p>Key usage.</p> <p>Provides information about the intended function of the protected key/sensitive data.</p> <p>The following key usage values defined by TR-31 are the only ones fully supported by ICSF:</p> <ul style="list-style-type: none"> • "B0", "B1", "B3" • "C0" • "D0", "D3" • "E0", "E1", "E2", "E3", "E4", "E5" • "F0", "F1", "F2", "F3", "F4" • "I0" • "K0", "K1", "K4" • "M0", "M1", "M3", "M6", "M7" • "P0" • "V0", "V1", "V2" <p>See Table 79 on page 323 for descriptions of the key usage values.</p>
7	1	<p>Algorithm.</p> <p>The approved algorithm for which the key may be used.</p> <p>The following algorithm values defined by TR-31 are the only ones supported by ICSF:</p> <p>ASCII value Meaning</p> <p>"A" Advanced Encryption Standard (AES) (maps to TR-31 Translate AES key in the source key-token).</p> <p>"D" Data Encryption Algorithm (DEA) (maps to TR-31 Translate single-length DES key in the source key-token).</p> <p>"H" (ASC X9 TR 31-2018) Hash-based message authentication code (HMAC) (specify the underlying hash algorithm in optional field with a block ID of "HM") (maps to TR-31 Translate HMAC key in the source key-token).</p> <p>"H" (ISO 20038) HMAC-SHA-1 (maps to TR-31 Translate HMAC key in the source key-token).</p> <p>"I" (ISO 20038) HMAC-SHA-2 (maps to TR-31 Translate HMAC key in the source key-token).</p> <p>"T" Triple Data Encryption Algorithm (TDEA) (maps to TR-31 Translate double-length or triple-length Triple-DES key in the source key-token).</p>

Table 78. Format and supported values of the required header for a X9.143 key block (continued)

Offset (bytes)	Length (bytes)	X9.143 key block header field name
8	1	<p>Mode of use.</p> <p>Defines the operation the protected key can perform.</p> <p>The following mode of use values defined by TR-31 are the only ones supported by ICSF:</p> <p>ASCII value Meaning</p> <p>"B" Both encrypt and decrypt data, wrap and unwrap keys (maps to TR-31 Translate rule-array keyword ENCDEC).</p> <p>"C" Both generate and verify of check/PIN values (maps to TR-31 Translate rule-array keyword GENVER).</p> <p>"D" Decrypt data, unwrap keys only (maps to TR-31 Translate rule-array keyword DEONLY).</p> <p>"E" Encrypt data, wrap keys only (maps to TR-31 Translate rule-array keyword ENONLY).</p> <p>"G" Generate of check/PIN values only (maps to TR-31 Translate rule-array keyword GEN-ONLY).</p> <p>"N" No special restrictions (other than restrictions implied by the key usage) (maps to TR-31 Translate rule-array keyword ANY).</p> <p>"V" Verify of check/PIN values only (maps to TR-31 Translate rule-array keyword VERONLY).</p> <p>"X" Key used to derive other key or keys (maps to TR-31 Translate rule-array keyword DERIVE).</p> <p>"1" IBM proprietary mode for keys (maps to TR-31 Translate rule-array keyword ATTR-CV).</p>
9	2	<p>Key version number.</p> <p>Version number used to indicate that contents of the key block is a component (partial key) or to prevent reinjection of old keys.</p>

Table 78. Format and supported values of the required header for a X9.143 key block (continued)

Offset (bytes)	Length (bytes)	X9.143 key block header field name
11	1	<p>Exportability.</p> <p>Defines whether the protected key may be transferred outside the cryptographic domain in which the key is found.</p> <p>The following exportability values defined by TR-31 are the only ones supported by CCA:</p> <p>ASCII value</p> <p>Meaning</p> <p>"E" Extra sensitive: Key exportable under a key-encrypting key meeting the requirements of X9.24 Parts 1 or 2; no ECB mode KEK allowed (maps to TR-31 Translate rule-array keyword EXP-TRST).</p> <p>"N" Non-exportable (maps to TR-31 Translate rule-array keyword EXP-NONE).</p> <p>"S" Sensitive: Key exportable under any key-encrypting key not necessarily meeting the requirements of X9.24 Parts 1 or 2 (maps to TR-31 Translate rule-array keyword EXP-ANY).</p>
12	2	<p>Number of optional blocks.</p> <p>Defines the number of optional blocks included in the key block. The minimum value is zero and the maximum is 99.</p>
14	1	<p>Key Context.</p> <p>Defines whether the key block is in a key exchange context (wrapped by a transport key) or in a storage context (for example, wrapped by the master key). The key context value does not require a certain exportability setting.</p> <p>ASCII value</p> <p>Meaning</p> <p>"0" This is the equivalent to external CCA key tokens.</p> <p>"1" This is the equivalent to internal CCA key tokens.</p> <p>"2" This is the equivalent to external CCA key tokens.</p>
15	1	Reserved.

Note: Information specific to X9-143 is taken from ANSI X9.143 Retail Financial Services Interoperable Secure Key Block Specification.

Table 79. Key usage values and meanings

ASCII value	Meaning
"B0"	BDK base derivation key (maps to TR-31 Translate rule array keyword "BDK"). This key is used to derive the initial PIN encryption key (IPEK) in the derived unique key per transaction (DUKPT) process defined in X9.24-3.

Table 79. Key usage values and meanings (continued)

ASCII value	Meaning
"B1"	Initial DUKPT key (maps to TR31 Key Create rule array keyword "DUKPT"). This key is used as the initial PIN encryption key (IPEK) in the derived unique key per transaction (DUKPT) process as defined in X9.24-3. Only valid with Key Usage 'X'.
"B3"	Key Derivation key (maps to TR31 Key Create rule array keyword "KDK"). This key is used as an input to an irreversible key derivation function to derive other keys. Only valid with Key Usage 'X'.
"C0"	CVK card verification key (maps to TR-31 Translate rule array keyword "CVK"). These are the keys used for computing or verifying (against a supplied value) a card verification code with the CVV, CVC, CVC2, and CVV2 algorithms.
"D0"	Symmetric data encryption key (maps to TR31 Key Create rule array keyword "ENC").
"D3"	Symmetric data encryption key for sensitive data (maps to TR31 Key Create rule array keyword "ENCSENS").
"E0"	Derivation key for an EMV/chip issuer master key: Application cryptograms (maps to TR31 Key Create rule array keyword "EMVAC-E").
"E1"	Derivation key for an EMV/chip issuer master key: Secure messaging for confidentiality (maps to TR31 Key Create rule array keyword "EMVSC-E").
"E2"	Derivation key for an EMV/chip issuer master key: Secure messaging for integrity (maps to TR31 Key Create rule array keyword "EMVSI-E").
"E3"	Derivation key for an EMV/chip issuer master key: Data authentication code (maps to TR31 Key Create rule array keyword "EMVDA-E").
"E4"	Derivation key for an EMV/chip issuer master key: Dynamic numbers (maps to TR31 Key Create rule array keyword "EMVDN-E").
"E5"	Derivation key for an EMV/chip issuer master key: Card personalization (maps to TR31 Key Create rule array keyword "EMVCP-E").
"F0"	EMV/chip issuer master key: Application cryptograms (maps to TR31 Create rule array keyword "EMVAC-F").
"F1"	EMV/chip issuer master key: Secure messaging for confidentiality (maps to TR31 Create rule array keyword "EMVSC-F").
"F2"	EMV/chip issuer master key: Secure messaging for integrity (maps to TR31 Create rule array keyword "EMVSI-F").
"F3"	EMV/chip issuer master key: Data authentication code (maps to TR31 Create rule array keyword "EMVDA-F").
"F4"	EMV/chip issuer master key: Dynamic numbers (maps to TR31 Create rule array keyword "EMVDN-F").
"I0"	Initialization vector (maps to TR31 Export rule array keyword "INITVEC").

Table 79. Key usage values and meanings (continued)

ASCII value	Meaning
"K0"	Key encryption or wrapping key (maps to TR-31 Translate rule array keyword "KEK").
"K1"	TR-31 key block protection key (maps TR-31 Translate rule array keyword "KEKWRAP"). The note for key usage "K0" also applies to this key usage.
"K4"	ISO 20038 key block protection key (maps to TR-31 Translate rule array keyword "KEK-WRK4"). The note for key usage "K0" also applies to this key usage.
"M0"	ISO 16609 MAC algorithm 1 (using TDEA) key (maps to TR-31 Translate rule array keyword "ISOMAC0").
"M1"	ISO 9797-1 MAC algorithm 1 (maps to TR-31 Translate rule array keyword "ISOMAC1"). The note for key usage "M0" also applies to this key usage.
"M3"	ISO 9797-1 MAC algorithm 3 (maps to TR-31 Translate rule array keyword "ISOMAC3"). The note for key usage "M0" also applies to this key usage.
"M6"	ISO 9797-1:2011 MAC algorithm 5/CMAC (maps to TR-31 Translate rule array keyword "ISOMAC6", Release 5.4 or later). These keys are used to compute or verify a code for message authentication.
"M7"	HMAC (maps to TR-31 Translate rule array keyword "HMAC", Release 5.6 or later). These keys are used to compute or verify a code for message authentication.
"P0"	PIN encryption key (maps to TR-31 Translate rule array keyword "PINENC"). These keys are used to protect PIN blocks.
"P2"	PIN generation key (maps to TR-31 Create rule array keyword "PINGEN"). These keys are used to generate PINs.
"V0"	PIN verification, KPV, other algorithm key (maps to TR31 Key Create rule array keyword "PINVO").
"V1"	PIN verification, IBM 3624 key (maps to TR31 Key Create rule array keyword "PINV3624").
"V2"	PIN verification, Visa PVV key (maps to TR31 Key Create rule array keyword "VISAPVV").

X9.143 (TR-31) optional block data defined by IBM

As defined by ANSI X9.143 and ASC X9 TR-31, a TR-31 key block can contain one or more optional blocks. A TR-31 key block contains at least one optional block when bytes 13 - 14 is a value other than ASCII "00".

The data of an IBM-defined optional block contains ASCII string "10" in the first two bytes and contains ASCII string "IBMC" beginning at offset 4 of the data. ICSF treats an optional block with these characteristics as a proprietary container. See [Table 80 on page 326](#) for details. An optional block with different characteristics is ignored.

If a TR-31 key block contains an optional block as defined by [Table 80 on page 326](#), the data contains either a tag-length-value (TLV) ID of '01' or '02'.

For TLV ID '01'

A copy of the 8-byte or 16-byte DES control vector that was in the CCA key-token of the key being exported. The copied control vector is in hex-ASCII format ("0"-"9", "A"-"F").

The control vector is only copied from the CCA key-token when the user of the TR-31 Translate service specifies a control vector transport control keyword (INCL-CV or ATTR-CV):

- If the optional block contains a control vector as the result of specifying the INCL-CV keyword during export, the key usage and mode of use fields indicate the key attributes, and these attributes are verified during export to be compatible with the ones in the included control vector.
- If the optional block contains a control vector as the result of specifying the ATTR-CV keyword during export, the key usage field (byte number 5-6 of the TR-31 key block) is set to the proprietary value "10", and the mode of use field (byte number 8) is set to the proprietary value "1". These proprietary values indicate that the key attributes are specified in the included control vector.

For TLV ID '02'

The data contains the IBM Internal X9-SWKB controls.

For additional information on how CCA uses an IBM-defined optional block in a TR-31 key block, see *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

Table 80. IBM optional block data in a TR-31 key block, control vector (ID "10")		
Offset (bytes)	Length (bytes)	Description
00	02	Proprietary ID of TR-31 optional block (alphanumeric-ASCII). ASCII value Meaning "10" IBM proprietary optional block.
02	02	Length of optional block (hex-ASCII). For TLV valued to "01": ASCII value Meaning "1C" 8-byte (single-length) control vector. "2C" 16-byte (double-length) control vector. For TLV valued to "02": ASCII value Meaning "24" 12-byte IBM internal controls.
Beginning of optional block data		

Table 80. IBM optional block data in a TR-31 key block, control vector (ID "10") (continued)

Offset (bytes)	Length (bytes)	Description
04	04	<p>"Magic" value (alphanumeric-ASCII).</p> <p>ASCII value Meaning</p> <p>"IBMC" A constant value used to reduce ambiguity and the chance for false interpretation of the proprietary optional blocks of non-IBM vendors.</p> <p>An optional block that uses the same proprietary ID, but does not include this magic value will be ignored.</p>
08	02	<p>Tag-length-value (TLV) ID (numeric-ASCII).</p> <p>ASCII value Meaning</p> <p>"01" IBM CCA control vector.</p> <p>"02" IBM Internal X9-SWKB controls.</p>
10	02	<p>Length of TLV (hex-ASCII).</p> <p>For TLV valued to "01":</p> <p>ASCII value Meaning</p> <p>"14" Length of TLV for 8-byte control vector (decimal 20).</p> <p>"24" Length of TLV for 16-byte control vector (decimal 36).</p> <p>For TLV valued to "02":</p> <p>ASCII value Meaning</p> <p>"1C" Length of TLV for 12-byte IBM internal controls (decimal 28).</p>
12	16 or 32	<p>For TLV ID "01": This should be 16 or 32 bytes long and contain the control vector (hex-ASCII).</p>
	24	<p>For TLV ID "02": This should be 24 bytes long and contain the IBM Internal X9-SWKB controls (hex-ASCII). Table 81 on page 328 documents the 12 bytes of controls after conversion to binary:</p> <ul style="list-style-type: none"> • Four bytes (12-15) are controls that apply to all key usages. • Four bytes (16-19) are type specific and only used for certain Key Usages. Unused bytes must be set to zero. • Four bytes (20-23) are reserved and must be set to zero.

Table 81. IBM internal X9-SWKB controls (TLV ID '02') after conversion to binary

Offset (bytes)	Length (bytes)	Description
12	1	General flag byte applicable for key usages. Bit Meaning when set On 0-4 Reserved. 5 Compliance Tagged key block. 6 CPACF export allowed. 7 Only usable in DK services, DKPINOP/KUF equivalent.
13	2	Reserved.
15	1	KDF Indicator. X'00' No KDF / comp-tag indicator. X'04' Comp-tag KDF for TR-31.
16	1	Type specific flags. Bit Meaning when set On 0 All use cases are allowed. Bits 1-7 must be zero. 1-7 Reserved.
17-19	3	Reserved for future use.
20-23	4	Reserved.

PKA key token formats

As with DES key tokens, the first byte of a PKA key token indicates the type of token. If the first byte of the key identifier is X'1E' or X'1F', this indicates that it is a **PKA key token**.

A first byte of X'1E' indicates an external token with a cleartext public key and optionally a private key that is either in cleartext or enciphered by a transport key-encrypting key.

A first byte of X'1F' indicates an internal token with a cleartext public key and a private key that is enciphered by the master key and ready for internal use.

PKA tokens are of variable length because they contain either RSA, ECC, ML-DSA, ML-KEM, CRYSTALS-Dilithium, or CRYSTALS-Kyber key values, which are variable in length. Consequently, length parameters precede all PKA token parameters. The maximum allowed size is 8000 bytes. PKA key tokens consist of a token header, any required sections, and optional sections, which depend on the token type.

A PKA key token can be a public or private key token, and a private key token can be internal or external. Therefore, there are three basic types of tokens, each of which can contain either RSA, ECC, ML-DSA, ML-KEM, CRYSTALS-Dilithium, or CRYSTALS-Kyber information:

- Public key tokens.

- Private external key tokens.
- Private internal key tokens.

Public key tokens contain only the public key. Private key tokens contain the public and private key pair.

Internal PKA tokens

PKA private internal key tokens contain both private and public key information. There is no need for an internal token with only the public key information because the public values are in the clear.

The first byte of X'1F' indicates an internal token with a cleartext public key and a private key that is enciphered with a PKA master key and ready for local (internal) use.

The format of a PKA private internal key token is similar to that of a private external token. The only differences are changes in the private key section and the addition of some internal information at the end of the token. This last section starts with the eyecatcher 'PKTN' rather than with a token or section marker.

PKA null key token

Table 82 on page 329 shows the format for a PKA null key token.

Table 82. Format of PKA Null Key Tokens	
Bytes	Description
0	X'00' Token identifier (indicates that this is a null key token).
1	Version, X'00'
2–3	X'0008' Length of the key token structure.
4–7	Ignored (should be zero).

RSA key token formats

This topic describes the different RSA key token formats.

RSA public key token

An RSA public key token contains the following sections:

- A required token header, starting with the token identifier X'1E'
- A required RSA public key section, starting with the section identifier X'04'

Table 83 on page 329 presents the format of an RSA public key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first.

Table 83. RSA Public Key Token		
Offset (Dec)	Number of Bytes	Description
Token Header (required)		
000	001	Token identifier. X'1E' indicates an external token.
001	001	Version, X'00'.
002	002	Length of the key token structure.
004	004	Ignored. Should be zero.
RSA Public Key Section (required)		

<i>Table 83. RSA Public Key Token (continued)</i>		
Offset (Dec)	Number of Bytes	Description
000	001	X'04', section identifier, RSA public key.
001	001	X'00', version.
002	002	Section length, 12+xxx+yyy.
004	002	Reserved field.
006	002	RSA public key exponent field length in bytes, "xxx".
008	002	Public key modulus length in bits.
010	002	RSA public key modulus field length in bytes, "yyy".
012	xxx	Public key exponent (this is generally a 1-, 3-, or 64- to 512-byte quantity), e. e must be odd and $1 < e < n$. (Frequently, the value of e is $2^{16}+1$)
12+xxx	yyy	Modulus, n.

RSA private external key token

An RSA private external key token contains the following sections:

- A required PKA token header starting with the token identifier X'1E'
- A required RSA private key section starting with one of the following section identifiers:
 - X'02' which indicates a modulus-exponent form RSA private key section (not optimized) with modulus length of up to 1024 bits.
 - X'08' which indicates an optimized Chinese Remainder Theorem form private key section with modulus bit length of up to 4096.
 - X'09' which indicates a modulus-exponent form RSA private key section (not optimized) with modulus length of up to 4096 bits.
 - X'30' which indicates a modulus-exponent form RSA private key section with modulus length of up to 8192 bits with an AES object protection key.
 - X'31' which indicates an Chinese Remainder Theorem form private key section with modulus bit length of up to 8192 bits with an AES object protection key.
- A required RSA public key section, starting with the section identifier X'04'
- An optional private key name section, starting with the section identifier X'10'

Table 84 on page 330 presents the basic record format of an RSA private external key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first. All binary fields (exponents, modulus, and so on) in the private sections of tokens are right-justified and padded with zeros to the left.

<i>Table 84. RSA Private External Key Token Basic Record Format</i>		
Offset (Dec)	Number of Bytes	Description
		Token Header (required)
000	001	Token identifier. X'1E' indicates an external token. The private key is either in cleartext or enciphered with a transport key-encrypting key.
001	001	Version, X'00'.
002	002	Length of the key token structure.
004	004	Ignored. Should be zero.

Table 84. RSA Private External Key Token Basic Record Format (continued)

Offset (Dec)	Number of Bytes	Description
		RSA Private Key Section (required) <ul style="list-style-type: none"> For 1024-bit Modulus-Exponent form refer to “RSA private key token, 1024-bit modulus-exponent external format” on page 332. For 4096-bit Modulus-Exponent form refer to “RSA private key token, 4096-bit modulus-exponent external format” on page 333. For 4096-bit Chinese Remainder Theorem form refer to “RSA private key token, 4096-bit Chinese Remainder Theorem external format” on page 334. For 4096-bit Modulus-Exponent form with AES OPK refer to “RSA private key, 8192-bit modulus-exponent format with AES encrypted OPK section (X'30') external form” on page 336. For 4096-bit Chinese Remainder Theorem form with AES OPK refer to “RSA private key, 8192-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') external form” on page 340.
		RSA Public Key Section (required)
000	001	X'04', section identifier, RSA public key.
001	001	X'00', version.
002	002	Section length, 12+xxx.
004	002	Reserved field.
006	002	RSA public key exponent field length in bytes, "xxx".
008	002	Public key modulus length in bits.
010	002	RSA public key modulus field length in bytes, which is zero for a private token. Note: In an RSA private key token, this field should be zero. The RSA private key section contains the modulus.
012	xxx	Public key exponent, e (this is generally a 1-, 3-, or 64- to 512-byte quantity). e must be odd and $1 < e < n$. (Frequently, the value of e is $2^{16}+1$ (=65,537).)
		Private Key Name (optional)
000	001	X'10', section identifier, private key name.
001	001	X'00', version.
002	002	Section length, X'0044' (68 decimal).
004	064	Private key name (in ASCII), left-justified, padded with space characters (X'20'). An access control system can use the private key name to verify that the calling application is entitled to use the key.

RSA private key token, 1024-bit modulus-exponent external format

Table 85. RSA Private Key Token, 1024-bit Modulus-Exponent external format		
Offset (Dec)	Number of Bytes	Description
000	001	X'02', section identifier, RSA private key, modulus-exponent format (RSA-PRIV)
001	001	X'00', version.
002	002	Length of the RSA private key section X'016C' (364 decimal).
004	020	SHA-1 hash value of the private key subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.
024	004	Reserved; set to binary zero.
028	001	Key format and security: X'00' Unencrypted RSA private key subsection identifier. X'82' Encrypted RSA private key subsection identifier.
029	001	Reserved, binary zero.
030	020	SHA-1 hash of the optional key-name section. If there is no key-name section, then 20 bytes of X'00'.
050	004	Key use flag bits. Bit Meaning When Set On 0 Key management usage permitted. 1 Signature usage not permitted. 6 The key is translatable. All other bits reserved, set to binary zero.
054	006	Reserved; set to binary zero.
060	024	Reserved; set to binary zero.
		Start of the optionally-encrypted secure subsection.
084	024	Random number, confounder.
108	128	Private-key exponent, d. $d = e^{-1} \bmod ((p-1)(q-1))$, and $1 < d < n$ where e is the public exponent.
		End of the optionally-encrypted subsection; the confounder field and the private-key exponent field are enciphered for key confidentiality when the key format and security flags (offset 28) indicate that the private key is enciphered. They are enciphered under a double-length transport key using the ede2 algorithm.
236	128	Modulus, n. $n = pq$ where p and q are prime and $1 < n < 2^{1024}$.

RSA private key token, 4096-bit modulus-exponent external format

This RSA private key token and the external X'09' token is supported on a CCA Crypto Express coprocessor.

Table 86. RSA Private Key Token, 4096-bit Modulus-Exponent external format		
Offset (Dec)	Number of Bytes	Description
000	001	X'09', section identifier, RSA private key, modulus-exponent format (RSAMEVAR).
001	001	X'00', version.
002	002	Length of the RSA private key section 132+ddd+nnn+xxx.
004	020	SHA-1 hash value of the private key subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.
024	002	Length of the encrypted private key section 8+ddd+xxx.
026	002	Reserved; set to binary zero.
028	001	Key format and security: X'00' Unencrypted RSA private key subsection identifier. X'82' Encrypted RSA private key subsection identifier.
029	001	Reserved, set to binary zero.
030	020	SHA-1 hash of the optional key-name section. If there is no key-name section, then 20 bytes of X'00'.
050	001	Key use flag bits. Bit Meaning When Set On 0 Key management usage permitted. 1 Signature usage not permitted. 6 The key is translatable All other bits reserved, set to binary zero.
051	001	Reserved; set to binary zero.
052	048	Reserved; set to binary zero.
100	016	Reserved; set to binary zero.
116	002	Length of private exponent, d, in bytes: ddd.
118	002	Length of modulus, n, in bytes: nnn.
120	002	Length of padding field, in bytes: xxx.
122	002	Reserved; set to binary zero.
		Start of the optionally-encrypted secure subsection.
124	008	Random number, confounder.

Table 86. RSA Private Key Token, 4096-bit Modulus-Exponent external format (continued)		
Offset (Dec)	Number of Bytes	Description
132	ddd	Private-key exponent, d. $d = e^{-1} \bmod ((p-1)(q-1))$, and $1 < d < n$ where e is the public exponent.
132+ddd	xxx	X'00' padding of length xxx bytes such that the length from the start of the random number above to the end of the padding field is a multiple of eight bytes.
		End of the optionally-encrypted subsection; the confounder field and the private-key exponent field are enciphered for key confidentiality when the key format and security flags (offset 28) indicate that the private key is enciphered. They are enciphered under a double-length transport key using the ede2 algorithm.
132+ddd+xxx	nnn	Modulus, n. $n = pq$ where p and q are prime and $1 < n < 2^{4096}$.

RSA private key token, 4096-bit Chinese Remainder Theorem external format

This RSA private key token with up to 2048-bit modulus is supported on all coprocessors. The modulus size is increased to 4096-bit on IBM z9 EC, IBM z9 BC, IBM z10 EC, IBM z10 BC, or later machines with the Nov. 2007 or later version of the licensed internal code installed on the CCA Crypto Express coprocessor.

Table 87. RSA Private Key Token, 4096-bit Chinese Remainder Theorem external format		
Offset (Dec)	Number of Bytes	Description
000	001	X'08', section identifier, RSA private key, CRT format (RSA-CRT)
001	001	X'00', version.
002	002	Length of the RSA private-key section, $132 + ppp + qqq + rrr + sss + uuu + xxx + nnn$.
004	020	SHA-1 hash value of the private key subsection cleartext, offset 28 to the end of the modulus.
024	004	Reserved; set to binary zero.
028	001	Key format and security: X'40' Unencrypted RSA private-key subsection identifier, Chinese Remainder form. X'42' Encrypted RSA private-key subsection identifier, Chinese Remainder form.
029	001	Reserved; set to binary zero.
030	020	SHA-1 hash of the optional key-name section and any following optional sections. If there are no optional sections, then 20 bytes of X'00'.

Table 87. RSA Private Key Token, 4096-bit Chinese Remainder Theorem external format (continued)

Offset (Dec)	Number of Bytes	Description
050	004	Key use flag bits. Bit Meaning When Set On 0 Key management usage permitted. 1 Signature usage not permitted. 6 The key is translatable. All other bits reserved, set to binary zero.
054	002	Length of prime number, p, in bytes: ppp.
056	002	Length of prime number, q, in bytes: qqg.
058	002	Length of d _p , in bytes: rrr.
060	002	Length of d _q , in bytes: sss.
062	002	Length of U, in bytes: uuu.
064	002	Length of modulus, n, in bytes: nnn.
066	004	Reserved; set to binary zero.
070	002	Length of padding field, in bytes: xxx.
072	004	Reserved, set to binary zero.
076	016	Reserved, set to binary zero.
092	032	Reserved; set to binary zero.
		Start of the optionally-encrypted secure subsection.
124	008	Random number, confounder.
132	ppp	Prime number, p.
132 + ppp	qqq	Prime number, q
132 + ppp + qqg	rrr	$d_p = d \bmod (p - 1)$
132 + ppp + qqg + rrr	sss	$d_q = d \bmod (q - 1)$
132 + ppp + qqg + rrr + sss	uuu	$U = q^{-1} \bmod (p)$.
132 + ppp + qqg + rrr + sss + uuu	xxx	X'00' padding of length xxx bytes such that the length from the start of the random number above to the end of the padding field is a multiple of eight bytes.
		End of the optionally-encrypted secure subsection; all of the fields starting with the confounder field and ending with the variable length pad field are enciphered for key confidentiality when the key format-and-security flags (offset 28) indicate that the private key is enciphered. They are enciphered under a double-length transport key using the TDES (CBC outer chaining) algorithm.

Table 87. RSA Private Key Token, 4096-bit Chinese Remainder Theorem external format (continued)

Offset (Dec)	Number of Bytes	Description
132 + ppp + qqg + rrr + sss + uuu + xxx	nnn	Modulus, n. $n = pq$ where p and q are prime and $1 < n < 2^{4096}$.

RSA private key, 8192-bit modulus-exponent format with AES encrypted OPK section (X'30') external form

This RSA private key token is supported on Crypto Express3 and later coprocessors. Associated data version 4 is supported on Crypto Express6 and later coprocessors with the July 2019 or later licensed internal code (LIC).

Table 88. RSA private key, 8192-bit Modulus-Exponent format with AES encrypted OPK section (X'30') external form

Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: X'30' RSA private key, ME format with AES encrypted OPK.
001	001	Section version number (X'00').
002	002	Section length: 122 + nnn + ppp
004	002	Length of "Associated Data" section
006	002	Length of payload data: ppp
008	002	Reserved, binary zero.
		Start of Associated Data
010	001	Associated Data Version: X'02' Version 2 X'04' Version 4
011	001	Key format and security flag: X'00' Unencrypted ME RSA private-key subsection identifier X'82' Encrypted ME RSA private-key subsection identifier
012	001	Key source flag: Reserved, binary zero.

Table 88. RSA private key, 8192-bit Modulus-Exponent format with AES encrypted OPK section (X'30') external form (continued)

Offset (bytes)	Length (bytes)	Description
013	001	<p>When associated data section version is X'02': Reserved, binary zero.</p> <p>When associated data section version is X'04': Compliance and export control byte.</p> <p>Bit</p> <p>Meaning</p> <p>B'1xxx xxxx' Compliant-tagged key.</p> <p>B'0xxx xxxx' Non-compliant-tagged key.</p> <p>B'xxxx xx1x' Private key translation is allowed (XLATE-OK).</p> <p>B'xxxx xx0x' Private key translation is not allowed (NO-XLATE).</p> <p>All other bits are reserved and must be zero.</p>
014	001	<p>Hash type:</p> <p>X'00' Clear key</p> <p>X'02' SHA-256</p>
015	032	<p>When associated data section version is X'02': SHA-256 hash of all optional sections that follow the public key section, if any. Otherwise, 32 bytes of binary zero.</p> <p>When associated data section version is X'04': Hash value of:</p> <ol style="list-style-type: none"> 1. The public key section (section identifier X'04') and 2. All optional sections that follow the public key section, if any. <p>If there are no optional sections, the hash covers only the public keys section.</p>
047	001	Reserved, binary zero.

Table 88. RSA private key, 8192-bit Modulus-Exponent format with AES encrypted OPK section (X'30') external form (continued)

Offset (bytes)	Length (bytes)	Description																						
048	002	<p>When associated data section version is X'02': Reserved, binary zero.</p> <p>When associated data section version is X'04':</p> <p>Usage bytes:</p> <ul style="list-style-type: none">Offset 48:<div><table><tr><th>Bit</th><th>Meaning</th></tr><tr><td>B'1xxx xxxx'</td><td>Digital Signature usage is allowed (U-DIGSIG). Services: CSNDDSG, CSNDDSV, CSNDT34B, CSNDT34D.</td></tr><tr><td>B'x1xx xxxx'</td><td>Non-Repudiation usage is allowed (U-NONRPD). Services: CSNDDSG, CSNDDSV.</td></tr><tr><td>B'xx1x xxxx'</td><td>Key Encipherment usage is allowed (U-KEYENC). Services: CSNDSYG, CSNDSYX, CSNDSYI, CSNDSYI2, CSNDT34R, CSNDPKE, CSNDPKD.</td></tr><tr><td>B'xxx1 xxxx'</td><td>Data Encipherment usage is allowed (U-DATENC). Services: CSNDPKE, CSNDPKD.</td></tr><tr><td>B'xxxx 1xxx'</td><td>Key agreement usage is allowed (U-KEYAGR).</td></tr><tr><td>B'xxxx x1xx'</td><td>keyCertSign usage is allowed (U-KCRTSN). Services: CSNDDSG, CSNDDSV.</td></tr><tr><td>B'xxxx xx1x'</td><td>Certificate Revocation List Sign usage is allowed (U-CRLSN). Services: CSNDDSG, CSNDDSV.</td></tr><tr><td>B'xxxx xxx1'</td><td>Only encipher operations are allowed during key agreement (U-ENCONL).</td></tr></table></div>Offset 49:<div><table><tr><th>Bit</th><th>Meaning</th></tr><tr><td>B'1xxx xxxx'</td><td>Only decipher operations are allowed during key agreement (U-DECONL).</td></tr></table></div>	Bit	Meaning	B'1xxx xxxx'	Digital Signature usage is allowed (U-DIGSIG). Services: CSNDDSG, CSNDDSV, CSNDT34B, CSNDT34D.	B'x1xx xxxx'	Non-Repudiation usage is allowed (U-NONRPD). Services: CSNDDSG, CSNDDSV.	B'xx1x xxxx'	Key Encipherment usage is allowed (U-KEYENC). Services: CSNDSYG, CSNDSYX, CSNDSYI, CSNDSYI2, CSNDT34R, CSNDPKE, CSNDPKD.	B'xxx1 xxxx'	Data Encipherment usage is allowed (U-DATENC). Services: CSNDPKE, CSNDPKD.	B'xxxx 1xxx'	Key agreement usage is allowed (U-KEYAGR).	B'xxxx x1xx'	keyCertSign usage is allowed (U-KCRTSN). Services: CSNDDSG, CSNDDSV.	B'xxxx xx1x'	Certificate Revocation List Sign usage is allowed (U-CRLSN). Services: CSNDDSG, CSNDDSV.	B'xxxx xxx1'	Only encipher operations are allowed during key agreement (U-ENCONL).	Bit	Meaning	B'1xxx xxxx'	Only decipher operations are allowed during key agreement (U-DECONL).
Bit	Meaning																							
B'1xxx xxxx'	Digital Signature usage is allowed (U-DIGSIG). Services: CSNDDSG, CSNDDSV, CSNDT34B, CSNDT34D.																							
B'x1xx xxxx'	Non-Repudiation usage is allowed (U-NONRPD). Services: CSNDDSG, CSNDDSV.																							
B'xx1x xxxx'	Key Encipherment usage is allowed (U-KEYENC). Services: CSNDSYG, CSNDSYX, CSNDSYI, CSNDSYI2, CSNDT34R, CSNDPKE, CSNDPKD.																							
B'xxx1 xxxx'	Data Encipherment usage is allowed (U-DATENC). Services: CSNDPKE, CSNDPKD.																							
B'xxxx 1xxx'	Key agreement usage is allowed (U-KEYAGR).																							
B'xxxx x1xx'	keyCertSign usage is allowed (U-KCRTSN). Services: CSNDDSG, CSNDDSV.																							
B'xxxx xx1x'	Certificate Revocation List Sign usage is allowed (U-CRLSN). Services: CSNDDSG, CSNDDSV.																							
B'xxxx xxx1'	Only encipher operations are allowed during key agreement (U-ENCONL).																							
Bit	Meaning																							
B'1xxx xxxx'	Only decipher operations are allowed during key agreement (U-DECONL).																							

Table 88. RSA private key, 8192-bit Modulus-Exponent format with AES encrypted OPK section (X'30') external form (continued)

Offset (bytes)	Length (bytes)	Description
050	001	<p>When associated data section version is X'02': Key-usage and translation control flag:</p> <p>Key-usage:</p> <p>B'11xx xxxx' Only key unwrapping (KM-ONLY).</p> <p>B'10xx xxxx' Both signature generation and key unwrapping (KEY-MGMT).</p> <p>B'01xx xxxx' Undefined.</p> <p>B'00xx xxxx' Only signature generation (SIG-ONLY).</p> <p>All other values are undefined.</p> <p>Translation control bit:</p> <p>B'xxxx xx1x' Private key translation is allowed (XLATE-OK).</p> <p>B'xxxx xx0x' Private key translation is not allowed (NO-XLATE).</p> <p>All other bits are reserved and must be zero.</p> <p>When associated data section version is X'04': Reserved, binary zero.</p>
051	001	<p>Format restriction byte for digital-signature hash formatting method.</p> <p>Value:</p> <p>B'0000 0000' No format restriction.</p> <p>B'0000 0001' ISO-9796 only.</p> <p>B'0000 0010' PKCS-1.0 only.</p> <p>B'0000 0011' PKCS-1.1 only.</p> <p>B'0000 0100' PKCS-PSS only.</p> <p>B'0000 0101' X9.31 only.</p> <p>B'0000 0110' ZERO-PAD only.</p> <p>All other values are reserved and undefined.</p>
052	002	Length of modulus: nnn bytes
054	002	Length of private exponent: ddd bytes
		End of Associated Data
056	048	<p>16 byte confounder + 32-byte Object Protection Key.</p> <p>OPK used as an AES key.</p> <p>encrypted with an AES KEK.</p>

Table 88. RSA private key, 8192-bit Modulus-Exponent format with AES encrypted OPK section (X'30') external form (continued)

Offset (bytes)	Length (bytes)	Description
104	016	Key verification pattern <ul style="list-style-type: none"> For an encrypted private key, KEK verification pattern (KVP) For a clear private key, binary zeros For a skeleton, binary zeros
120	002	Reserved, binary zeros.
122	nnn	Modulus
122+nnn	ppp	Payload starts here and includes: When this section is unencrypted: <ul style="list-style-type: none"> Clear private exponent d. Length ppp bytes : ddd + 0 When this section is encrypted: <ul style="list-style-type: none"> Private exponent d within the AESKW-wrapped payload. Length ppp bytes : ddd + AESKW format overhead

RSA private key, 8192-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') external form

This RSA private key token is supported on Crypto Express3 and later coprocessors. Associated data version 5 is supported on Crypto Express6 and later coprocessors with the July 2019 or later licensed internal code (LIC).

Table 89. RSA private key, 8192-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') external form

Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: X'31' RSA private key, CRT format with AES encrypted OPK
001	001	Section version number (X'00').
002	002	Section length: 134 + nnn + xxx
004	002	Length of "Associated Data" section
006	002	Length of payload data: xxx
008	002	Reserved, binary zero.
		Start of Associated Data
010	001	Associated Data Version: X'03' Version 3 X'05' Version 5

Table 89. RSA private key, 8192-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') external form (continued)

Offset (bytes)	Length (bytes)	Description
011	001	Key format and security flag: X'40' Unencrypted RSA private-key subsection identifier X'42' Encrypted RSA private-key subsection identifier
012	001	Key source flag: Reserved, binary zero.
013	001	When associated data section version is X'03': Reserved, binary zero. When associated data section version is X'05': Compliance and export control byte. Bit Meaning B'1xxx xxxx' Compliant-tagged key. B'0xxx xxxx' Non-compliant-tagged key. B'xxxx xx1x' Private key translation is allowed (XLATE-OK). B'xxxx xx0x' Private key translation is not allowed (NO-XLATE). All other bits are reserved and must be zero.
014	001	Hash type: X'00' Clear key X'02' SHA-256
015	032	When associated data section version is X'03': SHA-256 hash of all optional sections that follow the public key section, if any. Otherwise, 32 bytes of binary zero. When associated data section version is X'05': Hash value of: 1. The public key section (section identifier X'04') and 2. All optional sections that follow the public key section, if any. If there are no optional sections, the hash covers only the public keys section.
047	001	Reserved, binary zero.

Table 89. RSA private key, 8192-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') external form (continued)

Offset (bytes)	Length (bytes)	Description																						
048	002	<p>When associated data section version is X'03': Reserved, binary zero.</p> <p>When associated data section version is X'05':</p> <p>Usage bytes:</p> <ul style="list-style-type: none">Offset 48:<table><tr><th>Bit</th><th>Meaning</th></tr><tr><td>B'1xxx xxxx'</td><td>Digital Signature usage is allowed (U-DIGSIG). Services: CSNDDSG, CSNDDSV, CSNDT34B, CSNDT34D.</td></tr><tr><td>B'x1xx xxxx'</td><td>Non-Repudiation usage is allowed (U-NONRPD). Services: CSNDDSG, CSNDDSV.</td></tr><tr><td>B'xx1x xxxx'</td><td>Key Encipherment usage is allowed (U-KEYENC). Services: CSNDSYG, CSNDSYX, CSNDSYI, CSNDSYI2, CSNDT34R, CSNDPKE, CSNDPKD.</td></tr><tr><td>B'xxx1 xxxx'</td><td>Data Encipherment usage is allowed (U-DATENC). Services: CSNDPKE, CSNDPKD.</td></tr><tr><td>B'xxxx 1xxx'</td><td>Key agreement usage is allowed (U-KEYAGR).</td></tr><tr><td>B'xxxx x1xx'</td><td>keyCertSign usage is allowed (U-KCRTSN). Services: CSNDDSG, CSNDDSV.</td></tr><tr><td>B'xxxx xx1x'</td><td>Certificate Revocation List Sign usage is allowed (U-CRLSN). Services: CSNDDSG, CSNDDSV.</td></tr><tr><td>B'xxxx xxx1'</td><td>Only encipher operations are allowed during key agreement (U-ENCONL).</td></tr></table>Offset 49:<table><tr><th>Bit</th><th>Meaning</th></tr><tr><td>B'1xxx xxxx'</td><td>Only decipher operations are allowed during key agreement (U-DECONL).</td></tr></table>	Bit	Meaning	B'1xxx xxxx'	Digital Signature usage is allowed (U-DIGSIG). Services: CSNDDSG, CSNDDSV, CSNDT34B, CSNDT34D.	B'x1xx xxxx'	Non-Repudiation usage is allowed (U-NONRPD). Services: CSNDDSG, CSNDDSV.	B'xx1x xxxx'	Key Encipherment usage is allowed (U-KEYENC). Services: CSNDSYG, CSNDSYX, CSNDSYI, CSNDSYI2, CSNDT34R, CSNDPKE, CSNDPKD.	B'xxx1 xxxx'	Data Encipherment usage is allowed (U-DATENC). Services: CSNDPKE, CSNDPKD.	B'xxxx 1xxx'	Key agreement usage is allowed (U-KEYAGR).	B'xxxx x1xx'	keyCertSign usage is allowed (U-KCRTSN). Services: CSNDDSG, CSNDDSV.	B'xxxx xx1x'	Certificate Revocation List Sign usage is allowed (U-CRLSN). Services: CSNDDSG, CSNDDSV.	B'xxxx xxx1'	Only encipher operations are allowed during key agreement (U-ENCONL).	Bit	Meaning	B'1xxx xxxx'	Only decipher operations are allowed during key agreement (U-DECONL).
Bit	Meaning																							
B'1xxx xxxx'	Digital Signature usage is allowed (U-DIGSIG). Services: CSNDDSG, CSNDDSV, CSNDT34B, CSNDT34D.																							
B'x1xx xxxx'	Non-Repudiation usage is allowed (U-NONRPD). Services: CSNDDSG, CSNDDSV.																							
B'xx1x xxxx'	Key Encipherment usage is allowed (U-KEYENC). Services: CSNDSYG, CSNDSYX, CSNDSYI, CSNDSYI2, CSNDT34R, CSNDPKE, CSNDPKD.																							
B'xxx1 xxxx'	Data Encipherment usage is allowed (U-DATENC). Services: CSNDPKE, CSNDPKD.																							
B'xxxx 1xxx'	Key agreement usage is allowed (U-KEYAGR).																							
B'xxxx x1xx'	keyCertSign usage is allowed (U-KCRTSN). Services: CSNDDSG, CSNDDSV.																							
B'xxxx xx1x'	Certificate Revocation List Sign usage is allowed (U-CRLSN). Services: CSNDDSG, CSNDDSV.																							
B'xxxx xxx1'	Only encipher operations are allowed during key agreement (U-ENCONL).																							
Bit	Meaning																							
B'1xxx xxxx'	Only decipher operations are allowed during key agreement (U-DECONL).																							

Table 89. RSA private key, 8192-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') external form (continued)

Offset (bytes)	Length (bytes)	Description
050	001	<p>When associated data section version is X'03': Key-usage and translation control flag.</p> <p>Key-usage flag:</p> <p>B'11xx xxxx' Only key unwrapping (KM-ONLY).</p> <p>B'10xx xxxx' Both signature generation and key unwrapping (KEY-MGMT).</p> <p>B'01xx xxxx' Undefined.</p> <p>B'00xx xxxx' Only signature generation (SIG-ONLY).</p> <p>All other values are undefined.</p> <p>Translation control bit:</p> <p>B'xxxx xx1x' Private key translation is allowed (XLATE-OK).</p> <p>B'xxxx xx0x' Private key translation is not allowed (NO-XLATE).</p> <p>All other values are reserved and undefined.</p> <p>When associated data section version is X'05': Reserved, binary zero.</p>
051	001	<p>Format restriction byte for digital-signature hash formatting method.</p> <p>Value:</p> <p>B'0000 0000' No format restriction.</p> <p>B'0000 0001' ISO-9796 only.</p> <p>B'0000 0010' PKCS-1.0 only.</p> <p>B'0000 0011' PKCS-1.1 only.</p> <p>B'0000 0100' PKCS-PSS only.</p> <p>B'0000 0101' X9.31 only.</p> <p>B'0000 0110' ZERO-PAD only.</p> <p>All other values are reserved and undefined.</p>
052	002	Length of the prime number, p, in bytes: ppp.
054	002	Length of the prime number, q, in bytes: qqq
056	002	Length of dp : rrr.
058	002	Length of dq : sss.
060	002	Length of U: uu.
062	002	Length of modulus, nnn.
064	002	Reserved, binary zero.
066	002	Reserved, binary zero.

Table 89. RSA private key, 8192-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') external form (continued)

Offset (bytes)	Length (bytes)	Description
		End of Associated Data
068	048	16 byte confounder + 32-byte Object Protection Key. OPK used as an AES key. External tokens: Encrypted with an AES KEK. Internal tokens: Encrypted with the ECC master key.
116	016	Key verification pattern <ul style="list-style-type: none"> For an encrypted private key, KEK verification pattern (KVP). For a clear private key, binary zeros. For a skeleton, binary zero.
132	002	Reserved, binary zero.
134	nnn	Modulus, n, $n=pq$, where p and q are prime.
134+nnn	xxx	Payload starts here and includes: When this section is unencrypted: <ul style="list-style-type: none"> Clear prime number p Clear prime number q Clear dp Clear dq Clear U Length xxx bytes: $ppp + qq + rrr + sss + uuu + 0$ When this section is encrypted: <ul style="list-style-type: none"> prime number p prime number q dp dq U within the AESKW-wrapped payload. Length xxx bytes : $ppp + qq + rrr + sss + uuu + \text{AESKW format overhead}$

RSA private internal key token

An RSA private internal key token contains the following sections:

- A required PKA token header, starting with the token identifier X'1F'
- Basic record format of an RSA private internal key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first. All binary fields (exponents, modulus, and so on) in the private sections of tokens are right-justified and padded with zeros to the left.

Table 90. RSA Private Internal Key Token Basic Record Format		
Offset (Dec)	Number of Bytes	Description
Token Header (required)		
000	001	Token identifier. X'1F' indicates an internal token. The private key is enciphered with a PKA master key.
001	001	Version, X'00'.
002	002	Length of the key token structure excluding the internal information section.
004	004	Ignored; should be zero.
RSA Private Key Section and Secured Subsection (required)		
<ul style="list-style-type: none"> For 1024-bit X'02' Modulus-Exponent form, refer to “RSA private key token, 1024-bit X'02' modulus-exponent internal form” on page 346. For 1024-bit X'06' Modulus-Exponent form, refer to “RSA private key token, 1024-bit X'06' modulus-exponent internal form” on page 347. For 4096-bit X'08' Chinese Remainder Theorem form, refer to “RSA private key token, 4096-bit Chinese Remainder Theorem internal form” on page 357. For 4096-bit Modulus-Exponent form with AES OPK, refer to “RSA private key, 8192-bit modulus-exponent format with AES encrypted OPK section (X'30') internal form” on page 349. For 4096-bit Chinese Remainder Theorem form with AES OPK, refer to Table 94 on page 353. 		
RSA Public Key Section (required)		
000	001	X'04', section identifier, RSA public key.
001	001	X'00', version.
002	002	Section length, 12+xxx.
004	002	Reserved field.
006	002	RSA public key exponent field length in bytes, "xxx".
008	002	Public key modulus length in bits.
010	002	RSA public key modulus field length in bytes, which is zero for a private token.
012	xxx	Public key exponent (this is generally a 1, 3, or 64 to 512 byte quantity), e. e must be odd and $1 < e < n$. (Frequently, the value of e is $2^{16}+1$ (=65,537).
Private Key Name (optional)		
000	001	X'10', section identifier, private key name.
001	001	X'00', version.
002	002	Section length, X'0044' (68 decimal).
004	064	Private key name (in ASCII), left-justified, padded with space characters (X'20'). An access control system can use the private key name to verify that the calling application is entitled to use the key.
Internal Information Section (required)		
000	004	Eye catcher 'PKTN'.

Table 90. RSA Private Internal Key Token Basic Record Format (continued)

Offset (Dec)	Number of Bytes	Description
004	004	PKA token type. Bit Meaning When Set On 0 RSA key. 1 DSS key. 2 Private key. 3 Public key. 4 Private key name section exists. 5 Private key unenciphered. 6 Blinding information present. 7 Retained private key.
008	004	Address of token header.
012	002	Total length of total structure including this information section.
014	002	Count of number of sections.
016	016	PKA master key hash pattern.
032	001	Domain of retained key.
033	008	Serial number of processor holding retained key.
041	007	Reserved.

RSA private key token, 1024-bit X'02' modulus-exponent internal form

Table 91. RSA Private Internal Key Token, 1024-bit X'02' ME Form

Offset (Dec)	Number of Bytes	Description
000	001	X'02', section identifier, RSA private key.
001	001	X'00', version.
002	002	Length of the RSA private key section X'016C' (364 decimal).
004	020	SHA-1 hash value of the private key subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.
024	004	Reserved; set to binary zero.
028	001	Key format and security: X'02' RSA private key.

Table 91. RSA Private Internal Key Token, 1024-bit X'02' ME Form (continued)		
Offset (Dec)	Number of Bytes	Description
029	001	Format of external key from which this token was derived: X'21' External private key was specified in the clear. X'22' External private key was encrypted.
030	020	SHA-1 hash of the key token structure contents that follow the public key section. If no sections follow, this field is set to binary zeros.
050	001	Key use flag bits. Bit Meaning When Set On 0 Key management usage permitted. 1 Signature usage not permitted. All other bits reserved, set to binary zero.
051	009	Reserved; set to binary zero.
060	048	Object Protection Key (OPK) encrypted under the RSA-MK.
108	128	Secret key exponent d, encrypted under the OPK. $d=e^{-1} \bmod((p-1)(q-1))$
236	128	Modulus, n. $n=pq$ where p and q are prime and $1 < n < 2^{1024}$.

RSA private key token, 1024-bit X'06' modulus-exponent internal form

Table 92. RSA Private Internal Key Token, 1024-bit X'06' ME Form		
Offset (Dec)	Number of Bytes	Description
000	001	X'06', section identifier, RSA private key modulus-exponent format (RSA-PRIV).
001	001	X'00', version.
002	002	Length of the RSA private key section X'0198' (408 decimal) + rrr + iii + xxx.
004	020	SHA-1 hash value of the private key subsection cleartext, offset 28 to and including the modulus at offset 236.
024	004	Reserved; set to binary zero.
028	001	Key format and security: X'02' RSA private key.

Table 92. RSA Private Internal Key Token, 1024-bit X'06' ME Form (continued)		
Offset (Dec)	Number of Bytes	Description
029	001	Format of external key from which this token was derived: X'21' External private key was specified in the clear. X'22' External private key was encrypted. X'23' Private key was generated using regeneration data. X'24' Private key was randomly generated.
030	020	SHA-1 hash of the optional key-name section and any following optional sections. If there are no optional sections, this field is set to binary zeros.
050	004	Key use flag bits. Bit Meaning When Set On 0 Key management usage permitted. 1 Signature usage not permitted. All other bits reserved, set to binary zeros.
054	006	Reserved; set to binary zero.
060	048	Object Protection Key (OPK) encrypted under the RSA-MK using the ede3 algorithm.
108	128	Private key exponent d, encrypted under the OPK using the ede5 algorithm. $d=e^{-1} \bmod ((p-1)(q-1))$, and $1 < d < n$ where e is the public exponent.
236	128	Modulus, n. $n=pq$ where p and q are prime and $2^{512} < n < 2^{1024}$.
364	016	RSA master key verification pattern
380	020	SHA-1 hash value of the blinding information subsection cleartext, offset 400 to the end of the section.
400	002	Length of the random number r, in bytes: rrr.
402	002	Length of the random number r^{-1} , in bytes: iii.
404	002	Length of the padding field, in bytes: xxx.
406	002	Reserved; set to binary zeros.
408	Start of the encrypted blinding subsection	
408	rrr	Random number r (used in blinding).
408 + rrr	iii	Random number r^{-1} (used in blinding).
408 + rrr + iii	xxx	X'00' padding of length xxx bytes such that the length from the start of the encrypted blinding subsection to the end of the padding field is a multiple of eight bytes.

Table 92. RSA Private Internal Key Token, 1024-bit X'06' ME Form (continued)		
Offset (Dec)	Number of Bytes	Description
		End of the encrypted blinding subsection; all of the fields starting with the random number r and ending with the variable length pad field are encrypted under the OPK using TDES (CBC outer chaining) algorithm.

RSA private key, 8192-bit modulus-exponent format with AES encrypted OPK section (X'30') internal form

This RSA private key token is supported on Crypto Express3 and later coprocessors. Associated data version 4 is supported on Crypto Express6 and later coprocessors with the July 2019 or later licensed internal code (LIC).

Table 93. RSA private key, 8192-bit Modulus-Exponent format with AES encrypted OPK section (X'30') internal form		
Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: X'30' RSA private key, ME format with AES encrypted OPK.
001	001	Section version number (X'00').
002	002	Section length: 122 + nnn + ppp
004	002	Length of "Associated Data" section
006	002	Length of payload data: ppp
008	002	Reserved, binary zero.
		Start of Associated Data
010	001	Associated Data Version: X'02' Version 2 X'04' Version 4
011	001	Key format and security flag: X'02' Encrypted ME RSA private-key subsection identifier
012	001	Key source flag: Internal tokens: X'21' Imported from cleartext X'22' Imported from ciphertext X'23' Generated using regeneration data X'24' Randomly generated

Table 93. RSA private key, 8192-bit Modulus-Exponent format with AES encrypted OPK section (X'30') internal form (continued)

Offset (bytes)	Length (bytes)	Description
013	001	<p>When associated data section version is X'02': Reserved, binary zero.</p> <p>When associated data section version is X'04': Compliance and export control byte.</p> <p>Bit</p> <p>Meaning</p> <p>B'1xxx xxxx' Compliant-tagged key.</p> <p>B'0xxx xxxx' Non-compliant-tagged key.</p> <p>B'xxxx xx1x' Private key translation is allowed (XLATE-OK).</p> <p>B'xxxx xx0x' Private key translation is not allowed (NO-XLATE).</p> <p>All other bits are reserved and must be zero.</p>
014	001	<p>Hash type:</p> <p>X'00' Clear key</p> <p>X'02' SHA-256</p>
015	032	<p>When associated data section version is X'02': SHA-256 hash of all optional sections that follow the public key section, if any. Otherwise, 32 bytes of binary zero.</p> <p>When associated data section version is X'04': Hash value of:</p> <ol style="list-style-type: none"> 1. The public key section (section identifier X'04') and 2. All optional sections that follow the public key section, if any. <p>If there are no optional sections, the hash covers only the public keys section.</p>
047	001	Reserved, binary zero.

Table 93. RSA private key, 8192-bit Modulus-Exponent format with AES encrypted OPK section (X'30') internal form (continued)

Offset (bytes)	Length (bytes)	Description																						
048	002	<p>When associated data section version is X'02': Reserved, binary zero.</p> <p>When associated data section version is X'04':</p> <p>Usage bytes:</p> <ul style="list-style-type: none">Offset 48:<table><tr><th>Bit</th><th>Meaning</th></tr><tr><td>B'1xxx xxxx'</td><td>Digital Signature usage is allowed (U-DIGSIG). Services: CSNDDSG, CSNDDSV, CSNDT34B, CSNDT34D.</td></tr><tr><td>B'x1xx xxxx'</td><td>Non-Repudiation usage is allowed (U-NONRPD). Services: CSNDDSG, CSNDDSV.</td></tr><tr><td>B'xx1x xxxx'</td><td>Key Encipherment usage is allowed (U-KEYENC). Services: CSNDSYG, CSNDSYX, CSNDSYI, CSNDSYI2, CSNDT34R, CSNDPKE, CSNDPKD.</td></tr><tr><td>B'xxx1 xxxx'</td><td>Data Encipherment usage is allowed (U-DATENC). Services: CSNDPKE, CSNDPKD.</td></tr><tr><td>B'xxxx 1xxx'</td><td>Key agreement usage is allowed (U-KEYAGR).</td></tr><tr><td>B'xxxx x1xx'</td><td>keyCertSign usage is allowed (U-KCRTSN). Services: CSNDDSG, CSNDDSV.</td></tr><tr><td>B'xxxx xx1x'</td><td>Certificate Revocation List Sign usage is allowed (U-CRLSN). Services: CSNDDSG, CSNDDSV.</td></tr><tr><td>B'xxxx xxx1'</td><td>Only encipher operations are allowed during key agreement (U-ENCONL).</td></tr></table>Offset 49:<table><tr><th>Bit</th><th>Meaning</th></tr><tr><td>B'1xxx xxxx'</td><td>Only decipher operations are allowed during key agreement (U-DECONL).</td></tr></table>	Bit	Meaning	B'1xxx xxxx'	Digital Signature usage is allowed (U-DIGSIG). Services: CSNDDSG, CSNDDSV, CSNDT34B, CSNDT34D.	B'x1xx xxxx'	Non-Repudiation usage is allowed (U-NONRPD). Services: CSNDDSG, CSNDDSV.	B'xx1x xxxx'	Key Encipherment usage is allowed (U-KEYENC). Services: CSNDSYG, CSNDSYX, CSNDSYI, CSNDSYI2, CSNDT34R, CSNDPKE, CSNDPKD.	B'xxx1 xxxx'	Data Encipherment usage is allowed (U-DATENC). Services: CSNDPKE, CSNDPKD.	B'xxxx 1xxx'	Key agreement usage is allowed (U-KEYAGR).	B'xxxx x1xx'	keyCertSign usage is allowed (U-KCRTSN). Services: CSNDDSG, CSNDDSV.	B'xxxx xx1x'	Certificate Revocation List Sign usage is allowed (U-CRLSN). Services: CSNDDSG, CSNDDSV.	B'xxxx xxx1'	Only encipher operations are allowed during key agreement (U-ENCONL).	Bit	Meaning	B'1xxx xxxx'	Only decipher operations are allowed during key agreement (U-DECONL).
Bit	Meaning																							
B'1xxx xxxx'	Digital Signature usage is allowed (U-DIGSIG). Services: CSNDDSG, CSNDDSV, CSNDT34B, CSNDT34D.																							
B'x1xx xxxx'	Non-Repudiation usage is allowed (U-NONRPD). Services: CSNDDSG, CSNDDSV.																							
B'xx1x xxxx'	Key Encipherment usage is allowed (U-KEYENC). Services: CSNDSYG, CSNDSYX, CSNDSYI, CSNDSYI2, CSNDT34R, CSNDPKE, CSNDPKD.																							
B'xxx1 xxxx'	Data Encipherment usage is allowed (U-DATENC). Services: CSNDPKE, CSNDPKD.																							
B'xxxx 1xxx'	Key agreement usage is allowed (U-KEYAGR).																							
B'xxxx x1xx'	keyCertSign usage is allowed (U-KCRTSN). Services: CSNDDSG, CSNDDSV.																							
B'xxxx xx1x'	Certificate Revocation List Sign usage is allowed (U-CRLSN). Services: CSNDDSG, CSNDDSV.																							
B'xxxx xxx1'	Only encipher operations are allowed during key agreement (U-ENCONL).																							
Bit	Meaning																							
B'1xxx xxxx'	Only decipher operations are allowed during key agreement (U-DECONL).																							

Table 93. RSA private key, 8192-bit Modulus-Exponent format with AES encrypted OPK section (X'30') internal form (continued)

Offset (bytes)	Length (bytes)	Description
050	001	<p>When associated data section version is X'02': Key-usage and translation control flag:</p> <p>Key-usage:</p> <p>B'11xx xxxx' Only key unwrapping (KM-ONLY).</p> <p>B'10xx xxxx' Both signature generation and key unwrapping (KEY-MGMT).</p> <p>B'01xx xxxx' Undefined.</p> <p>B'00xx xxxx' Only signature generation (SIG-ONLY).</p> <p>All other values are undefined.</p> <p>Translation control bit:</p> <p>B'xxxx xx1x' Private key translation is allowed (XLATE-OK).</p> <p>B'xxxx xx0x' Private key translation is not allowed (NO-XLATE).</p> <p>All other bits are reserved and must be zero.</p> <p>When associated data section version is X'04': Reserved, binary zero.</p>
051	001	<p>Format restriction byte for digital-signature hash formatting method.</p> <p>Value:</p> <p>B'0000 0000' No format restriction.</p> <p>B'0000 0001' ISO-9796 only.</p> <p>B'0000 0010' PKCS-1.0 only.</p> <p>B'0000 0011' PKCS-1.1 only.</p> <p>B'0000 0100' PKCS-PSS only.</p> <p>B'0000 0101' X9.31 only.</p> <p>B'0000 0110' ZERO-PAD only.</p> <p>All other values are reserved and undefined.</p>
052	002	Length of modulus: nnn bytes
054	002	Length of private exponent: ddd bytes
		End of Associated Data
056	048	<p>16 byte confounder + 32-byte Object Protection Key.</p> <p>OPK used as an AES key.</p> <p>encrypted with the ECC master key.</p>

Table 93. RSA private key, 8192-bit Modulus-Exponent format with AES encrypted OPK section (X'30') internal form (continued)

Offset (bytes)	Length (bytes)	Description
104	016	Key verification pattern <ul style="list-style-type: none"> For an encrypted private key, <ul style="list-style-type: none"> When a non-compliant-tagged token (bit 0 at offset 13 is not set), the ECC master-key verification pattern (MKVP). When a compliant-tagged token (bit 0 at offset 13 is set), 5 bytes of the ECC MKVP followed by 3 bytes of internal compliance information. For a skeleton, binary zero.
120	002	Reserved, binary zeros.
122	nnn	Modulus
122+nnn	ppp	Payload starts here and includes: When this section is unencrypted: <ul style="list-style-type: none"> Clear private exponent d. Length ppp bytes : ddd + 0 When this section is encrypted: <ul style="list-style-type: none"> Private exponent d within the AESKW-wrapped payload. Length ppp bytes : ddd + AESKW format overhead

RSA private key, 8192-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') internal form

This RSA private key token is supported on Crypto Express3 and later coprocessors. Associated data version 5 is supported on Crypto Express6 and later coprocessors with the July 2019 or later licensed internal code (LIC).

Table 94. RSA private key, 8192-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') internal form

Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: X'31' RSA private key, CRT format with AES encrypted OPK
001	001	Section version number (X'00').
002	002	Section length: 134 + nnn + xxx
004	002	Length of "Associated Data" section
006	002	Length of payload data: xxx
008	002	Reserved, binary zero.
		Start of Associated Data
010	001	Associated Data Version: X'03' Version 3 X'05' Version 5

Table 94. RSA private key, 8192-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') internal form (continued)

Offset (bytes)	Length (bytes)	Description
011	001	Key format and security flag: X'08' Unencrypted RSA private-key subsection identifier
012	001	Key source flag: X'21' Imported from cleartext X'22' Imported from ciphertext X'23' Generated using regeneration data X'24' Randomly generated
013	001	When associated data section version is X'03': Reserved, binary zero. When associated data section version is X'05': Compliance and export control byte. Bit Meaning B'1xxx xxxx' Compliant-tagged key. B'0xxx xxxx' Non-compliant-tagged key. B'xxxx xx1x' Private key translation is allowed (XLATE-OK). B'xxxx xx0x' Private key translation is not allowed (NO-XLATE). All other bits are reserved and must be zero.
014	001	Hash type: X'00' Clear key X'01' SHA-256
015	032	When associated data section version is X'03': SHA-256 hash of all optional sections that follow the public key section, if any. Otherwise, 32 bytes of binary zero. When associated data section version is X'05': Hash value of: 1. The public key section (section identifier X'04') and 2. All optional sections that follow the public key section, if any. If there are no optional sections, the hash covers only the public keys section.
047	001	Reserved, binary zero.

Table 94. RSA private key, 8192-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') internal form (continued)

Offset (bytes)	Length (bytes)	Description																						
048	002	<p>When associated data section version is X'03': Reserved, binary zero.</p> <p>When associated data section version is X'05':</p> <p>Usage bytes:</p> <ul style="list-style-type: none">Offset 48:<table><tr><th>Bit</th><th>Meaning</th></tr><tr><td>B'1xxx xxxx'</td><td>Digital Signature usage is allowed (U-DIGSIG). Services: CSNDDSG, CSNDDSV, CSNDT34B, CSNDT34D.</td></tr><tr><td>B'x1xx xxxx'</td><td>Non-Repudiation usage is allowed (U-NONRPD). Services: CSNDDSG, CSNDDSV.</td></tr><tr><td>B'xx1x xxxx'</td><td>Key Encipherment usage is allowed (U-KEYENC). Services: CSNDSYG, CSNDSYX, CSNDSYI, CSNDSYI2, CSNDT34R, CSNDPKE, CSNDPKD.</td></tr><tr><td>B'xxx1 xxxx'</td><td>Data Encipherment usage is allowed (U-DATENC). Services: CSNDPKE, CSNDPKD.</td></tr><tr><td>B'xxxx 1xxx'</td><td>Key agreement usage is allowed (U-KEYAGR).</td></tr><tr><td>B'xxxx x1xx'</td><td>keyCertSign usage is allowed (U-KCRTSN). Services: CSNDDSG, CSNDDSV.</td></tr><tr><td>B'xxxx xx1x'</td><td>Certificate Revocation List Sign usage is allowed (U-CRLSN). Services: CSNDDSG, CSNDDSV.</td></tr><tr><td>B'xxxx xxx1'</td><td>Only encipher operations are allowed during key agreement (U-ENCONL).</td></tr></table>Offset 49:<table><tr><th>Bit</th><th>Meaning</th></tr><tr><td>B'1xxx xxxx'</td><td>Only decipher operations are allowed during key agreement (U-DECONL).</td></tr></table>	Bit	Meaning	B'1xxx xxxx'	Digital Signature usage is allowed (U-DIGSIG). Services: CSNDDSG, CSNDDSV, CSNDT34B, CSNDT34D.	B'x1xx xxxx'	Non-Repudiation usage is allowed (U-NONRPD). Services: CSNDDSG, CSNDDSV.	B'xx1x xxxx'	Key Encipherment usage is allowed (U-KEYENC). Services: CSNDSYG, CSNDSYX, CSNDSYI, CSNDSYI2, CSNDT34R, CSNDPKE, CSNDPKD.	B'xxx1 xxxx'	Data Encipherment usage is allowed (U-DATENC). Services: CSNDPKE, CSNDPKD.	B'xxxx 1xxx'	Key agreement usage is allowed (U-KEYAGR).	B'xxxx x1xx'	keyCertSign usage is allowed (U-KCRTSN). Services: CSNDDSG, CSNDDSV.	B'xxxx xx1x'	Certificate Revocation List Sign usage is allowed (U-CRLSN). Services: CSNDDSG, CSNDDSV.	B'xxxx xxx1'	Only encipher operations are allowed during key agreement (U-ENCONL).	Bit	Meaning	B'1xxx xxxx'	Only decipher operations are allowed during key agreement (U-DECONL).
Bit	Meaning																							
B'1xxx xxxx'	Digital Signature usage is allowed (U-DIGSIG). Services: CSNDDSG, CSNDDSV, CSNDT34B, CSNDT34D.																							
B'x1xx xxxx'	Non-Repudiation usage is allowed (U-NONRPD). Services: CSNDDSG, CSNDDSV.																							
B'xx1x xxxx'	Key Encipherment usage is allowed (U-KEYENC). Services: CSNDSYG, CSNDSYX, CSNDSYI, CSNDSYI2, CSNDT34R, CSNDPKE, CSNDPKD.																							
B'xxx1 xxxx'	Data Encipherment usage is allowed (U-DATENC). Services: CSNDPKE, CSNDPKD.																							
B'xxxx 1xxx'	Key agreement usage is allowed (U-KEYAGR).																							
B'xxxx x1xx'	keyCertSign usage is allowed (U-KCRTSN). Services: CSNDDSG, CSNDDSV.																							
B'xxxx xx1x'	Certificate Revocation List Sign usage is allowed (U-CRLSN). Services: CSNDDSG, CSNDDSV.																							
B'xxxx xxx1'	Only encipher operations are allowed during key agreement (U-ENCONL).																							
Bit	Meaning																							
B'1xxx xxxx'	Only decipher operations are allowed during key agreement (U-DECONL).																							

Table 94. RSA private key, 8192-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') internal form (continued)

Offset (bytes)	Length (bytes)	Description
050	001	<p>When associated data section version is X'03': Key-usage and translation control flag.</p> <p>Key-usage flag:</p> <p>B'11xx xxxx' Only key unwrapping (KM-ONLY).</p> <p>B'10xx xxxx' Both signature generation and key unwrapping (KEY-MGMT).</p> <p>B'01xx xxxx' Undefined.</p> <p>B'00xx xxxx' Only signature generation (SIG-ONLY).</p> <p>All other values are undefined.</p> <p>Translation control bit:</p> <p>B'xxxx xx1x' Private key translation is allowed (XLATE-OK).</p> <p>B'xxxx xx0x' Private key translation is not allowed (NO-XLATE).</p> <p>All other values are reserved and undefined.</p> <p>When associated data section version is X'05': Reserved, binary zero.</p>
051	001	<p>Format restriction byte for digital-signature hash formatting method.</p> <p>Value:</p> <p>B'0000 0000' No format restriction.</p> <p>B'0000 0001' ISO-9796 only.</p> <p>B'0000 0010' PKCS-1.0 only.</p> <p>B'0000 0011' PKCS-1.1 only.</p> <p>B'0000 0100' PKCS-PSS only.</p> <p>B'0000 0101' X9.31 only.</p> <p>B'0000 0110' ZERO-PAD only.</p> <p>All other values are reserved and undefined.</p>
052	002	Length of the prime number, p, in bytes: ppp.
054	002	Length of the prime number, q, in bytes: qqq
056	002	Length of dp : rrr.
058	002	Length of dq : sss.
060	002	Length of U: uu.
062	002	Length of modulus, nnn.
064	002	Reserved, binary zero.
066	002	Reserved, binary zero.

Table 94. RSA private key, 8192-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') internal form (continued)

Offset (bytes)	Length (bytes)	Description
		End of Associated Data
068	048	16 byte confounder + 32-byte Object Protection Key. OPK used as an AES key. Encrypted with the ECC-MK.
116	016	Key verification pattern <ul style="list-style-type: none"> For an encrypted private key, <ul style="list-style-type: none"> When a non-compliant-tagged token (bit 0 at offset 13 is not set), the ECC master-key verification pattern (MKVP). When a compliant-tagged token (bit 0 at offset 13 is set), 5 bytes of the ECC MKVP followed by 3 bytes of internal compliance information. For a skeleton, binary zero.
132	002	Reserved, binary zero.
134	nnn	Modulus, n, $n=pq$, where p and q are prime.
134+nnn	xxx	Payload starts here and includes: When this section is unencrypted: <ul style="list-style-type: none"> Clear prime number p Clear prime number q Clear dp Clear dq Clear U Length xxx bytes: ppp + qqg + rrr + sss +uuu + 0 When this section is encrypted: <ul style="list-style-type: none"> prime number p prime number q dp dq U within the AESKW-wrapped payload. Length xxx bytes : ppp + qqg + rrr + sss +uuu + AESKW format overhead

RSA private key token, 4096-bit Chinese Remainder Theorem internal form

This RSA private key token (up to 2048-bit modulus) is supported on all cryptographic coprocessors. The 4096-bit modulus private key token is supported on IBM z9 EC, IBM z9 BC, IBM z10 EC, IBM z10 BC, or IBM zEnterprise 196 with the Nov. 2007 or later version of the licensed internal code installed on the CCA Crypto Express coprocessor.

Table 95. RSA Private Internal Key Token, 4096-bit Chinese Remainder Theorem Internal Format

Offset (Dec)	Number of Bytes	Description
000	001	X'08', section identifier, RSA private key, CRT format (RSA-CRT)
001	001	X'00', version.

Table 95. RSA Private Internal Key Token, 4096-bit Chinese Remainder Theorem Internal Format (continued)

Offset (Dec)	Number of Bytes	Description
002	002	Length of the RSA private-key section, 132 + ppp + qqg + rrr + sss + uuu + ttt + iii + xxx + nnn.
004	020	SHA-1 hash value of the private-key subsection cleartext, offset 28 to the end of the modulus.
024	004	Reserved; set to binary zero.
028	001	Key format and security: X'08' Encrypted RSA private-key subsection identifier, Chinese Remainder form.
029	001	Key derivation method: X'21' External private key was specified in the clear. X'22' External private key was encrypted. X'23' Private key was generated using regeneration data. X'24' Private key was randomly generated.
030	020	SHA-1 hash of the optional key-name section and any following sections. If there are no optional sections, then 20 bytes of X'00'.
050	004	Key use flag bits: Bit Meaning When Set On 0 Key management usage permitted. 1 Signature usage not permitted. All other bits reserved, set to binary zero.
054	002	Length of prime number, p, in bytes: ppp.
056	002	Length of prime number, q, in bytes: qqg.
058	002	Length of d _p , in bytes: rrr.
060	002	Length of d _q , in bytes: sss.
062	002	Length of U, in bytes: uuu.
064	002	Length of modulus, n, in bytes: nnn.
066	002	Length of the random number r, in bytes: ttt.
068	002	Length of the random number r ⁻¹ , in bytes: iii.
070	002	Length of padding field, in bytes: xxx.
072	004	Reserved, set to binary zero.
076	016	RSA master key verification pattern.

Table 95. RSA Private Internal Key Token, 4096-bit Chinese Remainder Theorem Internal Format (continued)		
Offset (Dec)	Number of Bytes	Description
092	032	Object Protection Key (OPK) encrypted under the Asymmetric-Keys Master Key using the TDES (CBC outer chaining) algorithm.
124	Start of the encrypted secure subsection, encrypted under the OPK using TDES (CBC outer chaining).	
124	008	Random number, confounder.
132	ppp	Prime number, p.
132 + ppp	qqq	Prime number, q
132 + ppp + qqq	rrr	$d_p = d \bmod (p - 1)$
132 + ppp + qqq + rrr	sss	$d_q = d \bmod (q - 1)$
132 + ppp + qqq + rrr + sss	uuu	$U = q^{-1} \bmod (p)$.
132 + ppp + qqq + rrr + sss + uuu	ttt	Random number r (used in blinding).
132 + ppp + qqq + rrr + sss + uuu + ttt	iii	Random number r^{-1} (used in blinding).
132 + ppp + qqq + rrr + sss + uuu + ttt + iii	xxx	X'00' padding of length xxx bytes such that the length from the start of the confounder at offset 124 to the end of the padding field is a multiple of eight bytes.
	End of the encrypted secure subsection; all of the fields starting with the confounder field and ending with the variable length pad field are encrypted under the OPK using TDES (CBC outer chaining) for key confidentiality.	
132 + ppp + qqq + rrr + sss + uuu + ttt + iii + xxx	nnn	Modulus, n. $n = pq$ where p and q are prime and $1 < n < 2^{4096}$.

ECC key token format

The following table presents the format of the ECC Key Token.

Table 96. ECC Key Token Format		
Offset (Dec)	Number of bytes	Description
Token header		
000	001	Token identifier. X'00' Null token X'1E' External token X'1F' Internal token; the private key is protected by the master key

Table 96. ECC Key Token Format (continued)		
Offset (Dec)	Number of bytes	Description
001	001	Version, X'00'.
002	002	Length of the key token structure excluding the internal information section.
004	004	Ignored; should be zero.
ECC Token Private section		
000	001	X'20', section identifier, ECC private key
001	001	X'00', version.
002	002	Section length.
004	001	<p>Wrapping Method: This value indicates the wrapping method used to protect the data in the encrypted section. It is not the method used to protect the Object Protection Key (OPK).</p> <p>X'00' Clear – section is unencrypted.</p> <p>X'01' AESKW</p> <p>X'02' CBC Wrap - Other</p>
005	001	<p>Hash used for Wrapping</p> <p>X'01' SHA224</p> <p>X'02' SHA256</p> <p>X'04' Reserved.</p> <p>X'08 ' Reserved</p>
006	002	Reserved Binary Zero

Table 96. ECC Key Token Format (continued)

Offset (Dec)	Number of bytes	Description
008	001	<p>Key-usage and translation control flag:</p> <p>Management of symmetric keys and generation of digital signatures:</p> <p>B'11xx xxxx' Only key establishment (KM-ONLY).</p> <p>B'10xx xxxx' Both signature generation and key establishment (KEY-MGMT).</p> <p>B'01xx xxxx' Undefined.</p> <p>B'00xx xxxx' Only signature generation (SIG-ONLY).</p> <p>AESKW export control:</p> <p>B'xxxx x1xx' Private key export under AES key is allowed (AES1ECOK).</p> <p>B'xxxx x0xx' Private key export under AES key is not allowed (NOAES1EC).</p> <p>Translation control:</p> <p>B'xxxx xx1x' Private key translation is allowed (XLATE-OK).</p> <p>B'xxxx xx0x' Private key translation is not allowed (NO-XLATE).</p> <p>Key management:</p> <p>B'xxxx xxx1' Private key CPACF-export is allowed (XPRTCPAC).</p> <p>B'xxxx xxx0' Private key CPACF-export is not allowed (NOEXCPAC).</p> <p>All other bits are reserved and must be zero.</p>
009	001	<p>Curve type:</p> <p>X'00' Prime curve</p> <p>X'01' Brainpool curve</p> <p>X'02' Edwards curve.</p> <p>X'03' Koblitz curve.</p>

Table 96. ECC Key Token Format (continued)

Offset (Dec)	Number of bytes	Description
010	001	Key Format and Security Flag. External Token: X'40' Unencrypted ECC private key identifier X'42' Encrypted ECC private key identifier Internal Token: X'08' Encrypted ECC private key identifier
011	001	Reserved Binary Zero
012	002	Length of p in bits X'00C0' Prime P-192 X'00E0' Prime P-224 X'0100' Prime P-256 X'0180' Prime P-384 X'0209' Prime P-521 X'00A0' Brainpool p-160 X'00C0' Brainpool P-192 X'00E0' Brainpool P-224 X'0100' Brainpool P-256 X'0140' Brainpool P-320 X'0180' Brainpool P-384 X'0200' Brainpool P-512) X'00FF' Edwards 25519. X'01C0' Edwards 448. X'0100' Koblitz P-256.
014	002	IBM Associated Data length. The length of this field must be greater than or equal to 16

Table 96. ECC Key Token Format (continued)		
Offset (Dec)	Number of bytes	Description
016	008	External Token: <ul style="list-style-type: none"> • Unencrypted – Reserved Binary 0x'00' • Encrypted – KVP of the AESKEK Internal Token: MKVP of the ECC-MK
024	048	External Token: reserved binary zeros. Internal Token: Object Protection Key (OPK), ICV (Integrity Check value), 8 byte confounder and a 256-bit AES key used with the AESKW algorithm to encrypt the ECC private key. The OPK is encrypted by the AES master key using AESKW as well. Example format for OPK data passed to AESKW: <ul style="list-style-type: none"> • 8 bytes = A6A6A6A6A6A60000 • 40 bytes = Confounder(8)/Key(32)
072	002	Associated data length, aa
074	002	Length of formatted section in bytes, bb
076	aa	Associated data See “Associated data format for ECC token” on page 364.
076 + aa	Start of formatted section	If this section is in the clear it contains private key d. If it is encrypted it contains the AESKW wrapped payload.
76 + aa	bb	Formatted section which includes Private key d See “AESKW wrapped payload format for ECC private key token” on page 365.
76 + aa + bb	End of formatted section	
ECC Token Public Section		
000	001	X'21', section identifier
001	001	X'00', version.
002	002	Section length
004	004	Reserved field, binary zero
008	001	Curve type X'00' Prime curve X'01' Brainpool curve X'02' Edwards curve. X'03' Koblitz curve.
009	001	Reserved field, binary zero

Table 96. ECC Key Token Format (continued)		
Offset (Dec)	Number of bytes	Description
010	002	<p>Length of p in bits:</p> <p>X'00C0' Prime P-192</p> <p>X'00E0' Prime P-224</p> <p>X'0100' Prime P-256</p> <p>X'0180' Prime P-384</p> <p>X'0209' Prime P-521</p> <p>X'00A0' Brainpool P-160</p> <p>X'00C0' Brainpool P-192</p> <p>X'00E0' Brainpool P-224</p> <p>X'0100' Brainpool P-256</p> <p>X'0140' Brainpool P-320</p> <p>X'0180' Brainpool P-384</p> <p>X'0200' Brainpool P-512</p> <p>X'00FF' Edwards 25519.</p> <p>X'01C0' Edwards 448.</p> <p>X'0100' Koblitz P-256.</p>
012	002	<p>This field is the length of the public key q value in bytes, the maximum value could be up to 133 bytes, cc.</p> <p>For Prime and Brainpool curves, the value includes the key material length plus a one byte flag to indicate if the key material is compressed or uncompressed. For Edwards curves, the value is in little endian format.</p>
014	cc	Public Key , q field

Associated data format for ECC token

Table 97 on page 365 defines the associated data as it is stored in the ECC token in the clear. Associated data is data whose integrity but not confidentiality is protected by a key wrap mechanism.

<i>Table 97. Associated Data Format for ECC Private Key Token</i>		
Offset (Dec)	Number of Bytes	Description
000	001	Associated Data Version. 0 for ECC
001	001	Length of Key Label, kl
002	002	IBM Associated Data length, 16 + kl + xxx
004	002	IBM Extended Associated Data length, xxx
006	001	User Definable Associated Data length, yyy. User definable lengths are from 0 bytes to 100 bytes.
007	001	Curve Type
008	002	Length of p in bits
010	001	Usage flag
011	001	Format and Security flag
012	004	reserved
016	kl	Key Label (optional)
016 + kl	xxx	IBM Extended Associated Data
016 + kl + xxx	yyy	User-definable Associated Data

AESKW wrapped payload format for ECC private key token

This table defines the contents of the AESKW payload: data will be copied into this format, then encrypted with the OPK according to the AESKW specification, and the result will be stored in the encrypted data section.

<i>Table 98. AESKW Wrapped Payload Format for ECC Private Key Token</i>		
Offset (Dec)	Number of Bytes	Description
000	006	ICV ('A6'....)
006	001	Length of padding in bits
007	001	Length of the hash of the associated data in bytes, ii
008	004	Hash options
012	ii	Hash of Associated Data
12+ii	mm	Key data
12+ii+mm	0-7	Padding to a multiple of 8 bytes

QSA key token format

A QSA key token is the concatenation of these sections:

- A token header:
 - An external header (first byte X'1E').
 - An internal header (first byte X'1F').
- An optional private key section (section identifier X'50').
- A public key section (section identifier X'51').

- An optional private key name section (section identifier X'10').

Table 99. QSA Private Key section with OPK (X'50')		
Offset (bytes)	Length (bytes)	Description
000	001	Section Identifier: X'50' QSA Private Key.
001	001	Section version number: X'00' .

Table 99. QSA Private Key section with OPK (X'50') (continued)

Offset (bytes)	Length (bytes)	Description			
002	002	Section length in bytes: 128 + <i>ppp</i> . When 'Key format' (offset 12) is:			
		X'00' (Clear key)			
		Algorithm identifier (offset 9)	Algorithm parameter (offset 10)	<i>ppp</i> (in bytes)	Section length (in bytes)
		X'01'	X'0605'	3824	3952
		X'01'	X'0807'	5104	5232
		X'02'	X'0768'	1216	1344
		X'02'	X'1024'	1600	1728
		X'03'	X'0605'	3968	4096
		X'03'	X'0807'	4832	4960
		X'04'	X'0768'	1216	1344
		X'04'	X'1024'	1600	1728
		X'05', X'07'	X'0404'	2528	2656
		X'05', X'07'	X'0605'	4000	4128
		X'05', X'07'	X'0807'	4864	4992
		X'06'	X'0768'	1216	1344
		X'06'	X'1024'	1600	1728
		X'01' (Encrypted key)			
		Algorithm identifier (offset 9)	Algorithm parameter (offset 10)	<i>ppp</i> (in bytes)	Section length (in bytes)
		X'01'	X'0605'	3872	4000
		X'01'	X'0807'	5152	5280
		X'02'	X'0768'	1264	1392
		X'02'	X'1024'	1648	1776
		X'03'	X'0605'	4016	4144
		X'03'	X'0807'	4880	5008
		X'04'	X'0768'	1264	1392
		X'04'	X'1024'	1648	1776
		X'05', X'07'	X'0404'	2576	2704
		X'05', X'07'	X'0605'	4048	4176
		X'05', X'07'	X'0807'	4912	5040
		X'06'	X'0768'	1264	1392
		X'06'	X'1024'	1648	1776

Table 99. QSA Private Key section with OPK (X'50') (continued)		
Offset (bytes)	Length (bytes)	Description
004	002	Length of associated data section in bytes: 54.
006	002	Reserved, binary zero.
008	001	Associated data section version number: X'01'.
009	001	<p>Algorithm identifier:</p> <p>X'00' No algorithm.</p> <p>X'01' CRYSTALS-Dilithium Round 2.</p> <p>X'02' CRYSTALS-Kyber Round 2.</p> <p>X'03' CRYSTALS-Dilithium Round 3.</p> <p>X'04' CRYSTALS-Kyber Round 3.</p> <p>X'05' Pure ML-DSA.</p> <p>X'06' ML-KEM.</p> <p>X'07' Pre-hash ML-DSA.</p>
010	002	<p>Algorithm parameters:</p> <p>When 'Algorithm identifier' is X'01' or X'03', allowed values are:</p> <p>X'0605' CRYSTALS-Dilithium (6,5).</p> <p>X'0807' CRYSTALS-Dilithium (8,7).</p> <p>When 'Algorithm identifier' is X'02' or X'04', allowed values are:</p> <p>X'0768' CRYSTALS-Kyber (768).</p> <p>X'1024' CRYSTALS-Kyber (1024).</p> <p>When 'Algorithm identifier' is X'05' or X'07', allowed values are:</p> <p>X'0404' ML-DSA (4,4).</p> <p>X'0605' ML-DSA (6,5).</p> <p>X'0807' ML-DSA (8,7).</p> <p>When 'Algorithm identifier' is X'06', allowed values are:</p> <p>X'0768' ML-KEM (768).</p> <p>X'1024' ML-KEM (1024).</p>

Table 99. QSA Private Key section with OPK (X'50') (continued)												
Offset (bytes)	Length (bytes)	Description										
012	001	Key format: External token: X'00' Unencrypted QSA private key subsection identifier. X'01' Encrypted QSA private key subsection identifier. Internal token: X'01' Encrypted QSA private key subsection identifier. All other values are reserved and undefined.										
013	001	Key source flag byte: External token: X'00' No Key. X'23' Private key was generated using regeneration data. X'24' Randomly generated. Internal token: X'00' No Key. X'21' Imported from cleartext. X'22' Imported from ciphertext. X'23' Private key was generated using regeneration data. X'24' Randomly generated. All other values are reserved and undefined.										
014	001	Compliance and export control byte: <table><tr><th>Bit</th><th>Meaning</th></tr><tr><td>B'xxxx x1xx'</td><td>Private key export under AES key is allowed (AES1ECOK).</td></tr><tr><td>B'xxxx x0xx'</td><td>Private key export under AES key is not allowed (NOAES1EC).</td></tr><tr><td>B'xxxx xx1x'</td><td>Private key translation is allowed (XLATE-OK).</td></tr><tr><td>B'xxxx xx0x'</td><td>Private key translation is not allowed (NO-XLATE).</td></tr></table> All other values are reserved and undefined.	Bit	Meaning	B'xxxx x1xx'	Private key export under AES key is allowed (AES1ECOK).	B'xxxx x0xx'	Private key export under AES key is not allowed (NOAES1EC).	B'xxxx xx1x'	Private key translation is allowed (XLATE-OK).	B'xxxx xx0x'	Private key translation is not allowed (NO-XLATE).
Bit	Meaning											
B'xxxx x1xx'	Private key export under AES key is allowed (AES1ECOK).											
B'xxxx x0xx'	Private key export under AES key is not allowed (NOAES1EC).											
B'xxxx xx1x'	Private key translation is allowed (XLATE-OK).											
B'xxxx xx0x'	Private key translation is not allowed (NO-XLATE).											

Table 99. QSA Private Key section with OPK (X'50') (continued)

Offset (bytes)	Length (bytes)	Description
015	001	Hash type: X'00' Clear key. X'02' SHA-256.
016	002	Usage bytes: Byte 16: Bit Meaning B'1xxx xxxx' Digital Signature usage is allowed (U-DIGSIG). Services: CSNDDSG, CSNDDSV. B'xx1x xxxx' Key Encipherment usage is allowed (U-KEYENC). B'xxx1 xxxx' Data Encipherment usage is allowed (U-DATENC). Services: CSNDPKD, CSNDPKE. All other values are reserved and undefined. Byte 17: All values are reserved and undefined.
018	032	SHA-256 hash value of: 1. The public key section (section identifier X'51') and 2. All optional sections that follow the public key section, if any. If there are no optional sections, the hash will cover only the public key section.

Table 99. QSA Private Key section with OPK (X'50') (continued)

Offset (bytes)	Length (bytes)	Description			
050	002	'aaa' = QSA payload component 1 length.			
		Algorithm identifier (offset 9)	Algorithm parameter (offset 10)	aaa item description	aaa value
		X'01'	X'0605'	length of key D, also called 'seed'	32
		X'01'	X'0807'	length of key D, also called 'seed'	32
		X'02'	X'0768'	length of secret polynomial vector	1152
		X'02'	X'1024'	length of secret polynomial vector	1536
		X'03'	X'0605'	length of key D, also called 'seed'	32
		X'03'	X'0807'	length of key D, also called 'seed'	32
		X'04'	X'0768'	length of secret polynomial vector	1152
		X'04'	X'1024'	length of secret polynomial vector	1536
		X'05', X'07'	X'0404'	length of key D, also called 'seed'	32
		X'05', X'07'	X'0605'	length of key D, also called 'seed'	32
		X'05', X'07'	X'0807'	length of key D, also called 'seed'	32
		X'06'	X'0768'	length of secret polynomial vector	1152
		X'06'	X'1024'	length of secret polynomial vector	1536

Table 99. QSA Private Key section with OPK (X'50') (continued)

Offset (bytes)	Length (bytes)	Description			
052	002	'bbb' = QSA payload component 2 length.			
		Algorithm identifier (offset 9)	Algorithm parameter (offset 10)	bbb item description	bbb value
		X'01'	X'0605'	length of tr T : prf output	48
		X'01'	X'0807'	length of tr T : prf output	48
		X'02'	X'0768'	length of public key integrity check	32
		X'02'	X'1024'	length of public key integrity check	32
		X'03'	X'0605'	length of tr T : prf output	32
		X'03'	X'0807'	length of tr T : prf output	32
		X'04'	X'0768'	length of public key integrity check	32
		X'04'	X'1024'	length of public key integrity check	32
		X'05', X'07'	X'0404'	length of tr T : prf output	64
		X'05', X'07'	X'0605'	length of tr T : prf output	64
		X'05', X'07'	X'0807'	length of tr T : prf output	64
		X'06'	X'0768'	length of public key integrity check	32
		X'06'	X'1024'	length of public key integrity check	32

Table 99. QSA Private Key section with OPK (X'50') (continued)

Offset (bytes)	Length (bytes)	Description			
054	002	'ccc' = QSA payload component 3 length.			
		Algorithm identifier (offset 9)	Algorithm parameter (offset 10)	ccc item description	ccc value
		X'01'	X'0605'	length of vector 's1', of L elements	480
		X'01'	X'0807'	length of vector 's1', of L elements	672
		X'02'	X'0768'	length of random data	32
		X'02'	X'1024'	length of random data	32
		X'03'	X'0605'	length of vector 's1', of L elements	640
		X'03'	X'0807'	length of vector 's1', of L elements	672
		X'04'	X'0768'	length of random data	32
		X'04'	X'1024'	length of random data	32
		X'05', X'07'	X'0404'	length of vector 's1', of L elements	384
		X'05', X'07'	X'0605'	length of vector 's1', of L elements	640
		X'05', X'07'	X'0807'	length of vector 's1', of L elements	672
		X'06'	X'0768'	length of random data	32
		X'06'	X'1024'	length of random data	32

Table 99. QSA Private Key section with OPK (X'50') (continued)

Offset (bytes)	Length (bytes)	Description			
056	002	'ddd' = QSA payload component 4 length.			
		Algorithm identifier (offset 9)	Algorithm parameter (offset 10)	ddd item description	ddd value
		X'01'	X'0605'	length of vector 's2', of K elements	576
		X'01'	X'0807'	length of vector 's2', of K elements	768
		X'02'	X'0768'	not used	0
		X'02'	X'1024'	not used	0
		X'03'	X'0605'	length of vector 's2', of K elements	768
		X'03'	X'0807'	length of vector 's2', of K elements	768
		X'04'	X'0768'	not used	0
		X'04'	X'1024'	not used	0
		X'05', X'07'	X'0404'	length of vector 's2', of K elements	384
		X'05', X'07'	X'0605'	length of vector 's2', of K elements	768
		X'05', X'07'	X'0807'	length of vector 's2', of K elements	768
		X'06'	X'0768'	not used	0
		X'06'	X'1024'	not used	0

Table 99. QSA Private Key section with OPK (X'50') (continued)

Offset (bytes)	Length (bytes)	Description			
058	002	'eee' = QSA payload component 5 length.			
		Algorithm identifier (offset 9)	Algorithm parameter (offset 10)	eee item description	eee value
		X'01'	X'0605'	length of 't0', K * high bits (vector)	2688
		X'01'	X'0807'	length of 't0', K * high bits (vector)	3584
		X'02'	X'0768'	not used	0
		X'02'	X'1024'	not used	0
		X'03'	X'0605'	length of 't0', K * high bits (vector)	2496
		X'03'	X'0807'	length of 't0', K * high bits (vector)	3328
		X'04'	X'0768'	not used	0
		X'04'	X'1024'	not used	0
		X'05', X'07'	X'0404'	length of 't0', K * high bits (vector)	1664
		X'05', X'07'	X'0605'	length of 't0', K * high bits (vector)	2496
		X'05', X'07'	X'0807'	length of 't0', K * high bits (vector)	3328
		X'06'	X'0768'	not used	0
		X'06'	X'1024'	not used	0
060	002	Reserved, binary zero.			
062	056	<p>Object Protection key (OPK) data: The OPK consists of a 16-byte confounder and a 256-bit AES key, followed by 8 bytes of AESKW overhead.</p> <p>External token:</p> <ul style="list-style-type: none"> • Encrypted: Encrypted by AES KEK. • Clear: All bytes binary zero. <p>Internal token</p> <ul style="list-style-type: none"> • Encrypted: Encrypted with the ECC master key. 			

Table 99. QSA Private Key section with OPK (X'50') (continued)		
Offset (bytes)	Length (bytes)	Description
118	008	Key verification pattern (KVP): When 'Key format' (offset 12) is: X'00' (Clear key) Binary zero. X'01' (Encrypted key) Bytes 0 – 7: KVP <ul style="list-style-type: none"> • When internal, this is the ECC master key verification pattern (MKVP). • When external, this is the KVP for the KEK.
126	002	Reserved, binary zero
128	ppp	Payload starts here and includes: When this section is encrypted: Private exponent <i>d</i> in the payload wrapped by the OPK using the AESKW algorithm.

Table 100. QSA Public Key section (X'51')		
Offset (bytes)	Length (bytes)	Description
000	001	Section Identifier: X'51' QSA Public Key.
001	001	Section version number: X'00' .
002	002	Section length: 24 + <i>aaa</i> + <i>bbb</i> .
004	001	Key format: X'00' No MAC over public key subsection.
005	001	Algorithm identifier: X'00' No algorithm. X'01' CRYSTALS-Dilithium Round 2. X'02' CRYSTALS-Kyber Round 2. X'03' CRYSTALS-Dilithium Round 3. X'04' CRYSTALS-Kyber Round 3. X'05' Pure ML-DSA X'06' ML-KEM X'07' Pre-hash ML-DSA

Table 100. QSA Public Key section (X'51') (continued)										
Offset (bytes)	Length (bytes)	Description								
006	002	Algorithm parameters: When 'Algorithm identifier' is X'01' or X'03', allowed values are: X'0605' CRYSTALS-Dilithium (6,5). X'0807' CRYSTALS-Dilithium (8,7). When 'Algorithm identifier' is X'02' or X'04', allowed values are: X'0768' CRYSTALS-Kyber (768). X'1024' CRYSTALS-Kyber (1024). When 'Algorithm identifier' is X'05' or X'07', allowed values are: X'0404' ML-DSA (4,4). X'0605' ML-DSA (6,5). X'0807' ML-DSA (8,7). When 'Algorithm identifier' is X'06', allowed values are: X'0768' ML-KEM (768). X'1024' ML-KEM (1024).								
008	002	Usage bytes: Byte 8: <table><tr><th>Bit</th><th>Meaning</th></tr><tr><td>B'1xxx xxxx'</td><td>Digital Signature usage is allowed (U-DIGSIG). Services: CSNDDSV.</td></tr><tr><td>B'xx1x xxxx'</td><td>Key Encipherment usage is allowed (U-KEYENC).</td></tr><tr><td>B'xxx1 xxxx'</td><td>Data Encipherment usage is allowed (U-DATENC). Services: CSNDPKE.</td></tr></table> All other values are reserved and undefined. Byte 9: All values are reserved and undefined.	Bit	Meaning	B'1xxx xxxx'	Digital Signature usage is allowed (U-DIGSIG). Services: CSNDDSV.	B'xx1x xxxx'	Key Encipherment usage is allowed (U-KEYENC).	B'xxx1 xxxx'	Data Encipherment usage is allowed (U-DATENC). Services: CSNDPKE.
Bit	Meaning									
B'1xxx xxxx'	Digital Signature usage is allowed (U-DIGSIG). Services: CSNDDSV.									
B'xx1x xxxx'	Key Encipherment usage is allowed (U-KEYENC).									
B'xxx1 xxxx'	Data Encipherment usage is allowed (U-DATENC). Services: CSNDPKE.									

Table 100. QSA Public Key section (X'51') (continued)

Offset (bytes)	Length (bytes)	Description			
010	002	'aaa' = QSA public component 1 length.			
		Algorithm identifier (offset 9)	Algorithm parameter (offset 10)	aaa item description	aaa value
		X'01'	X'0605'	length of 'rho', also called 'nonce'	32
		X'01'	X'0807'	length of 'rho', also called 'nonce'	32
		X'02'	X'0768'	length of public polynomial vector	1152
		X'02'	X'1024'	length of public polynomial vector	1536
		X'03'	X'0605'	length of 'rho', also called 'nonce'	32
		X'03'	X'0807'	length of 'rho', also called 'nonce'	32
		X'04'	X'0768'	length of public polynomial vector	1152
		X'04'	X'1024'	length of public polynomial vector	1536
		X'05', X'07'	X'0404'	length of 'rho', also called 'nonce'	32
		X'05', X'07'	X'0605'	length of 'rho', also called 'nonce'	32
		X'05', X'07'	X'0807'	length of 'rho', also called 'nonce'	32
		X'06'	X'0768'	length of public polynomial vector	1152
		X'06'	X'1024'	length of public polynomial vector	1536

Table 100. QSA Public Key section (X'51') (continued)

Offset (bytes)	Length (bytes)	Description			
012	002	'bbb' = QSA public component 2 length.			
		Algorithm identifier (offset 9)	Algorithm parameter (offset 10)	bbb item description	bbb value
		X'01'	X'0605'	length of 't1' : K*low bits (vector)	1728
		X'01'	X'0807'	length of 't1' : K*low bits (vector)	2304
		X'02'	X'0768'	length of public seed	32
		X'02'	X'1024'	length of public seed	32
		X'03'	X'0605'	length of 't1' : K*low bits (vector)	1920
		X'03'	X'0807'	length of 't1' : K*low bits (vector)	2560
		X'04'	X'0768'	length of public seed	32
		X'04'	X'1024'	length of public seed	32
		X'05', X'07'	X'0404'	length of 't1' : K*low bits (vector)	1280
		X'05', X'07'	X'0605'	length of 't1' : K*low bits (vector)	1920
		X'05', X'07'	X'0807'	length of 't1' : K*low bits (vector)	2560
		X'06'	X'0768'	length of public seed	32
		X'06'	X'1024'	length of public seed	32
014	010	Reserved, binary zero.			

Table 100. QSA Public Key section (X'51') (continued)		
Offset (bytes)	Length (bytes)	Description
024	aaa	<p>QSA public component 1</p> <p>When 'Algorithm identifier' is X'01', X'03', X'05', or X'07' (ML-DSA, CRYSTALS-Dilithium):</p> <ul style="list-style-type: none"> 'rho', also called 'nonce'. <p>When 'Algorithm identifier' is X'02', X'04', or X'06' (ML-KEM, CRYSTALS-Kyber):</p> <ul style="list-style-type: none"> Public polynomial vector.
024 + aaa	bbb	<p>QSA public component 2</p> <p>When 'Algorithm identifier' is X'01', X'03', X'05', or X'07' (ML-DSA, CRYSTALS-Dilithium):</p> <ul style="list-style-type: none"> 't1': K * low bits (vector). <p>When 'Algorithm identifier' is X'02', X'04', or X'06' (ML-KEM, CRYSTALS-Kyber):</p> <ul style="list-style-type: none"> Public seed.

AESKW external format

AES Key Wrap (AESKW) is a symmetric encryption algorithm designed to encapsulate cryptographic key material. CCA ECC, ML-DSA, ML-KEM, CRYSTALS-Dilithium, and CRYSTALS-Kyber private key tokens can be exported to AESKW external format wrapped with an AES key-encrypting-key with the PKA Key Translate callable service. CCA private key tokens exported to AESKW external format cannot be written to the PKDS.

Table 101 on page 380 outlines the structure of the AESKW external format:

Table 101. AESKW external format structure		
Offset	Size	Field
Start of Associated Data		
	Sub-section: Header	
0	1	<p>Primary Identifier:</p> <p>X'53' ASCII 'S'.</p>
1	1	<p>Version:</p> <p>X'00' AESKW wrapping method.</p>
2	2	Structure length (SL): Length in bytes of the total structure (big endian).
	Sub-section: Key data	

Table 101. AESKW external format structure (continued)		
Offset	Size	Field
4	1	<p>Algorithm type:</p> <p>Value</p> <p>Meaning</p> <p>X'81' ECC key (private).</p> <p>X'82' QSA key : CRYSTALS-Dilithium Round 2 (private).</p> <p>X'83' QSA key : CRYSTALS-Kyber Round 2 (private).</p> <p>X'84' QSA key : CRYSTALS-Dilithium Round 3 (private).</p> <p>X'85' QSA key : CRYSTALS-Kyber Round 3 (private).</p> <p>X'86' QSA key : Pure ML-DSA (private).</p> <p>X'87' QSA key : ML-KEM (private).</p> <p>X'88' QSA key : Pre-hash ML-DSA (private).</p>
5	2	<p>Key Type:</p> <p>Values in this field depend on the input token and the value at offset X'04', Algorithm type.</p> <p>When input token is an ECC private key:</p> <p>Encoding:</p> <ol style="list-style-type: none"> 1. Upper nibble details curve type. 2. Lower three nibbles details, length of private key 'p' in bits. <p>Values</p> <ol style="list-style-type: none"> 1. X'0209' ECC PRIME 521. <p>When input token is a QSA private key:</p> <p>Encoding:</p> <ol style="list-style-type: none"> 1. Values correspond to the algorithm parameter field of the QSA token. <p>Values</p> <ol style="list-style-type: none"> 1. X'0404' ML-DSA (4,4). 2. X'0605' ML-DSA, CRYSTALS-Dilithium (6,5). 3. X'0807' ML-DSA, CRYSTALS-Dilithium (8,7). 4. X'1024' ML-KEM, CRYSTALS-Kyber (1024). 5. X'0768' ML-KEM, CRYSTALS-Kyber (768).

Table 101. AESKW external format structure (continued)

Offset	Size	Field																		
7	1	<p>Kuf count: Key usage fields count 0 - 4. Key usage field information defines restrictions on the use of the key.</p> <p>ECC private key token: Kuf count = 2.</p> <p>QSA ML-DSA, CRYSTALS-Dilithium: kuf count = 2.</p> <p>QSA ML-KEM, CRYSTALS-Kyber: kuf count = 2.</p> <p>Notes:</p> <ul style="list-style-type: none">Each <i>key-usage</i> field is 2 bytes in length. The value in this field indicates how many 2-byte key usage fields follow.There are 8 bytes of Associated Data after this field. Each of these remaining 8 bytes of Associated Data is either X'00' or a Kuf byte, as indicated by this kuf count field. This section is not variable size. Unused bytes have an X'00' value. <p>Examples:</p> <p>kuf_count = 1 2 bytes of key usage information follow, this is 1 kuf field. The remaining 6 bytes of Associated Data are 0x00 bytes.</p> <p>kuf_count = 2 4 bytes of key usage information follow, this is 2 kuf fields. The remaining 4 bytes of Associated Data are 0x00 bytes.</p>																		
8	1	<p>Bit value meanings:</p> <table><thead><tr><th>Bit</th><th>Meaning</th></tr></thead><tbody><tr><td>B'1xxx xxxx'</td><td>digitalSignature.</td></tr><tr><td>B'x1xx xxxx'</td><td>nonrepudiation or contentCommitment.</td></tr><tr><td>B'xx1x xxxx'</td><td>keyEncipherment.</td></tr><tr><td>B'xxx1 xxxx'</td><td>dataEncipherment.</td></tr><tr><td>B'xxxx 1xxx'</td><td>keyAgreement.</td></tr><tr><td>B'xxxx x1xx'</td><td>keyCertSign.</td></tr><tr><td>B'xxxx xx1x'</td><td>cRLSign.</td></tr><tr><td>B'xxxx xxx1'</td><td>encipherOnly (requires keyAgreement).</td></tr></tbody></table>	Bit	Meaning	B'1xxx xxxx'	digitalSignature.	B'x1xx xxxx'	nonrepudiation or contentCommitment.	B'xx1x xxxx'	keyEncipherment.	B'xxx1 xxxx'	dataEncipherment.	B'xxxx 1xxx'	keyAgreement.	B'xxxx x1xx'	keyCertSign.	B'xxxx xx1x'	cRLSign.	B'xxxx xxx1'	encipherOnly (requires keyAgreement).
Bit	Meaning																			
B'1xxx xxxx'	digitalSignature.																			
B'x1xx xxxx'	nonrepudiation or contentCommitment.																			
B'xx1x xxxx'	keyEncipherment.																			
B'xxx1 xxxx'	dataEncipherment.																			
B'xxxx 1xxx'	keyAgreement.																			
B'xxxx x1xx'	keyCertSign.																			
B'xxxx xx1x'	cRLSign.																			
B'xxxx xxx1'	encipherOnly (requires keyAgreement).																			
9	1	<p>Bit value meanings:</p> <table><thead><tr><th>Bit</th><th>Meaning</th></tr></thead><tbody><tr><td>B'1xxx xxxx'</td><td>decipherOnly. Requires keyAgreement bit at offset 8. Cannot be combined with encipherOnly.</td></tr></tbody></table> <p>All other values are reserved and undefined.</p>	Bit	Meaning	B'1xxx xxxx'	decipherOnly. Requires keyAgreement bit at offset 8. Cannot be combined with encipherOnly.														
Bit	Meaning																			
B'1xxx xxxx'	decipherOnly. Requires keyAgreement bit at offset 8. Cannot be combined with encipherOnly.																			

Table 101. AESKW external format structure (continued)		
Offset	Size	Field
10	1	This field is reserved and must be X'00' byte.
11	1	This field is reserved and must be X'00' byte.
12	1	This field is reserved and must be X'00' byte.
13	1	This field is reserved and must be X'00' byte.
14	1	This field is reserved and must be X'00' byte.
15	1	This field is reserved and must be X'00' byte.
End of Associated Data		
Start of AESKW wrapped payload		
16	6	Integrity constant: byte array. Value: X'A6A6A6A6A6A6'
22	1	PbL: Zero padding bit length. This padding is after the key, at the end of the payload.
23	1	ADLen: Associated Data byte length, in hex. Value: X'10'
24	16	Copy of Associated Data. The Associated Data is copied here. After decryption, this must match the clear data shown above in the 'Associated Data' section.

Table 101. AESKW external format structure (continued)

Offset	Size	Field
40	KL	<p>Keydata</p> <p>Format of keydata:</p> <p>When algorithm type is ECC (X'81'): The maximum size key, P521, will have KL=66 bytes.</p> <p>When algorithm type is QSA CRYSTALS-Dilithium Round 2 (X'82'), type (X'0605'):</p> <ul style="list-style-type: none"> • 32 B for component 1: key D. • 48 B for component 2: tr T. • 480 B for component 3: vector 's1'. • 576 B component 4: vector 's2'. • 2688 B component 5: 't0'. <p>When algorithm type is QSA CRYSTALS-Dilithium Round 2 (X'82'), type (X'0807'):</p> <ul style="list-style-type: none"> • 32 B for component 1: key D. • 48 B for component 2: tr T. • 672 B for component 3: vector 's1'. • 768 B component 4: vector 's2'. • 3584 B component 5: 't0'. <p>When algorithm type is QSA CRYSTALS-Kyber Round 2 (X'83'), type (X'0768'):</p> <ul style="list-style-type: none"> • 1152 B for component 1: secret polynomial vector. • 32 B for component 2: public key integrity check. • 32 B for component 3: random data. <p>When algorithm type is QSA CRYSTALS-Kyber Round 2 (X'83'), type (X'1024'):</p> <ul style="list-style-type: none"> • 1536 B for component 1: secret polynomial vector. • 32 B for component 2: public key integrity check. • 32 B for component 3: random data. <p>When algorithm type is QSA CRYSTALS-Dilithium Round 3 (X'84'), type (X'0605'):</p> <ul style="list-style-type: none"> • 32 B for component 1: key D. • 32 B for component 2: tr T. • 640 B for component 3: vector 's1'. • 768 B component 4: vector 's2'. • 2496 B component 5: 't0'. <p>When algorithm type is QSA CRYSTALS-Dilithium Round 3 (X'84'), type (X'0807'):</p> <ul style="list-style-type: none"> • 32 B for component 1: key D. • 32 B for component 2: tr T. • 672 B for component 3: vector 's1'. • 768 B component 4: vector 's2'. • 3328 B component 5: 't0'.

Table 101. AESKW external format structure (continued)

Offset	Size	Field
40 (con't)	KL	<p>Keydata</p> <p>Format of keydata:</p> <p>When algorithm type is QSA CRYSTALS-Kyber Round 3 (X'85'), type (X'0768'):</p> <ul style="list-style-type: none"> • 1152 B for component 1: secret polynomial vector. • 32 B for component 2: public key integrity check. • 32 B for component 3: random data. <p>When algorithm type is QSA CRYSTALS-Kyber Round 3 (X'85'), type (X'1024'):</p> <ul style="list-style-type: none"> • 1536 B for component 1: secret polynomial vector. • 32 B for component 2: public key integrity check. • 32 B for component 3: random data. <p>When algorithm type is QSA ML-DSA (X'86' or X'88'), type (X'0404'):</p> <ul style="list-style-type: none"> • 32 B for component 1: key D. • 64 B for component 2: tr T. • 384 B for component 3: vector 's1'. • 384 B component 4: vector 's2'. • 1664 B component 5: 't0'. <p>When algorithm type is QSA ML-DSA (X'86' or X'88'), type (X'0605'):</p> <ul style="list-style-type: none"> • 32 B for component 1: key D. • 64 B for component 2: tr T. • 640 B for component 3: vector 's1'. • 768 B component 4: vector 's2'. • 2496 B component 5: 't0'. <p>When algorithm type is QSA ML-DSA (X'86' or X'88'), type (X'0807'):</p> <ul style="list-style-type: none"> • 32 B for component 1: key D. • 64 B for component 2: tr T. • 672 B for component 3: vector 's1'. • 768 B component 4: vector 's2'. • 3328 B component 5: 't0'. <p>When algorithm type is QSA ML-KEM (X'87'), type (X'0768'):</p> <ul style="list-style-type: none"> • 1152 B for component 1: secret polynomial vector. • 32 B for component 2: public key integrity check. • 32 B for component 3: random data. <p>When algorithm type is QSA ML-KEM (X'87'), type (X'1024'):</p> <ul style="list-style-type: none"> • 1536 B for component 1: secret polynomial vector. • 32 B for component 2: public key integrity check. • 32 B for component 3: random data.
KL+40	PbL/ 8	Padding data: PbL count of 0b0 bits.
End of AESKW wrapped payload		

Table 101. AESKW external format structure (continued)		
Offset	Size	Field
KL+40+ (PbL/8)		<p>Final size (FS).</p> <p>Algorithm Size</p> <p>ECC: 521-bit P521: KL = 66 bytes; PbL = 48 bits; FS = 112 bytes (This is the max for ECC).</p> <p>QSA CRYSTALS-Dilithium Round 2 (X'82'), type (X'0605'): KL = 3824; PbL = 0; FS = 3864 bytes.</p> <p>QSA CRYSTALS-Dilithium Round 2 (X'82'), type (X'0807'): KL = 5104; PbL = 0; FS = 5144 bytes.</p> <p>QSA CRYSTALS-Kyber Round 2 (X'83'), type (X'0768'): KL = 1216; PbL = 0; FS = 1256 bytes.</p> <p>QSA CRYSTALS-Kyber Round 2 (X'83'), type (X'1024'): KL = 1600; PbL = 0; FS = 1640 bytes.</p> <p>QSA CRYSTALS-Dilithium Round 3 (X'84'), type (X'0605'): KL = 3968; PbL = 0; FS = 4008 bytes.</p> <p>QSA CRYSTALS-Dilithium Round 3 (X'84'), type (X'0807'): KL = 4832; PbL = 0; FS = 4872 bytes.</p> <p>QSA CRYSTALS-Kyber Round 3 (X'85'), type (X'0768'): KL = 1216; PbL = 0; FS = 1256 bytes.</p> <p>QSA CRYSTALS-Kyber Round 3 (X'85'), type (X'1024'): KL = 1600; PbL = 0; FS = 1640 bytes.</p> <p>QSA ML-DSA (X'86' or X'88'), type (X'0404'): KL = 2528; PbL = 0; FS = 2568 bytes.</p> <p>QSA ML-DSA (X'86' or X'88'), type (X'0605'): KL = 4000; PbL = 0; FS = 4040 bytes.</p> <p>QSA ML-DSA (X'86' or X'88'), type (X'0807'): KL = 4864; PbL = 0; FS = 4904 bytes.</p> <p>QSA ML-KEM (X'87'), type (X'0768'): KL = 1216; PbL = 0; FS = 1256 bytes.</p> <p>QSA ML-KEM (X'87'), type (X'1024'): KL = 1600; PbL = 0; FS = 1640 bytes.</p>

Trusted blocks

A trusted block is an extension of CCA PKA key tokens using new section identifiers. They are an integral part of a remote key-loading process.

Trusted blocks contain various items, some of which are optional, and some of which can be present in different forms. Tokens are composed of concatenated sections that, unlike CCA PKA key tokens, occur in no prescribed order.

As with other CCA key-tokens, both internal and external forms are defined:

- An external trusted block contains a randomly generated confounder and a triple-length MAC key enciphered under a DES IMP-PKA transport key. The MAC key is used to calculate an ISO 16609 CBC mode TDES MAC of the trusted block contents. An external trusted block is created by the Trusted Block Create callable service. This service can:
 1. Create an inactive external trusted block.
 2. Change an external trusted block from inactive to active.

- An internal trusted block contains a confounder and triple-length MAC key enciphered under a variant of the PKA master key. The MAC key is used to calculate a TDES MAC of the trusted block contents. A PKA master key verification pattern is also included to enable determination that the proper master key is available to process the key. The Remote Key Export service only operates on trusted blocks that are internal. An internal trusted block must be imported from an external trusted block that is active using the PKA Key Import service.

Note: Trusted blocks do not contain a private key section.

Trusted block sections

A trusted block is a concatenation of a header followed by an unordered set of sections. The data structures of these sections are summarized in the following table:

Table 102. Trusted block sections		
Section	Reference	Usage
Header	Table 103 on page 388	Trusted block token header
X'11'	Table 104 on page 389	Trusted block public key
X'12'	Table 105 on page 390	Trusted block rule
X'13'	Table 112 on page 397	Trusted block name (key label)
X'14'	Table 113 on page 397	Trusted block information
X'15'	Table 117 on page 399	Trusted block application-defined data

Every trusted block starts with a token header. The first byte of the token header determines the key form:

- An external header (first byte X'1E'), created by the Trusted Block Create verb
- An internal header (first byte X'1F'), imported from an active external trusted block by the PKA Key Import verb

Following the token header of a trusted block is an unordered set of sections. A trusted block is formed by concatenating these sections to a trusted block header:

- An optional public-key section (trusted block section identifier X'11')

The trusted block trusted RSA public-key section includes the key itself in addition to a key-usage flag. No multiple sections are allowed.

- An optional rule section (trusted block section identifier X'12')

A trusted block may have zero or more rule sections.

1. A trusted block with no rule sections can be used by the PKA Key Token Change and PKA Key Import callable services. A trusted block with no rule sections can also be used by the Digital Signature Verify verb, provided there is an RSA public-key section that has its key-usage flag bits set to allow digital signature operations.
 2. At least one rule section is required when the Remote Key Export verb is used to:
 - Generate an RKX key-token
 - Export an RKX key-token
 - Export a CCA DES key-token
 - Encrypt the clear generated or exported key using the provided vendor certificate
 3. If a trusted block has multiple rule sections, each rule section must have a unique 8-character Rule ID.
- An optional name (key label) section (trusted block section identifier X'13')

The trusted block name section provides a 64-byte variable to identify the trusted block, just as key labels are used to identify other CCA keys. This name, or label, enables a host access-control system

such as RACF to use the name to verify that the application has authority to use the trusted block. No multiple sections are allowed.

- A required information section (trusted block section identifier X'14')

The trusted block information section contains control and security information related to the trusted block. The information section is required while the others are optional. This section contains the cryptographic information that guarantees its integrity and binds it to the local system. No multiple sections are allowed.

- An optional application-defined data section (trusted block section identifier X'15')

The trusted block application-defined data section can be used to include application-defined data in the trusted block. The purpose of the data in this section is defined by the application. CCA does not examine or use this data in any way. No multiple sections are allowed.

Trusted block integrity

An enciphered confounder and triple-length MAC key contained within the required information section of the trusted block is used to protect the integrity of the trusted block. The randomly generated MAC key is used to calculate an ISO 16609 CBC mode TDES MAC of the trusted block contents. Together, the MAC key and MAC value provide a way to verify that the trusted block originated from an authorized source, and binds it to the local system.

An external trusted block has its MAC key enciphered under an IMP-PKA key-encrypting key. An internal trusted block has its MAC key enciphered under a variant of the PKA master key, and the master key verification pattern is stored in the information section.

Number representation in trusted blocks

- All length fields are in binary.
- All binary fields (exponents, lengths, and so forth) are stored with the high-order byte first; thus the least significant bits are to the right and preceded with zero-bits to the width of a field.
- In variable-length binary fields that have an associated field-length value, leading bytes that would otherwise contain X'00' can be dropped and the field can be shortened to contain only the significant bits.

Format of trusted block sections

At the beginning of every trusted block is a trusted block header. The header contains the following information:

- A token identifier, which specifies whether the token contains an external or internal key-token.
- A token version number to allow for future changes.
- A length in bytes of the trusted block, including the length of the header.

The trusted block header is defined in the following table:

Table 103. Trusted block header		
Offset (bytes)	Length (bytes)	Description
000	001	Token identifier (a flag that indicates token type) X'1E' External trusted block token. X'1F' Internal trusted block token.
001	001	Token version number (X'00').
002	002	Length of the key-token structure in bytes.
004	004	Reserved, binary zero.

Note: See “Number representation in trusted blocks” on page 388.

Following the header, in no particular order, are trusted block sections. There are five different sections that are defined, each identified by a one-byte section identifier (X'11' - X'15'). Two of the five sections have subsections that are defined. A subsection is a tag-length-value (TLV) object, which is identified by a two-byte subsection tag.

Only sections X'12' and X'14' have subsections that are defined; the other sections do not. A section and its subsections, if any, are one contiguous unit of data. The subsections are concatenated to the related section, but are otherwise in no particular order. Section X'12' has five subsections that are defined (X'0001' - X'0005'), and section X'14' has two (X'0001' and X'0002'). Of all the subsections, only subsection X'0001' of section X'14' is required. Section X'14' is also required.

The trusted block sections and subsections are described in detail in the following sections.

Trusted block section X'11'

Trusted block section X'11' contains the trusted RSA public key in addition to a key-usage flag indicating whether the public key is usable in key-management operations, digital signature operations, or both.

Section X'11' is optional. No multiple sections are allowed. It has no subsections that are defined.

This section is defined in the following table:

Table 104. Trusted block trusted RSA public-key section (X'11')		
Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: X'11' Trusted block trusted RSA public key.
001	001	Section version number (X'00').
002	002	Section length (16+xxx+yyy).
004	002	Reserved, must be binary zero.
006	002	RSA public-key exponent field length in bytes, xxx.
008	002	RSA public-key modulus length in bits.
010	002	RSA public-key modulus field length in bytes, yyy.
012	xxx	Public-key exponent, e (this field length is typically 1, 3, or 64 - 512 bytes). e must be odd and $1 \leq e < n$. (e is frequently valued to 3 or $2^{16}+1$ (=65537), otherwise e is of the same order of magnitude as the modulus). Note: Although the current product implementation does not generate such a public key, you can import an RSA public key having an exponent valued to two (2). Such a public key (a Rabin key) can correctly validate an ISO 9796-1 digital signature.
012+xxx	yyy	RSA public-key modulus, n . $n=pq$, where p and q are prime and $2^{512} \leq n < 2^{4096}$. The field length is 64 - 512 bytes.
012+xxx+yyy	004	Flags: X'00000000' Trusted block public key can be used in digital signature operations only. X'80000000' Trusted block public key can be used in both digital signature and key management operations. X'C0000000' Trusted block public key can be used in key management operations only.

Note: See “Number representation in trusted blocks” on page 388.

Trusted block section X'12'

Trusted block section X'12' contains information that defines a rule. A trusted block can have zero or more rule sections.

1. A trusted block with no rule sections can be used by the PKA Key Token Change and PKA Key Import callable services. A trusted block with no rule sections can be used by the Digital Signature Verify verb, provided there is an RSA public-key section that has its key-usage flag set to allow digital signature operations.
2. At least one rule section is required when the Remote Key Export verb is used to:
 - Generate an RKX key-token.
 - Export an RKX key-token.
 - Export a CCA DES key-token.
 - Generate or export a key encrypted by a public key. The public key is contained in a vendor certificate (section X'11'), and is the root certification key for the ATM vendor. It is used to verify the digital signature on public-key certificates for specific individual ATMs.
3. If a trusted block has multiple rule sections, each rule section must have a unique 8-character Rule ID.

Section X'12' is the only section that is allowed to have multiple sections. Section X'12' is optional. Multiple sections are allowed.

Note: The overall length of the trusted block can not exceed its maximum size of 3500 bytes.

Five subsections (TLV objects) are defined.

This section is defined in the following table:

Table 105. Trusted block rule section (X'12')		
Offset (bytes)	Length (bytes)	Description
Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: X'12' Trusted block rule.
001	001	Section version number (X'00').
002	002	Section length in bytes (20+yyy).
004	008	Rule ID (in ASCII). An 8-byte character string that uniquely identifies the rule within the trusted block. Valid ASCII characters are: A...Z, a...z, 0...9, - (hyphen), and _ (underscore), left-justified and padded on the right with space characters.
012	004	Flags (undefined flag bits are reserved and must be zero). X'00000000' Generate new key. X'00000001' Export existing key.
016	001	Generated key length. Length in bytes of key to be generated when flags value (offset 012) is set to generate a new key; otherwise, ignore this value. Valid values are 8, 16, or 24; return an error if not valid.

Table 105. Trusted block rule section (X'12') (continued)		
Offset (bytes)	Length (bytes)	Description
017	001	<p>Key-check algorithm identifier (all others are reserved and must not be used):</p> <p>Value Meaning</p> <p>X'00' Do not compute key-check value. In a call to CSNDRKX or CSNFRKX, set the key_check_length variable to zero.</p> <p>X'01' Encrypt an 8-byte block of binary zeros with the key. In a call to CSNDRKX or CSNFRKX, set the key_check_length variable to 8.</p> <p>X'02' Compute the MDC-2 hash of the key. In a call to CSNDRKX or CSNFRKX, set the key_check_length variable to 16.</p>
018	001	<p>Symmetric encrypted output key format flag (all other values are reserved and must not be used).</p> <p>Return the indicated symmetric key-token by using the <i>sym_encrypted_key_identifier</i> parameter.</p> <p>Value Meaning</p> <p>X'00' Return an RKX key-token encrypted under a variant of the MAC key.</p> <p>Note: This is the only key format that is permitted when the flags value (offset 012) is set to generate a new key.</p> <p>X'01' Return a CCA DES key-token encrypted under a transport key.</p> <p>Note: This is the only key format that is permitted when the flags value (offset 012) is set to export an existing key.</p>
019	001	<p>Asymmetric encrypted output key format flag (all other values are reserved and must not be used).</p> <p>Return the indicated asymmetric key-token in the <i>asym_encrypted_key</i> variable.</p> <p>Value Meaning</p> <p>X'00' Do not return an asymmetric key. Set the <i>asym_encrypted_key_length</i> variable to zero.</p> <p>X'01' Output in PKCS1.2 format.</p> <p>X'02' Output in RSAOAEP format.</p>
020	yyy	Rule section subsections (tag-length-value objects). A series of 0 - 5 objects in TLV format.

Note: See “Number representation in trusted blocks” on page 388.

Section X'12' has five rule subsections (tag-length-value objects) defined. These subsections are summarized in the following table:

Table 106. Summary of trusted block rule subsection			
Rule subsection tag	TLV object	Optional or required	Comments
X'0001'	Transport key variant	Optional	Contains variant to be exclusive-ORed into the cleartext transport key.
X'0002'	Transport key rule reference	Optional; required to use an RKX key-token as a transport key	Contains the rule ID for the rule that must have been used to create the transport key.

Table 106. Summary of trusted block rule subsection (continued)

Rule subsection tag	TLV object	Optional or required	Comments
X'0003'	Common export key parameters	Optional for key generation; required for key export of an existing key	Contains the export key and source key minimum and maximum lengths, an output key variant length and variant, a CV length, and a CV to be exclusive-ORed with the cleartext transport key to control usage of the key.
X'0004'	Source key reference	Optional; required if the source key is an RKX key-token	Contains the rule ID for the rule used to create the source key. Note: Include all rules that will ever be needed when a trusted block is created. A rule cannot be added to a trusted block after it has been created.
X'0005'	Export key CCA token parameters	Optional; used for export of CCA DES key tokens only	Contains mask length, mask, and CV template to limit the usage of the exported key. Also contains the template length and template that defines which source key labels are allowed. The key type of a source key input parameter can be "filtered" by using the export key CV limit mask (offset 005) and limit template (offset 005+yyy) in this subsection.

Note: See “Number representation in trusted blocks” on page 388.

Trusted block section X'12' subsection X'0001'

Subsection X'0001' of the trusted block rule section (X'12') is the transport key variant TLV object. This subsection is optional. It contains a variant to be exclusive-ORed into the cleartext transport key.

This subsection is defined in the following table:

Table 107. Transport key variant subsection (X'0001' of trusted block rule section (X'12'))		
Offset (bytes)	Length (bytes)	Description
000	002	Subsection tag: X'0001' Transport key variant TLV object.
002	002	Subsection length in bytes (8+ <i>nnn</i>).
004	001	Subsection version number (X'00').
005	002	Reserved, must be binary zero.
007	001	Length of variant field in bytes (<i>nnn</i>). This length must be greater than or equal to the length of the transport key that is identified by the <i>transport_key_identifier</i> parameter. If the variant is longer than the key, truncate it on the right to the length of the key before use.
008	<i>nnn</i>	Transport key variant. Exclusive-OR this variant into the cleartext transport key, provided: (1) the length of the variant field value (offset 007) is not zero, and (2) the symmetric encrypted output key format flag (offset 018 in section X'12') is X'01'. Note: A transport key is not used when the symmetric encrypted output key is in RKX key-token format.

Note: See “Number representation in trusted blocks” on page 388.

Trusted block section X'12' subsection X'0002'

Subsection X'0002' of the trusted block rule section (X'12') is the transport key rule reference TLV object. This subsection is optional. It contains the rule ID for the rule that must have been used to create the transport key. This subsection must be present to use an RKX key-token as a transport key.

This subsection is defined in the following table:

Table 108. Transport key rule reference subsection (X'0002') of trusted block rule section (X'12')		
Offset (bytes)	Length (bytes)	Description
000	002	Subsection tag: X'0002' Transport key rule reference TLV object.
002	002	Subsection length in bytes (14).
004	001	Subsection version number (X'00').
005	001	Reserved, must be binary zero.
006	008	Rule ID. Contains the rule identifier for the rule that must have been used to create the RKX key-token used as the transport key. The Rule ID is an 8-byte string of ASCII characters, left-justified and padded on the right with space characters. Acceptable characters are A...Z, a...z, 0...9, - (X'2D'), and _ (X'5F'). All other characters are reserved for future use.

Trusted block section (X'12') subsection X'0003'

Subsection X'0003' of the trusted block rule section (X'12') is the common export key parameters TLV object. This subsection is optional, but is required for the key export of an existing source key (identified by the *source_key_identifier* parameter) in either RKX key-token format or CCA DES key-token format. For new key generation, this subsection applies the output key variant to the cleartext generated key, if such an option is wanted. It contains the input source key and output export key minimum and maximum lengths, an output key variant length and variant, a CV length, and a CV to be exclusive-ORed with the cleartext transport key.

This subsection is defined in the following table:

Table 109. Common export key parameters subsection (X'0003') of trusted block rule section (X'12')		
Offset (bytes)	Length (bytes)	Description
000	002	Subsection tag: X'0003' Common export key parameters TLV object.
002	002	Subsection length in bytes (12+xxx+yyy).
004	001	Subsection version number (X'00').
005	002	Reserved, must be binary zero.
007	001	Flags (must be set to binary zero).
008	001	Export key minimum length in bytes. Length must be 8, 16, or 24. Also applies to the source key.
009	001	Export key maximum length in bytes (yyy). Length must be 8, 16, or 24. Also applies to the source key.
010	001	Output key variant length in bytes (xxx). Valid values are 0 or 8 - 255. If greater than 0, the length must be at least as long as the longest key ever to be exported that uses this rule. If the variant is longer than the key, truncate it on the right to the length of the key before use. Note: The output key variant (offset 011) is not used if this length is zero.
011	xxx	Output key variant. The variant can be any value. Exclusive-OR this variant into the cleartext value of the output.

Table 109. Common export key parameters subsection (X'0003') of trusted block rule section (X'12') (continued)		
Offset (bytes)	Length (bytes)	Description
011+xxx	001	<p>CV length in bytes (yyy).</p> <ul style="list-style-type: none"> If the length is not 0, 8, or 16, return an error. If the length is 0, and if the source key is a CCA DES key-token, preserve the CV in the symmetric encrypted output if the output is to be in the form of a CCA DES key-token. If a non-zero length is less than the length of the key that is identified by the <i>source_key_identifier</i> parameter, return an error. If the length is 16, and if the CV (offset 012+xxx) is valued to 16 bytes of X'00' (ignoring the key-part bit), then: <ol style="list-style-type: none"> Ignore all CV bit definitions. If CCA DES key-token format, set the flag byte of the symmetric encrypted output key to indicate that a CV value is present. If the source key is 8 bytes, do not replicate the key to 16 bytes.
012+xxx	yyy	<p>CV.</p> <p>Place this CV into the output exported key-token, if the symmetric encrypted output key format selected (offset 018 in rule section) is CCA DES key-token.</p> <ul style="list-style-type: none"> If the symmetric encrypted output key format flag (offset 018 in section X'12') indicates return an RKX key-token (X'00'), then ignore this CV. Otherwise, exclusive-OR this CV into the cleartext transport key. Exclusive-OR the CV of the source key into the cleartext transport key if the CV length (offset 011+xxx) is set to 0. If a transport key to encrypt a source key has equal left and right key halves, return an error. Replicate the key halves of the key that is identified by the <i>source_key_identifier</i> parameter whenever all of these conditions are met: <ol style="list-style-type: none"> The Replicate Key command (offset X'00DB') is enabled in the active role. The CV length (offset 011+xxx) is 16, and both CV halves are non-zero. The <i>source_key_identifier</i> parameter (contained in either a CCA DES key-token or RKX key-token) identifies an 8-byte key. The key-form bits (40 - 42) of this CV do not indicate a single-length key (are not set to zero) Key-form bit 40 of this CV does not indicate that the key is to have guaranteed unique halves (is not set to 1). <p>Note: A transport key is not used when the symmetric encrypted output key is in RKX key-token format.</p>

Note: See “Number representation in trusted blocks” on page 388.

Trusted block section X'12' subsection X'0004'

Subsection X'0004' of the trusted block rule section (X'12') is the source key rule reference TLV object. This subsection is optional, but is required if using an RKX key-token as a source key (identified by *source_key_identifier* parameter). It contains the rule ID for the rule that is used to create the export key. If this subsection is not present, an RKX key-token format source key will not be accepted for use.

This subsection is defined in the following table:

Table 110. Source key rule reference subsection (X'0004' of trusted block rule section (X'12'))		
Offset (bytes)	Length (bytes)	Description
000	002	<p>Subsection tag:</p> <p>X'0004'</p> <p>Source key rule reference TLV object.</p>
002	002	Subsection length in bytes (14).
004	001	Subsection version number (X'00').
005	001	Reserved, must be binary zero.

Table 110. Source key rule reference subsection (X'0004' of trusted block rule section (X'12') (continued)		
Offset (bytes)	Length (bytes)	Description
006	008	<p>Rule ID.</p> <p>Rule identifier for the rule that must have been used to create the source key.</p> <p>The Rule ID is an 8-byte string of ASCII characters, left-justified and padded on the right with space characters. Acceptable characters are A...Z, a...z, 0...9, - (X'2D'), and _ (X'5F'). All other characters are reserved for future use.</p>

Note: See “Number representation in trusted blocks” on page 388.

Trusted block section X'12' subsection X'0005'

Subsection X'0005' of the trusted block rule section (X'12') is the export key CCA token parameters TLV object. This subsection is optional. It contains a mask length, mask, and template for the export key CV limit. It also contains the template length and template for the source key label. When using a CCA DES key-token as a source key input parameter, its key type can be "filtered" by using the export key CV limit mask (offset 005) and limit template (offset 005+yyy) in this subsection.

This subsection is defined in the following table:

Table 111. Export key CCA token parameters subsection (X'0005') of trusted block rule section (X'12')		
Offset (bytes)	Length (bytes)	Description
000	002	<p>Subsection tag:</p> <p>X'0005'</p> <p>Export key CCA token parameters TLV object.</p>
002	002	Subsection length in bytes (10+yyy+yyy+zzz).
004	001	Subsection version number (X'00').
005	002	Reserved, must be binary zero.
007	001	Flags (must be set to binary zero).
008	001	<p>Export key CV limit mask length in bytes (yyy).</p> <p>Do not use CV limits if this CV limit mask length (yyy) is zero. Use CV limits if yyy is non-zero, in which case yyy:</p> <ul style="list-style-type: none"> • Must be 8 or 16. • Must not be less than the export key minimum length (offset 008 in subsection X'0003'). • Must be equal in length to the actual source key length of the key. <p>Example: An export key minimum length of 16 and an export key CV limit mask length of 8 returns an error.</p>
009	yyy	<p>Export key CV limit mask (does not exist if yyy=0).</p> <p>Indicates which CV bits to check against the source key CV limit template (offset 009+yyy).</p> <p>Examples: A mask of X'FF' means check all bits in a byte. A mask of X'FE' ignores the parity bit in a byte.</p>

Table 111. Export key CCA token parameters subsection (X'0005') of trusted block rule section (X'12') (continued)		
Offset (bytes)	Length (bytes)	Description
009+yyy	yyy	<p>Export key CV limit template (does not exist if yyy=0).</p> <p>Specifies the required values for those CV bits that are checked based on the export key CV limit mask (offset 009).</p> <p>The export key CV limit mask and template have the same length, yyy. This is because these two variables work together to restrict the acceptable CVs for CCA DES key tokens to be exported. The checks work as follows:</p> <ol style="list-style-type: none"> 1. If the length of the key to be exported is less than yyy, return an error. 2. Logical AND the CV for the key to be exported with the export key CV limit mask. 3. Compare the result to the export key CV limit template. 4. Return an error if the comparison is not equal. <p>Examples: An export key CV limit mask of X'FF' for CV byte 1 (key type) along with an export key CV limit template of X'3F' (key type CVARENC) for byte 1 filters out all key types except CVARENC keys.</p> <p>Note: Using the mask and template to permit multiple key types is possible, but cannot consistently be achieved with one rule section. For example, setting bit 10 to 1 in the mask and the template permits PIN processing keys and cryptographic variable encrypting keys, and only those keys. However, a mask to permit PIN-processing keys and key-encrypting keys, and only those keys, is not possible. In this case, multiple rule sections are required, one to permit PIN-processing keys and the other to permit key-encrypting keys.</p>
009+ yyy+ yyy	001	<p>Source key label template length in bytes (zzz).</p> <p>Valid values are 0 and 64. Return an error if the length is 64 and a source key label is not provided.</p>
010+ yyy+ yyy	zzz	<p>Source key label template (does not exist if zzz=0).</p> <p>If a key label is identified by the <i>source_key_identifier</i> parameter, verify that the key label name matches this template. If the comparison fails, return an error. The source key label template must conform to the following rules:</p> <ul style="list-style-type: none"> • The key label template must be 64 bytes. • The first character cannot be in the range X'00' - X'1F', nor can it be X'FF'. • The first character cannot be numeric (X'30' - X'39'). • A key label name is terminated by a space character (X'20') on the right and must be padded on the right with space characters. • The only special characters that are permitted are #, \$, @, and * (X'23', X'24', X'40', and X'2A'). • The wildcard X'2A' (*) is only permitted as the first character, the last character, or the only character in the template. • Only alphanumeric characters (a...z, A...Z, 0...9), the four special characters (X'23', X'24', X'40', and X'2A'), and the space character (X'20') are allowed.

Note: See “Number representation in trusted blocks” on page 388.

Trusted block section X'13'

Trusted block section X'13' contains the name (key label). The trusted block name section provides a 64-byte variable to identify the trusted block, just as key labels are used to identify other CCA keys. This name, or label, enables a host access-control system such as RACF to use the name to verify that the application has authority to use the trusted block.

Section X'13' is optional. No multiple sections are allowed. It has no subsections that are defined. This section is defined in the following table:

Table 112. Trusted block key label (name) section X'13'		
Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: X'13' Trusted block name (key label).
001	001	Section version number (X'00').
002	002	Section length in bytes (68).
004	064	Name (key label).

Note: See “Number representation in trusted blocks” on page 388.

Trusted block section X'14'

Trusted block section X'14' contains control and security information that is related to the trusted block. This information section is separate from the public key and other sections because this section is required while the others are optional. This section contains the cryptographic information that guarantees its integrity and binds it to the local system.

Section X'14' is required. No multiple sections are allowed. Two subsections are defined. This section is defined in the following table:

Table 113. Trusted block information section X'14'		
Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: X'14' Trusted block information.
001	001	Section version number (X'00').
002	002	Section length in bytes (10+xxx).
004	002	Reserved, binary zero.
006	004	Flags: X'00000000' Trusted block is in the inactive state. X'00000001' Trusted block is in the active state.
010	xxx	Information section subsections (tag-length-value objects). One or two objects in TLV format.

Note: See “Number representation in trusted blocks” on page 388.

Section X'14' has two information subsections (tag-length-value objects) defined. These subsections are summarized in the following table:

Table 114. Summary of trusted block information subsections			
Rule subsection tag	TLV object	Optional or required	Comments
X'0001'	Protection information	Required	Contains the encrypted 8-byte confounder and triple-length (24-byte) MAC key, the ISO 16609 TDES CBC MAC value, and the MKVP of the PKA master key (computed by using MDC4).

Table 114. Summary of trusted block information subsections (continued)			
Rule subsection tag	TLV object	Optional or required	Comments
X'0002'	Activation and expiration dates	Optional	Contains flags indicating whether the coprocessor is to validate dates, and contains the activation and expiration dates that are considered valid for the trusted block.

Note: See “Number representation in trusted blocks” on page 388.

Trusted block section X'14' subsection X'0001'

Subsection X'0001' of the trusted block information section (X'14') is the protection information TLV object. This subsection is required. It contains the encrypted 8-byte confounder and triple-length (24-byte) MAC key, the ISO-16609 TDES CBC MAC value, and the MKVP of the PKA master key (computed by using MDC4).

This subsection is defined in the following table:

Table 115. Protection information subsection (X'0001') of trusted block information section (X'14')		
Offset (bytes)	Length (bytes)	Description
000	002	Subsection tag: X'0001' Trusted block information TLV object.
002	002	Subsection length in bytes (62).
004	001	Subsection version number (X'00').
005	001	Reserved, must be binary zero.
006	032	Encrypted MAC key. Contains the encrypted 8-byte confounder and triple-length (24-byte) MAC key in the following format: <div> Offset Description 00 - 07 Confounder. 08 - 15 Left key. 16 - 23 Middle key. 24 - 31 Right key. </div>
038	008	MAC. Contains the ISO-16609 TDES CBC message authentication code value.
046	016	MKVP. Contains the PKA master key verification pattern, computed by using MDC4, when the trusted block is in internal form, otherwise contains binary zero.

Note: See “Number representation in trusted blocks” on page 388.

Trusted block section X'14' subsection X'0002'

Subsection X'0002' of the trusted block information section (X'14') is the activation and expiration dates TLV object. This subsection is optional. It contains flags indicating whether the coprocessor is to validate dates, and contains the activation and expiration dates that are considered valid for the trusted block.

This subsection is defined in the following table:

Table 116. Activation and expiration dates subsection (X'0002') of trusted block information section (X'14')		
Offset (bytes)	Length (bytes)	Description
000	002	Subsection tag: X'0002' Activation and expiration dates TLV object.
002	002	Subsection length in bytes (16).
004	001	Subsection version number (X'00').
005	001	Reserved, must be binary zero.
006	002	Flags: X'0000' The coprocessor does not check dates. X'0001' The coprocessor checks dates. Compare the activation date (offset 008) and the expiration date (offset 012) to the coprocessor's internal real-time clock. Return an error if the coprocessor date is before the activation date or after the expiration date.
008	004	Activation date. Contains the first date that the trusted block can be used for generating or exporting keys. Format of the date is YYMD, where: YY Big-endian year (return an error if greater than 9999). M Month (return an error if any value other than X'01' - X'0C'). D Day of month (return an error if any value other than X'01' - X'1F'; day must be valid for given month and year, including leap years). Return an error if the activation date is after the expiration date or is not valid.
012	004	Expiration date. Contains the last date that the trusted block can be used. Same format as activation date (offset 008). Return an error if date is not valid.

Note: See [“Number representation in trusted blocks”](#) on page 388.

Trusted block section X'15'

Trusted block section X'15' contains application-defined data. The trusted block application-defined data section can be used to include application-defined data in the trusted block. The purpose of the data in this section is defined by the application; it is not examined or used by CCA in any way.

Section X'15' is optional. No multiple sections are allowed. It has no subsections that are defined. This section is defined in the following table:

Table 117. Trusted block application-defined data section X'15'		
Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: X'15' Application-defined data.
001	001	Section version number (X'00').
002	002	Section length (6+xxx).

Table 117. Trusted block application-defined data section X'15' (continued)		
Offset (bytes)	Length (bytes)	Description
004	002	Application data length (xxx). The value of xxx can be from 0 bytes to a length that does not cause the trusted block to exceed its maximum size of 3500 bytes.
006	xxx	Application-defined data. Can be used to hold a public-key certificate for the trusted public key.

Note: See [“Number representation in trusted blocks”](#) on page 388.

Data areas

These topics present the format of the Cryptographic Communication Vector Table (CCVT) and the Cryptographic Communication Vector Table Extension (CCVE) data areas.

The Cryptographic Communication Vector Table (CCVT)

The CCVT is the ICSF base control block and contains addresses of common areas for use by ICSF components. Indicators in the CCVT also provide ICSF status information. The CCVT is getmained in subpool 245 under the line. The ICSF CCVT is anchored off of SCVTCCVT in the SCVT macro.

ONLY these fields are part of the programming interface:

- CCVT_CKDSN
- CCVT_PKDSN
- CCVT_STATF
- CCVT_TKDSN
- CCVT_USERPARM
- CCVTCCVE
- CCVTDACC
- CCVTFMID
- CCVTHFLG
- CCVTID
- CCVTINS2
- CCVTINST
- CCVTLNTH
- CCVTNAMES
- CCVTPRPC
- CCVTRLVL
- CCVTSERVCNT
- CCVTSFLG
- CCVTVER

Table 118 on page 401 describes the contents of the Cryptographic Communication Vector Table. Any bits that are not described in the table are reserved.

Table 118. Cryptographic communication vector table

Offset (Dec)	Number of bytes	Field name	Description
0	4	CCVTID	Eyecatcher ('CCVT').
4	2	CCVTVER	CCVT version number - 2 printable EBCDIC digits.
9	1	CCVT_STATF	Statistic option flags.
14	2	CCVTRLVL	ICSF level.
16	4	CCVTCCVE	Cryptographic Communication Vector Table Extension (CCVE) address. The address of a private area extension of the CCVT. You should place any fields not needed by other address spaces in the CCVE.
28	4	CCVTPRPC	Entry point for the pre-PC processing module, CSFARPC.
32	4	CCVTINST	For installation use.
56	8	CCVTINS2	An 8-byte area for installation use.
68	4	CCVTLNTH	Maximum installation data length.
80	1	CCVTHFLG	Flag bytes. Bit Meaning When Set On 0 Crypto assist instructions available. 1 Additional secure Crypto device available. 2 Support for 64-bit callers. 3 ICSF Cross-System Services environment is active for CKDS. 4 ICSF Cross-System Services environment is active for TKDS. 5 RSA 4096-bit function enabled and the RNGL service is available. 6 Secure key AES is available. 7 AES master key is active.

Table 118. Cryptographic communication vector table (continued)

Offset (Dec)	Number of bytes	Field name	Description
81	1	CCVTSFLG	<p>Flag bytes.</p> <p>Bit Meaning When Set On</p> <p>0 ICSF during initialization.</p> <p>1 ICSF was able to complete cleanup, so no EOM cleanup is needed.</p> <p>2 PKCS #11 operating in FIPS standard mode (only applies to FIPSMODE(YES) and FIPSMODE(NO)).</p> <p>3 PKCS #11 operating in FIPS compatibility mode (only applies to FIPSMODE(COMPAT) and FIPSMODE(NO)).</p> <p>4 Reserved for ICSF use.</p> <p>5 ICSF is available to be called.</p>
136	8	CCVTFMID	ICSF FMID.
144	8	CCVT_USERPARM	ICSF user parameter.
276	4	CCVTDACC	ICSF DAC instructions control block for RMF.
484	44	CCVT_PKDSN	Name of the active PKDS. If no PKDS was specified, the first character will be an EBCDIC blank (X'40').
528	4	CCVTNAMES	Array of 8-character callable service names, indexed in the range 1 to CCVTSERV CNT, inclusive, by the 'IBM assigned service number' (see CSFASPB and CSFZIXIB).
704	44	CCVT_CKDSN	Name of the active CKDS. If no CKDS was specified, the first character will be an EBCDIC blank (X'40').
748	44	CCVT_TKDSN	Name of the active TKDS.
792	2	CCVTSERV CNT	Maximum defined index of CCVTNAMES.

The Cryptographic Communication Vector Table Extension (CCVE)

The CCVE is an extension of the CCVT that contains fields that can exist. The CCVE exists in ICSF extended private. It should contain any ICSF base control block fields that are not needed by other address spaces.

ONLY these fields are part of the programming interface:

- CCVEINPL
- CCVEINPP
- CCVESECC
- CCVTID
- CCVTSERV CNT
- CCVTVER

Table 119 on page 403 describes the contents of the Cryptographic Communication Vector Table Extension. Any bits that are not described in the table are reserved.

<i>Table 119. Cryptographic Communication Vector Table Extension</i>			
Offset (Dec)	Number of Bytes	Field Name	Description
0	4	CCVTID	Eyecatcher ('CCVT').
4	2	CCVTVR	CCVT version number - 2 printable EBCDIC digits.
328	4	CCVEINPP	Pointer to installation optional parameter.
332	4	CCVEINPL	Length of the installation optional parameter.
372	8	CCVESECC	Reserved for security exit.
792	2	CCVTSERV CNT	Maximum defined index of CCVTNAMES.

Generic Service Table (CSFMGST)

Table 120 on page 403 describes the format of the generic service table, a control block that is used to control the call of installation-defined services.

<i>Table 120. Generic Service Table Block Format</i>		
Offset (Dec)	Number of Bytes	Description
0	4	EBCDIC ID.
4	2	Version number.
6	2	Length of the MGST.
8	4	Number of entries in the array.
12	4	Subpool this table is in.
16	4	Reserved.
20	4	Reserved.
24	4	Reserved.
28	4	Reserved.
Variable Section of the MGST (Repeat for each entry in the array)		
0	8	IBM-assigned name.
8	8	Installation-assigned name.

Table 120. Generic Service Table Block Format (continued)

Offset (Dec)	Number of Bytes	Description
16	4	<p>Flags.</p> <p>Bit Meaning When Set On</p> <p>0 Service has been requested by the installation.</p> <p>1 Service has been loaded.</p> <p>2 Service is active.</p> <p>3 Service is required.</p> <p>4 Service is UDX.</p>
20	4	Address of the service.
24	4	Installation-assigned service number.
28	4	Reserved.

RMF measurements table

The RMF measurements table consists of the DACC and DACC extension (DACC version 06 (DACC_VER) and later). The address of the DACC extension block will be CCVTDACC (pointer in the CSFCCVT macro) + DACC_LEN (the overall length of the DACC block).

Table 121 on page 404 describes the contents of the performance measurements for RMF. The count fields are double-word length.

Table 121. RMF measurements record format

Offset (Dec)	Number of bytes	Field name	Description
0	4	DACC_ID	The DACC ID.
4	4	DACC_VER	The version number of the block.
8	4	DACC_LEN	The control block length.
12	2	DACC_ENT_CNT	Number of entries.
14	2	DACC_ENT_LEN	Length of each entry.
16	8	DACC_ENT_ID	<p>Identifier of count array - character 'ENCSDDES'. The Encipher service will collect data as follows:</p> <ul style="list-style-type: none"> Collection for single DES is done separately. The number of service calls, number of bytes of data enciphered, and the number of hardware instructions used to encipher the data will be collected.
24	8	DACC_ENT_SVC_CNT	Count of ENCSDDES service calls.
32	8	DACC_ENT_BYT_CNT	Count of ENCSDDES bytes processed.

Table 121. RMF measurements record format (continued)			
Offset (Dec)	Number of bytes	Field name	Description
40	8	DACC_ENT_INT_CNT	Count of ENCSDES instructions.
48	8	DACC_ENT_ID	Identifier of count array - character 'ENCTDES'. The Encipher service will collect data as follows: <ul style="list-style-type: none"> Double and triple DES will be counted together. The number of service calls, number of bytes of data enciphered, and the number of hardware instructions used to encipher the data will be collected.
56	8	DACC_ENT_SVC_CNT	Count of ENCTDES service calls.
64	8	DACC_ENT_BYT_CNT	Count of ENCTDES bytes processed.
72	8	DACC_ENT_INT_CNT	Count of ENCTDES instructions.
80	8	DACC_ENT_ID	Identifier of count array - character DECSDES. The Decipher service will collect data as follows: <ul style="list-style-type: none"> Collection for single DES is done separately. The number of service calls, number of bytes of data deciphered, and the number of hardware instructions used to decipher the data will be collected.
88	8	DACC_ENT_SVC_CNT	Count of DECSDES service calls.
96	8	DACC_ENT_BYT_CNT	Count of DECSDES bytes processed.
104	8	DACC_ENT_INT_CNT	Count of DECSDES instructions.
112	8	DACC_ENT_ID	Identifier of count array - character DECTDES. The Decipher service will collect data as follows: <ul style="list-style-type: none"> Double and triple DES will be counted together. The number of service calls, number of bytes of data deciphered, and the number of hardware instructions used to decipher the data will be collected.
120	8	DACC_ENT_SVC_CNT	Count of DECTDES service calls.
128	8	DACC_ENT_BYT_CNT	Count of DECTDES bytes processed.
136	8	DACC_ENT_INT_CNT	Count of DECTDES instructions.
144	8	DACC_ENT_ID	Identifier of count array - character MACGEN. The MAC Generate service will collect data as follows: <ul style="list-style-type: none"> Single and various double key MAC will be gathered together. The number of service calls, number of bytes of data MAC'd, and the number of instructions will be collected.
152	8	DACC_ENT_SVC_CNT	Count of MACGEN service calls.
160	8	DACC_ENT_BYT_CNT	Count of MACGEN bytes processed.
168	8	DACC_ENT_INT_CNT	Count of MACGEN instructions.

Table 121. RMF measurements record format (continued)

Offset (Dec)	Number of bytes	Field name	Description
176	8	DACC_ENT_ID	Identifier of count array - character MACVER. The MAC Verify service will collect data as follows: <ul style="list-style-type: none"> Single and various double key MAC will be gathered together. The number of service calls, number of bytes of data MAC'd, and the number of instructions will be collected.
184	8	DACC_ENT_SVC_CNT	Count of MACVER service calls.
192	8	DACC_ENT_BYT_CNT	Count of MACVER bytes processed.
200	8	DACC_ENT_INT_CNT	Count of MACVER instructions.
208	8	DACC_ENT_ID	Identifier of count array - character OWH. The One Way Hash service will collect data as follows: <ul style="list-style-type: none"> For SHA-1, the number of service calls, number of bytes of bytes of data hashed, and the number of instructions will be collected.
216	8	DACC_ENT_SVC_CNT	Count of OWH service calls.
224	8	DACC_ENT_BYT_CNT	Count of OWH bytes processed.
232	8	DACC_ENT_INT_CNT	Count of OWH instructions.
240	8	DACC_ENT_ID	Identifier of count array - character PTR. The Encrypted PIN Translate, Encrypted PIN Translate2, and Encrypted PIN Translate Enhanced services will collect data as follows: <ul style="list-style-type: none"> Collect the number of service calls only.
248	8	DACC_ENT_SVC_CNT	Count of PTR, PTR2, and PTRE service calls.
256	16		Reserved.
272	8	DACC_ENT_ID	Identifier of count array - character PVR. The PIN Verify and the PIN Verify2 services will collect data as follows: <ul style="list-style-type: none"> Collect the number of service calls only.
280	8	DACC_ENT_SVC_CNT	Count of PVR and PVR2 service calls.
288	16		Reserved.
304	8	DACC_ENT_ID	Identifier of count array - character OWH256. The One Way Hash service will collect data as follows: <ul style="list-style-type: none"> For SHA-224 and SHA-256, the number of service calls, number of bytes of data hashed, and the number of instructions will be collected.
312	8	DACC_ENT_SVC_CNT	Count of OWH service calls for SHA-224 and SHA-256.
320	8	DACC_ENT_BYT_CNT	Count of OWH bytes processed for SHA-224 and SHA-256.
328	8	DACC_ENT_INT_CNT	Count of OWH instructions for SHA-224 and SHA-256.

Table 121. RMF measurements record format (continued)			
Offset (Dec)	Number of bytes	Field name	Description
336	8	DACC_ENT_ID	Identifier of count array - character OWH512. The One Way Hash service will collect data as follows: <ul style="list-style-type: none"> For SHA-384 and SHA-512, the number of service calls, number of bytes of data hashed, and the number of instructions will be collected.
344	8	DACC_ENT_SVC_CNT	Count of OWH service calls for SHA-384 and SHA-512.
352	8	DACC_ENT_BYT_CNT	Count of OWH bytes processed for SHA-384 and SHA-512.
360	8	DACC_ENT_INT_CNT	Count of OWH instructions for SHA-384 and SHA-512.
368	8	DACC_ENT_ID	Identifier of count array - character 'ENCAES'. The Symmetric algorithm encipher service will collect data as follows: The number of service calls, number of bytes of data enciphered, and the number of instructions used to encipher the data will be collected.
376	8	DACC_ENT_SVC_CNT	Count of SAE service calls
384	8	DACC_ENT_BYT_CNT	Count of ENCAES bytes processed
392	8	DACC_ENT_INT_CNT	Count of ENCAES instruction
400	8	DACC_ENT_ID	Identifier of count array - character 'DECAES'. The Symmetric algorithm decipher service will collect data as follows: the number of service calls, number of bytes of data deciphered, and the number of instructions used to decipher the data will be collected.
408	8	DACC_ENT_SVC_CNT	Count of SAD service calls
416	8	DACC_ENT_BYT_CNT	Count of DECAES bytes processed
424	8	DACC_ENT_INT_CNT	Count of DECAES instruction
432	8	DACC_ENT_ID	Identifier of count array - character 'DSGRSA'. The Digital Signature Generate service will collect the number of service calls processed to generate a digital signature using an RSA private key.
440	8	DACC_ENT_SVC_CNT	Count of DSG service calls using an RSA private key
448	16		Reserved
464	8	DACC_ENT_ID	Identifier of count array - character 'DSGECC'. The Digital Signature Generate service will collect the number of service calls processed to generate a digital signature using an ECC private key.
472	8	DACC_ENT_SVC_CNT	Count of DSG service calls using an ECC private key
480	16		Reserved
496	8	DACC_ENT_ID	Identifier of count array - character 'DSVRSA'. The Digital Signature Verify service will collect the number of service calls processed to verify a digital signature using an RSA private key.

Table 121. RMF measurements record format (continued)

Offset (Dec)	Number of bytes	Field name	Description
504	8	DACC_ENT_SVC_CNT	Count of DSV service calls using an RSA private key
512	16		Reserved
528	8	DACC_ENT_ID	Identifier of count array - character 'DSVECC'. The Digital Signature Verify service collects the number of service calls processed to verify a digital signature using an ECC private key.
536	8	DACC_ENT_SVC_CNT	Count of DSV service calls using an ECC private key
544	16		Reserved
560	8	DACC_ENT_ID	Identifier of count array - character 'MACGEN2'. The MAC Generate2 service collects data as follows: <ul style="list-style-type: none"> • The number of service calls. • The number of bytes of data MACed. • The number of instructions used to MAC the data.
568	8	DACC_ENT_SVC_CNT	Count of MACGEN2 service calls.
576	8	DACC_ENT_BYT_CNT	Count of MACGEN2 bytes processed.
584	8	DACC_ENT_INT_CNT	Count of MACGEN2 instructions.
592	8	DACC_ENT_ID	Identifier of count array - character 'MACVER2'. The MAC Verify2 service collects data as follows: <ul style="list-style-type: none"> • The number of service calls. • The number of bytes of data MACed. • The number of instructions used to MAC the data.
600	8	DACC_ENT_SVC_CNT	Count of MACVER2 service calls.
608	8	DACC_ENT_BYT_CNT	Count of MACVER2 bytes processed.
616	8	DACC_ENT_INT_CNT	Count of MACVER2 instructions.
624	8	DACC_ENT_ID	Identifier of count array - character 'FPEE'. The FPE encipher service collects data as follows: The number of service calls, number of bytes of data encrypted, and the number of instructions used to encipher the data.
632	8	DACC_ENT_SVC_CNT	Count of FPEE service calls.
640	8	DACC_ENT_BYT_CNT	Count of FPEE bytes processed.
648	8	DACC_ENT_INT_CNT	Count of FPEE instructions.
656	8	DACC_ENT_ID	Identifier of count array - character 'FPED'. The FPE decipher service collects data as follows: The number of service calls, number of bytes of data decrypted, and the number of instructions used to decrypt the data.
664	8	DACC_ENT_SVC_CNT	Count of FPED service calls.
672	8	DACC_ENT_BYT_CNT	Count of FPED bytes processed.
680	8	DACC_ENT_INT_CNT	Count of FPED instructions.

Table 121. RMF measurements record format (continued)			
Offset (Dec)	Number of bytes	Field name	Description
688	8	DACC_ENT_ID	Identifier of count array - character 'FPET'. The FPE translate service collects data as follows: The number of service calls, number of bytes of data translated, and the number of instructions used to translate the data.
696	8	DACC_ENT_SVC_CNT	Count of FPET service calls.
704	8	DACC_ENT_BYT_CNT	Count of FPET bytes processed.
712	8	DACC_ENT_INT_CNT	Count of FPET instructions.
720	8	DACC_ENT_ID	Identifier of count array - character 'FFXE'. The FFX encipher service collects data as follows: The number of service calls, number of bytes of data encrypted, and the number of instructions used to encipher the data.
728	8	DACC_ENT_SVC_CNT	Count of FFXE service calls.
736	8	DACC_ENT_BYT_CNT	Count of FFXE bytes processed.
744	8	DACC_ENT_INT_CNT	Count of FFXE instructions.
752	8	DACC_ENT_ID	Identifier of count array - character 'FFXD'. The FFX decipher service collects data as follows: The number of service calls, number of bytes of data decrypted, and the number of instructions used to decrypt the data.
760	8	DACC_ENT_SVC_CNT	Count of FFXD service calls.
768	8	DACC_ENT_BYT_CNT	Count of FFXD bytes processed.
776	8	DACC_ENT_INT_CNT	Count of FFXD instructions.
784	8	DACC_ENT_ID	Identifier of count array - character 'FFXT'. The FFX translate service collects data as follows: The number of service calls, number of bytes of data translated, and the number of instructions used to translate the data.
792	8	DACC_ENT_SVC_CNT	Count of FFXT service calls.
800	8	DACC_ENT_BYT_CNT	Count of FFXT bytes processed.
808	8	DACC_ENT_INT_CNT	Count of FFXT instructions.
816	8	DACC_ENT_ID	Identifier of count array - character 'DSGQSA'. The Digital Signature Generate service collects the number of service calls processed to generate a digital signature using a ML-DSA, Dilithium private key.
824	8	DACC_ENT_SVC_CNT	Count of DSG service calls using a ML-DSA, Dilithium private key.
832	16		Reserved.
848	8	DACC_ENT_ID	Identifier of count array - character 'DSVQSA'. The Digital Signature Verify service collects the number of service calls processed to verify a digital signature using a ML-DSA, Dilithium private key.
856	8	DACC_ENT_SVC_CNT	Count of DSV service calls using a ML-DSA, Dilithium private key.

Table 121. RMF measurements record format (continued)

Offset (Dec)	Number of bytes	Field name	Description
864	16		Reserved.

RMF measurements extension

The address of the DACC extension block will be CCVTDACC (pointer in the CSFCCVT macro) + DACC_LEN (the overall length of the DACC block).

Version '01' of the DACC extension block contains the domain ID of the LPAR in use.

Table 122. RMF measurements extension

Offset (Dec)	Number of bytes	Field name	Description
0	4	DACC_EX_ID	Extension identifier 'DACX' (EBCDIC).
4	2	DACC_EXT_VER	Version number '01' (EBCDIC).
6	2	DACC_EXT_LEN	Length of the extension (integer).
8			Start of data area.
8	1	DACC_EXT_DOMAIN_ID	Domain id of the LPAR is use (integer).
9	3		Reserved.

Appendix B. ICSF SMF records

This topic contains the ICSF SMF records as well as non-ICSF SMF records that contain crypto-related fields. For details on customizing the crypto fields for the non-ICSF records, see *z/OS Compliance Data Collection*.

ICSF records

Record type 82 (X'52') - ICSF record.

Non-ICSF records

- Record type 1154 (X'482') Subtype 49 - ICSF Compliance Evidence.
- Record type 1154 (X'482') Subtype 128 - ICSF Processor Activity Compliance Evidence.
- Record type 0 (X'00') - IPL.
 - SMFORST and SMFORS4K fields are added at offsets 82 - 87 to indicate the number of crypto and NNPI counters supported by the system (APAR OA61511).
- SMF record type 30 (X'1E') - Common address space work.
 - The Crypto counters sections contain the counters for the CPACF cryptographic instructions used by a job for the period that the record represents.

Record type 82 (X'52') - ICSF record

Record type 82 is used to record information about the events and operations of the Integrated Cryptographic Service Facility (ICSF) program product. Record type 82 is written to the SMF data set at the completion of certain cryptographic functions:

- **Subtype 1** — is written whenever ICSF is started or the options refresh is performed.
- **Subtype 7** — is written when an operational key is imported from a coprocessor.
- **Subtype 8** — is written whenever the in-storage copy of the CKDS is refreshed.
- **Subtype 9** — is written whenever the CKDS is updated by a dynamic CKDS update service or the KDS Metadata write service or the CKDS KEYS utility.
- **Subtype 13** — is written whenever the PKDS is updated by a dynamic PKDS update service or the KDS Metadata write service or the PKDS KEYS utility.
- **Subtype 14** — is written when a clear master key part is entered on a cryptographic coprocessor.
- **Subtype 15** — is written whenever a retained key is created or deleted.
- **Subtype 16** — is written for each request and reply from calls to the CSFPCI service.
- **Subtype 18** — is written when the configuration of a coprocessor or accelerator changes.
- **Subtype 19** — no longer written.
- **Subtype 20** — is written periodically to record processing times for coprocessors or accelerators.
- **Subtype 21** — is written when ICSF issues IXCJOIN to join the ICSF sysplex group or issues IXCLEAVE to leave the sysplex group.
- **Subtype 22** — is written when the Trusted Block Create Callable services are invoked.
- **Subtype 23** — is written when the token data set (TKDS) is updated
- **Subtype 24** — is written when duplicate tokens are found.
- **Subtype 25** — is written when key store policy checking detects the unauthorized use of a key token.
- **Subtype 26** — is written whenever the in-storage copy of the PKDS is refreshed.
- **Subtype 27** — is written when key store policy PKA key extensions checking detects the unauthorized use of a key.
- **Subtype 28** — is written for information about High Performance Encrypted Key.

Record Type 82

- **Subtype 29** — is written for each TKE workstation audit record received from a TKE workstation.
- **Subtype 30** — is written for each time an archived or inactive key data set record is referenced.
- **Subtype 31** — is written for cryptographic statistics data.
- **Subtype 40** — is written for lifecycle events related to symmetric CCA tokens and TR-31 key blocks. This replaces subtype 9.
- **Subtype 41** — is written for lifecycle events related to asymmetric CCA tokens. This replaces subtype 13.
- **Subtype 42** — is written for lifecycle events related to PKCS#11 objects. This replaces subtype 23.
- **Subtype 44** — is written for usage events related to symmetric CCA tokens and TR-31 key blocks.
- **Subtype 45** — is written for usage events related to asymmetric CCA tokens.
- **Subtype 46** — is written for usage events related to PKCS#11 objects.
- **Subtype 47** — is written for supported PKCS #11 usage events which do not involve an object.
- **Subtype 48** — is written for compliance warning events.
- **Subtype 49** — is written for master key events that result in a new master key value being promoted to current.

Macro to Symbolically Address Record Type 82: The SMF record mapping macro for ICSF type 82 record is CSFZSM82.

The mapping macro, CSFZSM82, resides in SYS1.MODGEN.

Record environment

The following conditions exist for the generation of each of the subtypes of this record:

Macro

Subtype Macro

1

SMFWTM (record exit: IEFU83).

3,4,5,6,7,8

SMFEWTM,BRANCH=YES,MODE=XMEM (record exit: IEFU85).

Record mapping

Two different record formats are produced by ICSF; one format that applies to subtypes smaller than 40 and another format that applies to subtypes 40 and higher.

For subtypes smaller than 40, the SMF record header is followed by the subtype-specific information and optionally, the audit header and audit section.

Table 123. Format of an SMF Type 82 record for subtypes smaller than 40	
SMF format	
SMF record header	
Subtype information	
Audit section header (optional)	
Audit section (optional)	

For subtypes 40 and higher, the SMF record header is followed by an ICSF header, a main section that contains the subtype-specific information, and optionally, the audit header and audit section.

Table 124. Format of an SMF Type 82 record for subtypes 40 and higher

SMF format
SMF record header
ICSF header
Main section with subtype information
Audit section header (optional)
Audit section (optional)

SMF header

Table 125. SMF record header

Offsets	Name	Length	Format	Description
0	0 SMF82len	2	binary	Record length. This field and the next field (total of four bytes) form the RDW (record descriptor word).
2	2 SMF82seg	2	binary	Segment descriptor (see record length field).
4	4 SMF82flg	1	binary	System indicator: Bit Meaning When Set 0-2 Reserved 3-6 Version indicators 7 Reserved.
5	5 SMF82rty	1	binary	Record type 82 (X'52').
6	6 SMF82tme	4	binary	Time since midnight, in hundredths of a second, that the record was moved into the SMF buffer.
10	A SMF82dte	4	packed	Date when the record was moved into the SMF buffer, in the form 0cyydddF.
14	E SMF82sid	4	EBCDIC	System identification (from the SID parameter).
18	12 SMF82ssi	4	EBCDIC	Subsystem identification.
22	16 SMF82sty	2	binary	Record subtype.

ICSF header (for all subtypes 40 or greater)

Table 126. ICSF header (for all subtypes 40 or greater)

Offsets	Name	Length	Format	Description
Dec Hex				
0	0 SMF82IHDR_VER	1	binary	Version number of this record (X'01'). Incremented if a change is made to the record that is incompatible with the prior version.
1	1	1		Reserved.
2	2 SMF82IHDR_HLEN	2	binary	Length of this header.
4	4	4		Reserved.
8	8 SMF82IHDR_MAIN_OFF	2	binary	Offset from SMF82Ihdr to main section.
10	A SMF82IHDR_MAIN_LEN	2	binary	Length of main section.

Table 126. ICSF header (for all subtypes 40 or greater) (continued)					
Offsets		Name	Length	Format	Description
Dec	Hex				
12	C	SMF82IHDR_AUD_OFF	2	binary	Offset from SMF82Ihdr to audit section. If there is no audit section, this field is zero.
14	E	SMF82IHDR_AUD_LEN	2	binary	Length of audit section.
16	10	SMF82IHDR_End	0		End of ICSF header.

Main section (subtype information)

The data contained in the main section is specific to each subtype. The content of each subtype is described in later sections of this document.

Audit header and audit section

Provides optional server user or end user audit information. When auditing information is supplied, there may be a server user section only, an end user section only, or both a server user and end user section. The SMF82AUD_hdr_numsect field of the Audit Header indicates how many sections are present and the SMF82AUDSECT_type field in the section indicates the type of section. If both a server user section and an end user section are present, they can appear in either order.

Table 127. SMF type 82 server user or end user audit section					
Offsets		Name	Length	Format	Description
0	0	SMF82AUDSECT_type	4	EBCDIC	Type of the section that follows. Either: <ul style="list-style-type: none"> 'SERV' (for server user). 'USER' (for end user). The task-level ACEE is logged if one exists. Otherwise, the HOME address-space-level ACEE is logged. 'HOME' (for home user, only logged when there is a task-level ACEE). The HOME address-space-level ACEE is logged. 'SECO' (for secondary user, only logged when the HOME address space is different from the SECONDARY address space). The 'HOME' and 'SECO' sections may only be logged for the following subtypes: 40, 41, 42, 44, 45, 46, 47.
4	4	SMF82AUDSECT_numflds	2	Binary	Number of triples in this section
6	6	SMF82AUDSECT_totlen	2	Binary	Overall length of this section, including this header
8	8	Tag-Length-Value (TLV) triplets start here and are defined in Table 129 on page 415 . These repeat as many times as the SMF82AUDSECT_numflds field indicates.			

Table 128. Audit header				
Offsets	Name	Length	Format	Description
0	SMF82AUD_hdr_eye	4	EBCDIC	Eyecatcher 'AUDT'.
4	SMF82AUD_hdr_ver	2	EBCDIC	Version. Currently '01'.
6	SMF82AUD_hdr_hlen	2	Binary	Length of this header.

Table 128. Audit header (continued)				
Offsets	Name	Length	Format	Description
8	SMF82AUD_hdr_numsect	4	Binary	Number of audit sections following this header.
12	SMF82AUD_hdr_totlen	4	Binary	Length of all audit sections plus this header.

Each Tag-Length-Value (TLV) triplet is a structure that is called SMF82_triplet and is defined as follows. The values for the tags and the format and maximum length of the data are defined in [Table 130 on page 415](#).

Table 129. Tag-Length-Value (TLV) triplet structure (SMF82_triplet)					
Offsets		Name	Length	Format	Description
0	0	SMF82_triplet_tag	2	Binary	Tag of the information in this TLV
2	2	SMF82_triplet_tlen	2	Binary	Length of this TLV including these first two fixed fields
4	4	SMF82_triplet_val	*	Varies	Data for this TLV

The tag values and their corresponding information are described in the following table. The tag value is defined in the constant SMF82AUD_TAG_XXX and the maximum length in SMF82AUD_MAXLEN_XXX. For example, the tag for X500_IDN is SMF82AUD_TAG_X500_IDN and maximum length of the associated data is SMF82AUD_MAXLEN_X500_IDN.

Table 130. TLV triplet tag values					
Tag Value	Name	Length	Format	Description	
1 1	X500_IDN	0-255	EBCDIC	X.500 Certificate Issuer's Distinguished Name (ACEEX5PR->IDN)	
2 2	X500_SDN	0-255	EBCDIC	X.500 Certificate Subject's Distinguished Name (ACEEX5PR->SDN)	
10 A	IDID_USRI	1-246	UTF-8	X.500 Distinguished Name of distributed client end user (ACEEIDID-> IDID1UDN)	
11 B	IDID_USRF	1	Binary	Format of IDID_USRI (ACEEIDID->IDID1NMF) 0 Undetermined 1 Straight string 2 X.500 format	
12 C	IDID_REG	1-255	UTF-8	Name of the registry that authenticated the user (ACEEIDID->IDID1RN)	
14 E	USRI	8	EBCDIC	RACF user ID (ACEEUSRI)	
15 F	GRPN	8	EBCDIC	Connect group (ACEEGRPN)	
16 10	TRM_USER	8	EBCDIC	Terminal ID (ACEETRM)	
17 11	JOB_JBN	8	EBCDIC	Job name (JMRJOB)	
18 12	JOB_RST	4	Binary	Job entry time (JMRENTY) in hundredths of a second that the reader recognized the JOB statement for this job. This field can be zero. See 'Standard and extended SMF record headers' in <i>z/OS MVS System Management Facilities (SMF)</i> for a detailed description of the formatted headers.	

Table 130. TLV triplet tag values (continued)					
Tag Value	Name	Length	Format	Description	
26	1A JOB_RSD	4	Binary	Job entry date (JMREDATE) that the reader recognized the JOB statement for this job in the form 0CYYDDDF. This field can be zero. See 'Standard and extended SMF record headers' in <i>z/OS MVS System Management Facilities (SMF)</i> for a detailed description of the formatted headers.	
27	1B JOB_STOD	16	Binary	Job selection time (TOD clock) in STCKE format. May only be logged for subtypes 40, 41, 42, 44, 45, 46, 47.	
28	1C JOB_ID	8	EBCDIC	Job ID May only be logged for subtypes 40, 41, 42, 44, 45, 46, 47.	
34	22 JOB_UID	8	Binary	User-defined identification field (JMRUSEID)	
42	2A SEC	8	EBCDIC	Security label (TOKSCL)	

Tag-Length-Value (TLV) triplets

Some subtypes' main section consists of Tag-Length-Value (TLV) triplets. Tag-Length-Value triplets are in the following format with a total length of SMF82IHDR_MAIN_LEN:

Table 131. Tag-Length-Value triplets					
Offsets		Name	Length	Format	Description
Dec	Hex				
0	0	SMF82_triplet_tag	2	binary	The tag identifying the type of data that this triplet contains.
2	2	SMF82_triplet_tlen	2	binary	Length of this triplet including the tag and the length fields.
4	4	SMF82_triplet_val			The value for this triplet.

Service names used in SMF records

Appendix G, “Resource names for CCA and ICSF entry points,” on page 503 contains a list of service names used in SMF records.

Subtype 1 - Initialization/Options Refresh section

Table 132. Subtype 1 Initialization/Options Refresh				
Offsets	Name	Length	Format	Description
0	0 SMF82t01_CCVE_STATUS	4	binary	<p>Cryptographic communication vector table extension (CCVE) status bits.</p> <p>Bit</p> <p>Meaning When Set</p> <p>0 Special security mode allowed.</p> <p>1 Reserved.</p> <p>2 RNG Cache enabled.</p> <p>3-5 Reserved.</p> <p>6 SAF checking for authorized callers.</p> <p>7-14 Reserved.</p> <p>15 Reserved.</p> <p>16 Default wrapping for internal tokens is the enhanced method version 1.</p> <p>17 Default wrapping for external tokens is the enhanced method version 1.</p> <p>18 Key archive reference message.</p> <p>19 Default wrapping for internal tokens is the enhanced method version 3 (WRAPENH3).</p> <p>20 Default wrapping for external tokens is the enhanced method version 3 (WRAPENH3).</p> <p>21-31 Reserved.</p> <p>Notes:</p> <ul style="list-style-type: none"> • Bits 16 and 19 are mutually exclusive. • Bits 17 and 20 are mutually exclusive.
4	4 SMF82t01_CCVT_STATUS	1	binary	<p>Cryptographic communication vector table (CCVT) status bits.</p> <p>Bit</p> <p>Meaning When Set</p> <p>0-3 Reserved.</p> <p>4 Compatible with CUSP and PCF.</p> <p>5-7 Reserved.</p>
5	5 SMF82t01_CDX	1	binary	Current crypto domain index.

Table 132. Subtype 1 Initialization/Options Refresh (continued)

Offsets	Name	Length	Format	Description
6	6 SMF82t01_CSFPRM	1	binary	The DDNAME of the member that contains the installation options data set. ICSF internally allocates CSFPARM2 DD based on CSFPARM DD with the member name removed. 0 Release prior to HCR77D0. 1 CSFPARM2 DD (CSFPARM DD). 2 IEFPARM DD (Parmlib concatenation).
7	7	5		Reserved.
12	C SMF82t01_CKDS	44	EBCDIC	Name of the cryptographic key data set (CKDS) that was read into storage.
56	38 SMF82t01_MAX_LENGTH	4	binary	Maximum length for data.
60	3C SMF82t01_USERPARM	8	EBCDIC	USERPARM specifies installation use in the installation options data set.
68	44 SMF82t01_PKDS	44	EBCDIC	PKDS name.
112	70 SMF82t01_TKDS	44	EBCDIC	TKDS name.

Subtype 7 - Operational key load section

Note: The number of hexadecimal digits of the key check value recorded in the field SMF82KV is determined by the value of the MASTERKCVLEN parameter in installation options data set. The SMF82KV field is padded with zeros when the number of digits is less than the length of the field.

Table 133. Subtype 7 operational key entry

Offsets	Name	Length	Format	Description
0	0 SMF82t07_BITS	3	binary	Key part (KPART) bits Bit Meaning When Set 0 Key part verification pattern valid. 1 Reserved. 2 Coprocessor is a CEX2C. (This has been deprecated; see SMF82KAP.) 3 Coprocessor is a CEX3C. (This has been deprecated; see SMF82KAP.) 4 Coprocessor is a CEX4C or higher. (This has been deprecated; see SMF82KAP.) 5 The number of valid nibbles in field SFM82KV is controlled by the MASTERKCVLEN parameter in the options data set. Field SMF82KVL lists the number of valid nibbles recorded. 6-23 Reserved.
3	3 SMF82KVL	1	binary	The number of valid nibbles in field SMF82t07_KPVP.
4	4 SMF82t07_KPVP	8	binary	Key check value of the key. The key check value is left-justified and padded with zeros.
12	C SMF82t07_AP_INDEX	1	binary	Coprocessor number.

Table 133. Subtype 7 operational key entry (continued)

Offsets	Name	Length	Format	Description
13	D SMF82t07_CDX	1	binary	Current crypto domain index.
14	E SMF82t07_AP_TYPE	1	binary	Coprocessor type: X'07' Coprocessor is a CEX2C. X'09' Coprocessor is a CEX3C. X'0A' Coprocessor is a CEX4C. X'0B' Coprocessor is a CEX5C. X'0C' Coprocessor is a CEX6C. X'0D' Coprocessor is a CEX7C. X'0E' Coprocessor is a CEX8C or higher.
15	F	1		Reserved.
16	10 SMF82t07_CKDS	44	EBCDIC	Name of the CKDS containing the key part.
60	3C SMF82t07_KeyLabel	72	EBCDIC	CKDS entry being modified.

Subtype 8 - Cryptographic key data set refresh section

Table 134. Subtype 8 Cryptographic key data set refresh

Offsets	Name	Length	Format	Description
0	0 SMF82t08_OLD_CKDS	44	EBCDIC	Name of the CKDS being replaced.
44	2C SMF82t08_NEW_CKDS	44	EBCDIC	Name of the CKDS to replace the current CKDS.

Subtype 9 - Dynamic CKDS update

Table 135. Subtype 9 Dynamic CKDS update

Offsets	Name	Length	Format	Description
0	0 SMF82t09_BITS	4	binary	Update CKDS bits Bit Meaning When Set 0 CKDS record added. 1 CKDS record changed. 2 CKDS record deleted. 3 CKDS record archived. 4 CKDS record recalled. 5 CKDS record metadata changed. 6 CKDS record was changed by the CKDS KEYS utility. 7-31 Reserved. Note: When bit 6 is off, the changes indicated by the other bits were made by a callable service. When bit 6 is on, the changes were made by the CKDS KEYS utility.
4	4 SMF82t09_CKDS	44	EBCDIC	CKDS name.
48	30 SMF82t09_KeyLabel	72	EBCDIC	CKDS entry being modified.

Subtype 13 - Dynamic PKDS update

Table 136. Subtype 13 Dynamic PKDS update

Offsets	Name	Length	Format	Description
0	0 SMF82t13_BITS	4	binary	Update PKDS bits Bit Meaning When Set 0 PKDS record added. 1 PKDS record changed. 2 PKDS record deleted. 3 PKDS record archived. 4 PKDS record recalled. 5 PKDS record metadata changed. 6 PKDS record was changed by the PKDS KEYS utility. 7-31 Reserved. Note: When bit 6 is off, the changes indicated by the other bits were made by a callable service. When bit 6 is on, the changes were made by the PKDS KEYS utility.
4	4 SMF82t13_PKDS	44	EBCDIC	PKDS name.

Table 136. Subtype 13 Dynamic PKDS update (continued)

Offsets	Name	Length	Format	Description
48 30	SMF82t13_KeyLabel	72	EBCDIC	PKDS entry being modified.

Subtype 14 - Cryptographic coprocessor master key entry

Table 137. Subtype 14 Cryptographic coprocessor master key entry

Offsets	Name	Length	Format	Description
0	0 SMF82t14_NMKbits	4	binary	<p>Flag bytes</p> <p>Bit Meaning When Set</p> <p>0 DES NMK verification pattern is valid.</p> <p>1 RSA NMK verification pattern is valid.</p> <p>2 DES Key key part verification pattern is valid.</p> <p>3 RSA Key Key part verification pattern is valid.</p> <p>4 AES NMK verification pattern is valid.</p> <p>5 AES key part verification pattern is valid.</p> <p>6 ECC NMK verification pattern is valid.</p> <p>7 ECC key part verification pattern is valid.</p> <p>8 Always on.</p> <p>9 Reserved.</p> <p>10 Coprocessor is a CEX2C. (This has been deprecated; see SMF82AAP.)</p> <p>11 Coprocessor is a CEX3C. (This has been deprecated; see SMF82AAP.)</p> <p>12 Coprocessor is a CEX4C or higher. (This has been deprecated; see SMF82AAP.)</p> <p>13-24 Reserved.</p> <p>25 DES NMK entered was 24-bytes long.</p> <p>26-31 Reserved.</p>
4	4 SMF82t14_NMK_VP	16	binary	New master key register verification pattern.
20	14 SMF82t14_KP_VP	16	binary	Key part verification pattern.
36	24 SMF82t14_procnum	1	binary	Cryptographic Processor number.
37	25 SMF82t14_serialnum	8	EBCDIC	Cryptographic Processor serial number.
45	2D SMF82t14_NMK_CDX	1	binary	Cryptographic Coprocessor domain.

Table 137. Subtype 14 Cryptographic coprocessor master key entry (continued)

Offsets	Name	Length	Format	Description
46	2E SMF82t14_NMK_AP_TYPE	1	binary	Coprocessor type: X'07' Coprocessor is a CEX2C. X'09' Coprocessor is a CEX3C. X'0A' Coprocessor is a CEX4C. X'0B' Coprocessor is a CEX5C. X'0C' Coprocessor is a CEX6C. X'0D' Coprocessor is a CEX7C. X'0E' Coprocessor is a CEX8C or higher.
47	2F	1		Reserved.

Subtype 15 - PCI Cryptographic coprocessor retained key create/delete

Table 138. Subtype 15 PCI Cryptographic coprocessor retained key create/delete

Offsets	Name	Length	Format	Description
0	0 SMF82t15_FLAGS	4	binary	First flag byte Bit Meaning When Set 0 Retained key created. 1 Retained key deleted on coprocessor. 2 Retained key deleted from PKDS. 3-7 Reserved. 8 Always on. 9 Reserved. 10 Coprocessor is a CEX2C. (This has been deprecated; see SMF82RAP.) 11 Coprocessor is a CEX3C. (This has been deprecated; see SMF82RAP.) 12 Coprocessor is a CEX4C or higher. (This has been deprecated; see SMF82RAP.) 13-31 Reserved.
4	4 SMF82t15_LABEL	64	EBCDIC	Label of Retained private key.
68	44 SMF82t15_AP	1	binary	Cryptographic Coprocessor number.
69	45 SMF82t15_SERNUM	8	EBCDIC	Cryptographic Coprocessor serial number.
77	4D SMF82t15_DOMAIN	1	binary	Cryptographic Coprocessor domain.

Table 138. Subtype 15 PCI Cryptographic coprocessor retained key create/delete (continued)

Offsets	Name	Length	Format	Description
78	4E SMF82t15_AP_TYPE	1	binary	Coprocessor type: X'07' Coprocessor is a CEX2C. X'09' Coprocessor is a CEX3C. X'0A' Coprocessor is a CEX4C. X'0B' Coprocessor is a CEX5C. X'0C' Coprocessor is a CEX6C. X'0D' Coprocessor is a CEX7C. X'0E' Coprocessor is a CEX8C or higher.
79	4F	1		Reserved.

Subtype 16 - Cryptographic Processor Interface

Table 139. Subtype 16 Cryptographic Processor Interface

Offsets	Name	Length	Format	Description
0	0 SMF82t16_FLAGS	4	binary	Flag bytes Bit Meaning When Set 0 Request command. 1 Reply response. 2-7 Reserved. 8 Always on. 9 Reserved. 10 Coprocessor is a CEX2C. (This has been deprecated; see SMF82t16_AP_TYPE.) 11 Coprocessor is a CEX3C. (This has been deprecated; see SMF82t16_AP_TYPE.) 12 Coprocessor is a CEX4 or higher. (This has been deprecated; see SMF82t16_AP_TYPE.) 13-29 Reserved 30 Coprocessor is configured for CCA. 31 Coprocessor is configured for PKCS #11.
4	4 SMF82t16_proc_num	1	binary	Cryptographic Coprocessor number.
5	5 SMF82t16_ser_num	8	EBCDIC	Cryptographic Coprocessor serial number.
13	D SMF82t16_DOMAIN	1	binary	Cryptographic Coprocessor domain.

Table 139. Subtype 16 Cryptographic Processor Interface (continued)

Offsets	Name	Length	Format	Description
14	E SMF82t16_AP_TYPE	1	binary	Coprocessor type: X'07' Coprocessor is a CEX2C. X'09' Coprocessor is a CEX3C. X'0A' Coprocessor is a CEX4C. X'0B' Coprocessor is a CEX5C. X'0C' Coprocessor is a CEX6C. X'0D' Coprocessor is a CEX7C. X'0E' Coprocessor is a CEX8C or higher. Note: For CEX4 and higher, bits 30 and 31 at offset 0 indicate whether the coprocessor is configured as a CCA or PKCS #11 coprocessor.
15	F	1		Reserved.
16	10 SMF82t16_ParmBlkLen	4	binary	Parameter block length, "xxx".
20	14 SMF82t16_DataBlkLen	4	binary	Parameter data block length, "yyy".
24	18 SMF82t16_ParmBlks			Parameter block of length "xxx" followed by parameter data block of length "yyy".

Fixed length audit data - begins at offset 24 + xxx + yyy.

Table 140. Subtype 16 Cryptographic Processor Interface audit data

Offsets	Name	Length	Format	Description
0	0 SMF82t16_auditData		structure	Fixed length audit data.
0	0 SMF82t16_PAL	4	binary	Length of fixed audit data.
4	4 SMF82t16_PAD	4	binary	PKCS #11 Admin request ID. All zeros if not applicable.
8	8 SMF82t16_PFI	2	binary	Function ID.
10	A SMF82t16_PFR	4	binary	Function Return code 0 Success 4 Not authorized 8 Error
14	E SMF82t16_PDE	256	EBCDIC	Function description.
270	10E SMF82t16_PUS	20	binary	Transaction Sequence Number (TSN) for commands or, for CCA coprocessor requests only, User ID Nonce (random number) for queries. All blanks if not applicable.
290	122 SMF82t16_PTA	8	EBCDIC	Authority for CCA coprocessor requests. Blanks for PKCS #11 coprocessor requests.

Subtype 18 - Cryptographic processor configuration

Table 141. Subtype 18 Cryptographic Processor Configuration				
Offsets	Name	Length	Format	Description
0	0 SMF82t18_FLAGS	4	binary	<p>Flag bytes</p> <p>Bit Meaning When Set</p> <p>0 A Cryptographic processor has been brought online.</p> <p>1 A Cryptographic processor has been taken offline.</p> <p>2 A Cryptographic processor has changed compliance mode.</p> <p>3-7 Reserved.</p> <p>8 Always on.</p> <p>9 Reserved.</p> <p>10 Coprocessor is a CEX2C. (This has been deprecated; see SMF82t18_AP_Type.)</p> <p>11 Coprocessor is a CEX2A. (This has been deprecated; see SMF82t18_AP_Type.)</p> <p>12 Coprocessor is a CEX3C. (This has been deprecated; see SMF82t18_AP_Type.)</p> <p>13 Coprocessor is a CEX3A. (This has been deprecated; see SMF82t18_AP_Type.)</p> <p>14 Coprocessor is a CEX4 or higher. (This has been deprecated; see SMF82t18_AP_Type.)</p> <p>15-28 Reserved.</p> <p>29 Configured as an accelerator</p> <p>30 Configured as a CCA coprocessor</p> <p>31 Configured as a PKCS #11 coprocessor</p>
4	4 SMF82t18_AP_Index	1	binary	Cryptographic Coprocessor number.
5	5 SMF82t18_serialNum	8	EBCDIC	Cryptographic Coprocessor serial number.

Table 141. Subtype 18 Cryptographic Processor Configuration (continued)

Offsets	Name	Length	Format	Description														
13	D SMF82t18_AP_Type	1	binary	<p>Coprocessor type:</p> <p>X'06' Coprocessor is a CEX2A.</p> <p>X'07' Coprocessor is a CEX2C.</p> <p>X'08' Coprocessor is a CEX3A.</p> <p>X'09' Coprocessor is a CEX3C.</p> <p>X'0A' Coprocessor is a CEX4.</p> <p>X'0B' Coprocessor is a CEX5.</p> <p>X'0C' Coprocessor is a CEX6.</p> <p>X'0D' Coprocessor is a CEX7C.</p> <p>X'0E' Coprocessor is a CEX8C or higher.</p> <p>Note: For CEX4 and higher, bits 29, 30, and 31 at offset 0 indicate whether the coprocessor is configured as an accelerator, a CCA, or a PKCS #11 coprocessor.</p>														
14	E	8	binary	<p>System Compliance Information.</p> <p>Byte 0:</p> <table><thead><tr><th>Bit</th><th>Meaning</th></tr></thead><tbody><tr><td>0</td><td>Compliance mode is active.</td></tr><tr><td>1</td><td>Compliance migration mode is active.</td></tr><tr><td>2-7</td><td>Reserved.</td></tr></tbody></table> <p>Bytes 1-6: Reserved.</p> <p>Byte 7:</p> <table><thead><tr><th>Bit</th><th>Meaning</th></tr></thead><tbody><tr><td>0-6</td><td>Reserved.</td></tr><tr><td>7</td><td>PCI-HSM 2016 compliance mode is active.</td></tr></tbody></table> <p>Note: These byte references only relate to this 8-byte structure.</p>	Bit	Meaning	0	Compliance mode is active.	1	Compliance migration mode is active.	2-7	Reserved.	Bit	Meaning	0-6	Reserved.	7	PCI-HSM 2016 compliance mode is active.
Bit	Meaning																	
0	Compliance mode is active.																	
1	Compliance migration mode is active.																	
2-7	Reserved.																	
Bit	Meaning																	
0-6	Reserved.																	
7	PCI-HSM 2016 compliance mode is active.																	

Subtype 20 - Cryptographic processor processing times

Table 142. Subtype 20 Cryptographic Processor Processing Times				
Offsets	Name	Length	Format	Description
0	0 SMF82t20_FLAGS	4	binary	Flag bytes Bit Meaning When Set 0 Reserved. 1 Coprocessor is a CEX2C. (This has been deprecated; see SMF82t20_AP_TYPE.) 2 Coprocessor is a CEX2A. (This has been deprecated; see SMF82t20_AP_TYPE.) 3 Coprocessor is a CEX3C. (This has been deprecated; see SMF82t20_AP_TYPE.) 4 Coprocessor is a CEX3A. (This has been deprecated; see SMF82t20_AP_TYPE.) 5 Coprocessor is a CEX4 or higher. (This has been deprecated; see SMF82t20_AP_TYPE.) 6-28 Reserved. 29 Configured as an accelerator. 30 Configured as a CCA coprocessor. 31 Configured as a PKCS #11 coprocessor.
4	4 SMF82t20_NQAP	8	binary	Coprocessor time before NQAP.
12	C SMF82t20_DQAP	8	binary	Coprocessor time after DQAP.
20	14 SMF82t20_WAIT	8	binary	Coprocessor time after WAIT.
28	1C SMF82t20_QLEN	4	binary	Coprocessor queue length.
32	20 SMF82t20_FUNC	2	EBCDIC	Coprocessor sub function code.
34	22 SMF82t20_APIX	1	binary	Coprocessor index.
35	23 SMF82t20_APID	8	EBCDIC	Coprocessor serial number.
43	2B SMF82t20_DOMN	1	binary	Domain.
44	2C SMF82t20_REFR	1	binary	Reference number.

Table 142. Subtype 20 Cryptographic Processor Processing Times (continued)

Offsets	Name	Length	Format	Description
45	2D SMF82t20_AP_TYPE	1	binary	Coprocessor type: X'06' Coprocessor is a CEX2A. X'07' Coprocessor is a CEX2C. X'08' Coprocessor is a CEX3A. X'09' Coprocessor is a CEX3C. X'0A' Coprocessor is a CEX4. X'0B' Coprocessor is a CEX5. X'0C' Coprocessor is a CEX6. X'0D' Coprocessor is a CEX7C. X'0E' Coprocessor is a CEX8C or higher. Note: For CEX4 and higher, bits 29, 30, and 31 at offset 0 indicate whether the coprocessor is configured as an accelerator, a CCA, or a PKCS #11 coprocessor.
46	2E	2		Reserved.
48	30 SMF82t20_NQAPE	16	binary	AP extended time before NQAP.
64	40 SMF82t20_DQAPE	16	binary	AP extended time after DQAP.
80	50 SMF82t20_WAITE	16	binary	AP extended time after WAIT.

Subtype 21 - ICSF sysplex group change section

Table 143. Subtype 21 ICSF Sysplex Group Change

Offsets	Name	Length	Format	Description
0	0 SMF82t21_GROUP	8	EBCDIC	Name of ICSF Sysplex group.
8	8 SMF82t21_MEMBER	8	EBCDIC	Name of sysplex member.
16	F SMF82t21_ACTION	1	binary	ICSF Sysplex member status flags Bit Meaning When Set 0 Member joined the ICSF sysplex group. 1 Member left the ICSF sysplex group. 2–7 Reserved.

Table 143. Subtype 21 ICSF Sysplex Group Change (continued)

Offsets	Name	Length	Format	Description
17	11 SMF82t21_REASON	1	binary	ICSF Sysplex member conditions of status flags Bit Meaning When Set 0 Member joined or left the ICSF sysplex due to normal initialization/termination processing 1 Member left the ICSF sysplex due to error 2–7 Reserved.
18	12	2		Reserved.
20	14 SMF82t21_TIME	8	EBCDIC	Time of ICSF sysplex join/leave index.
28	1C SMF82t21_KDS	44	EBCDIC	Name of active CKDS.
72	48 SMF82t21_TIMEE	16	binary	Extended time of ICSF sysplex join/leave index.

Subtype 22 - Trusted block create callable services section

Table 144. Subtype 22 Trusted Block Create Callable Services

Offsets	Name	Length	Format	Description
0	0 SMF82t22_FLAGS	4	binary	Process Flag bytes Bit Meaning When Set 0 Created Inactive Trusted Block. 1 Activate an Inactive Block. 2 Trusted Block has Public Key. 3–31 Reserved.
4	4 SMF82t22_CallerASID	2	binary	ASID of caller.
6	6 SMF82t22_LabelIn	64	EBCDIC	Label of Input Trusted Block.
70	46 SMF82t22_LabelOut	64	EBCDIC	Label of Output Trusted Block.
134	86 SMF82t22_LabelTrans	64	EBCDIC	Label of Transport Key.

Subtype 23 - Token data set update

Table 145. Subtype 23 Token Data Set Update

Offsets	Name	Length	Format	Description
0	0 SMF82t23_FLAGS	4	binary	TKDS bits
				Bit Meaning When Set 0 TKDS record added. 1 TKDS record changed. 2 TKDS record deleted. 3 TKDS record archived. 4 TKDS record recalled. 5 TKDS record metadata changed. 6 TKDS record was changed by the PKCS11 Token utility. 7-31 Reserved. Note: When bit 6 is off, the changes indicated by the other bits were made by a callable service. When bit 6 is on, the changes were made by the PKCS11 Token utility.
4	4 SMF82t23_TKDS	44	EBCDIC	TKDS name.
48	30 SMF82t23_Handle	44	EBCDIC	TKDS handle being processed.

Subtype 24 - Duplicate tokens found

Table 146. Subtype 24 Duplicate Tokens Found

Offsets	Name	Length	Format	Description
0	0 SMF82t24_lab_cnt_strt	4	binary	Start of duplicate labels.
4	4 SMF82t24_lab_cnt_end	4	binary	End of duplicate labels.
8	8 SMF82t24_dup_lab_cnt	4	binary	Number of duplicate labels.
16	10 SMF82t24_KDSname	44	binary	Name of key data set.
The following field is repeated <i>count</i> (SMF82t24_dup_lab_cnt) number of times.				
60	3C SMF82t24_labels	72 * SMF82t24_dup_lab _cnt	EBCDIC	Key labels.

Subtype 25 - Key store policy for key token authorization checking

The key store policy must be activated before this SMF record subtype is logged. The subtype is logged when the callable service request fails the Key Token Authorization Checking key store policy check.

Table 147. Subtype 25 Key Store Policy Key Token Authorization Checking

Offsets	Name	Length	Format	Description
0	0 SMF82t25_KDS	44	EBCDIC	Data set name.

Table 147. Subtype 25 Key Store Policy Key Token Authorization Checking (continued)				
Offsets	Name	Length	Format	Description
44	2C SMF82t25_FLAGS	4	binary	Key store policy flags: Bit Meaning When Set 0 Warning. 1 List is incomplete. 2 List is from CKDS. 3 List is from PKDS. 4 Authorization failures. 5 Archived failures. 6 Proactive failures. 7 Deactivated failures. 8-31 Reserved.
48	30 SMF82t25_label_cnt	4	binary	Number of key labels following.
The following field is repeated <i>count</i> (SMF82t25_label_cnt) number of times.				
52	34 SMF82t25_labels	72	EBCDIC	Unauthorized duplicate key label and key type.

Subtype 26 - Public key data set refresh

Table 148. Subtype 26 Public Key Data Set Refresh				
Offsets	Name	Length	Format	Description
0	0 SMF82t26_flags	4	binary	Flags: Bit Meaning When Set 0 Data space was refreshed. 1-31 Reserved.
4	4 SMF82t26_OLD_PKDS	44	EBCDIC	Old PKDS Name.
48	30 SMF82t26_NEW_PKDS	44	EBCDIC	New PKDS Name.

Subtype 27 - PKA key management extensions

Table 149. Subtype 27 PKA Key Management Extensions

Offsets	Name	Length	Format	Description
0	0 SMF82t27_flags	4	binary	<p>PKA Key Management Extension flags:</p> <p>Bit</p> <p>Meaning When Set</p> <p>0 PKA token may not be used for requested function.</p> <p>1 SYM token may not be exported by the provided PKA token.</p> <p>2 PKA label list is incomplete.</p> <p>3 SYM label list is incomplete.</p> <p>4 Input is an X.509 certificate.</p> <p>24 Trusted certificate repository has changed.</p> <p>25 PKA Key Management Extensions in WARNONLY mode.</p> <p>26 An error was detected during processing.</p> <p>27 Trusted cert repository was empty.</p> <p>28 An error was detected while extracting APPLDATA.</p> <p>29 The repository was not found.</p> <p>30 One or more certs could not be parsed.</p> <p>Bits 0-4 are set during callable services.</p> <p>Bits 24-30 are set during repository parsing.</p> <p>Bits 5-23 and 31 are reserved.</p>
4	4 SMF82t27_function	8	EBCDIC	Name of the service that issued this SMF record. The name is in the form CSFzzz.
12	C SMF82t27_APPLDATAlen	1	binary	Length of the enablement profile APPLDATA or current repository name.
13	D SMF82t27_APPLDATA	247	EBCDIC	Enablement profile APPLDATA or current repository name.
260	104 SMF82t27_FUNCSPEC	0	binary	Function-specific section of the record.
260	104 SMF82t27_APPLDATA_PARSE	0	binary	APPLDATA parsing results section.
260	104 SMF82t27_SAF_RC	2	binary	SAF_RC or 'FFFF'X.
262	106 SMF82t27_SERV_RC	2	binary	RACF RC or ICSF RC.
264	108 SMF82t27_SERV_RS	4	binary	RACF RS or ICSF RS.
260	104 SMF82t27_SERVICE_SECT	0	binary	Callable services section.
260	104 SMF82t27_PKA_rec_cnt	4	binary	Number of PKA labels present in this record. When the input key to the service is an X.509 certificate, there are no PKA labels present.
264	108 SMF82t27_SYM_rec_cnt	4	binary	Number of SYM labels present in this record.

Table 149. Subtype 27 PKA Key Management Extensions (continued)					
Offsets	Name	Length	Format	Description	
The following is repeated SMF82t27_PKA_rec_cnt number of times.					
268	10C SMF82t27_PKAlabels	64	EBCDIC	PKA key label.	
The following is repeated SMF82t27_SYM_rec_cnt number of times.					
268+ zzz	10C+ zzz SMF82t27_PKAlabels	72	EBCDIC	SYM key label.	

Subtype 28 - High performance encrypted key

The record logs all requests to rewrap a secure cryptographic key under the CPACF master key for use with the CPACF encryption instructions. Symmetric keys can be passed as a record in the CKDS or key token. Asymmetric keys can be passed as a record in the PKDS or a key token.

Table 150. Subtype 28 High Performance Encrypted Key					
Offsets	Name	Length	Format	Description	
0	0 SMF82t28_flags	4	binary	<div>High Performance Encrypted Key flags:</div> <div>Symmetric keys:</div> <div>Bit</div> <div>Meaning When Set</div> <div>0</div> <div>Rewrapping operation is not permitted for this symmetric key.</div> <div>1</div> <div>Rewrapping operation was permitted for this symmetric key.</div> <div>2</div> <div>The list of labels is incomplete.</div> <div>3</div> <div>The key identifier was supplied as a key token, not as a label in the CKDS.</div> <div>Asymmetric keys:</div> <div>Bit</div> <div>Meaning When Set</div> <div>4</div> <div>Rewrapping operation was performed. The key identifier was an asymmetric key supplied as a label in the PKDS.</div> <div>5</div> <div>Rewrapping operation was performed. The key identifier was an asymmetric key token, not a label in the PKDS.</div> <div>Bits 6 - 31 are reserved.</div>	
4	4 SMF82t28_function	8	EBCDIC	Name of the service that issued this SMF record. The name is in the form of CSFzzzz.	
12	C SMF82t28_labelCnt	4	binary	Number of labels present in this record.	
The following is repeated SMF82t28_labelCnt number of times.					
Note: When bit 1 of the SMF82t28_flags field at offset 24 is on, this label is a CKDS record label that was SAF checked for authorization. When bit 4 of the SMF82t28_flags field is on, this label is the PKDS record label of the key token that was rewrapped.					
16	10 SMF82t28_labels	72 or 64	EBCDIC	SYM key label and type or ASYM label.	

Subtype 29 - TKE workstation audit record

Table 151. Subtype 29 TKE Workstation Audit Record

Offsets	Name	Length	Format	Description
0	0 SMF82t29_FLAGS	4	binary	Flags -- Reserved.
4	4 SMF82t29_namelen	2	binary	TKE workstation name length (<i>nlen</i>).
6	6 SMF82t29_reclen	2	binary	TKE audit record data length (<i>nlen</i>).
8	8 SMF82t29_name	<i>nlen</i>	EBCDIC	TKE workstation name.
8 + <i>nlen</i>	8 + <i>nlen</i>		<i>nlen</i>	TKE audit record data.

Subtype 30 - Key store policy archived and inactive KDS records

Table 152. Subtype 30 Archived and inactive KDS records

Offsets	Name	Length	Format	Description
0	0 SMF82t30_flags	4	binary	Flag bytes Notes: <ul style="list-style-type: none">• Bits 0 to 2 are mutually exclusive.• Bits 8-11 are mutually exclusive. Bit Meaning When Set 0 CKDS 1 PKDS 2 TKDS 3-7 Reserved. 8 Record that is archived was referenced by service. By policy, service call failed. 9 Record that is archived was referenced by service. By policy, service call succeeded 10 Record that is pre-active was referenced by service. Service call failed. 11 Record that is inactive was referenced by service. Service call failed. 12 Record that is archived was referenced by a service that performs data encryption. By policy, the service call failed. 13-30 Reserved 31 Archived Key for Data Decryption Use control is enabled.
4	4 SMF82t30_dsname	44	EBCDIC	Key data set name.
48	30 SMF82t30_label	72	EBCDIC	Key data set entry.

Notes:

1. Bit 8 of the SMF82t30_flags field indicated that the request to use an archived key failed. The failure is due to (in order of precedence):

- The service request was for a service that does data encryption:
 - The Archived Key for Data Decryption Use control is enabled; or
 - The Archived Key for Data Decryption Use control is disabled and the Key Archive Use control is disabled.
 - The service request was for a service that does data decryption and both Archived Key for Data Decryption Use and the Key Archive Use controls are disabled.
 - The Key Archive Use control is disabled.
2. Bit 12 of the SMF82t30_flags field indicated that the request to use an archived key failed. This is the case when the Archived Key for Data Decryption Use control is enabled and the service request was for a service that does data encryption. The Key Archive Use control is not considered when bit 12 is set.

Subtype 31 - Cryptographic usage statistics

ICSF supports a cryptographic usage statistics section containing a header and a variable number of triplets.

Note: A single SMF record cannot exceed 32K bytes.

Table 153. Subtype 31 Cryptographic usage statistics				
Offsets (Dec)	Name	Length	Format	Description
0	SMF82t31_ver	1	binary	Version number.
1	SMF82t31_domain	1	binary	ICSF domain index.
2	SMF82t31_hlen	2	binary	Length of this header.
4	SMF82t31_trip1_off	2	binary	Offset from SMF82t31 into triplet section.
6	SMF82t31_trip1_len	2	binary	Length of triplet section.
8	SMF82t31_start	16	binary	Start time (TOD clock) of the SMF interval in STCKE format.
24	SMF82t31_end	16	binary	End time (TOD clock) of the SMF interval in STCKE format.
40	SMF82t31_userid_as	8	EBCDIC	The HOME address space user ID.
48	SMF82t31_userid_tk	8	EBCDIC	The task level user ID (if present).
56	SMF82t31_jobid	8	EBCDIC	The job ID for the HOME address space.
64	SMF82t31_jobname	8	EBCDIC	The job name for the HOME address space.
72	SMF82t31_jobname2	8	EBCDIC	The job name of the SECONDARY address space (ICSF caller).
80	SMF82t31_plexname	8	EBCDIC	The sysplex member name.
88	SMF82t31_userid_as2	8	EBCDIC	The SECONDARY address space user ID.
96	SMF82t31_jobid2	8	EBCDIC	The job ID for the SECONDARY address space.

Table 153. Subtype 31 Cryptographic usage statistics (continued)				
Offsets (Dec)	Name	Length	Format	Description
104	SMF82t31_jobstod	16	binary	The HOME address space job selection time (TOD) clock in STCKE format.
120	SMF82t31_jobstod2	16	binary	The SECONDARY address space job selection time (TOD) clock in STCKE format.
136	SMF82t31_jobedate	4	binary	Job entry date (JMREDATE) that the reader recognized the JOB statement for the HOME address space in the form 0CYDDDF. This field can be zero. See 'Standard and extended SMF record headers' in z/OS MVS System Management Facilities (SMF) for a detailed description of the formatted headers.
140	SMF82t31_jobetime	4	binary	Job entry time (JMRENTY) in hundredths of a second that the reader recognized the JOB statement for the HOME address space. This field can be zero. See 'Standard and extended SMF record headers' in z/OS MVS System Management Facilities (SMF) for a detailed description of the formatted headers.

Each Tag-Length-Value (TLV) triplet is a structure called SMF82_TRIPL. The values for the tags, the format, and the maximum length of the data are defined in [Table 154 on page 436](#).

Table 154. Subtype 31 SMF82_TRIPL				
Offsets (Dec)	Name	Length	Format	Description
0	SMF82_triplet_tag	2	binary	Tag of the data.
2	SMF82_triplet_tlen	2	binary	Length of the tag, length, and data fields.
4	SMF82_triplet_val	*	varies	Value of the data.

The tag values and their corresponding information are described in [Table 155 on page 437](#).

Table 155. Subtype 31 tag values				
Tag ID (2 bytes)	Tag name	Length (2 bytes)	Format	Description
Cryptographic engine (ENG) usage statistics				
X'0201'	SMF82t31val_ENG_CARD	20	structure	Crypto card usage count. <ul style="list-style-type: none"> • 4-byte EBCDIC identifier (for example, 5C01). • 8-byte EBCDIC serial number. • 4-byte binary card usage count.
X'0203'	SMF82t31val_ENG_CPACF	8	binary	CPACF usage count.
X'0204'	SMF82t31val_ENG_SFTW	8	binary	Crypto software usage count.
Cryptographic service (SRV) usage statistics				
X'0205'	SMF82t31val_SRV	16	structure	ICSF callable service usage count. <ul style="list-style-type: none"> • 8-byte EBCDIC service name. • 4-byte binary service usage count. See Appendix G, “Resource names for CCA and ICSF entry points,” on page 503 for service names.
X'0206'	SMF82STAT_SRVUDX	16	structure	UDX service usage count. <ul style="list-style-type: none"> • 8-byte EBCDIC UDX service name. • 4-byte binary UDX service usage count.
Cryptographic algorithm (ALG) usage statistics				
X'0207'	SMF82t31val_ALG	16	structure	Crypto algorithm usage count. <ul style="list-style-type: none"> • 8-byte EBCDIC algorithm name. • 4-byte binary algorithm usage count. See Table 156 on page 437 for algorithm names.

Table 156. SMF82t31val_ALG algorithm names	
Algorithm name	Description
AES	AES algorithm. Key length cannot be determined.
AES128	128-bit AES algorithm.
AES192	192-bit AES algorithm.
AES256	256-bit AES algorithm.
BCRYPT	BCRYPT hashing algorithm.
Blowfish	Blowfish algorithm (PKCS #11 only).
ChaCha20	ChaCha20 algorithm (PKCS #11 only).
DES	DES algorithm. Key length cannot be determined.

<i>Table 156. SMF82t31val_ALG algorithm names (continued)</i>	
Algorithm name	Description
DES112	112-bit DES algorithm.
DES168	168-bit DES algorithm.
DES56	56-bit DES algorithm.
DH	DH algorithm (PKCS #11 only).
DSA	DSA algorithm (PKCS #11 only).
ECCBP160	160-bit ECC algorithm.
ECCBP192	192-bit ECC algorithm.
ECCBP224	224-bit ECC algorithm.
ECCBP256	256-bit ECC algorithm.
ECCBP320	320-bit ECC algorithm.
ECCBP384	384-bit ECC algorithm.
ECCBP512	512-bit ECC algorithm.
ECCKB256	256-bit ECC algorithm.
ECCP192	192-bit ECC algorithm.
ECCP224	224-bit ECC algorithm.
ECCP256	256-bit ECC algorithm.
ECCP384	384-bit ECC algorithm.
ECCP521	521-bit ECC algorithm.
ED25519	ECC curve25519 signature algorithm.
ED448	ECC curve448 signature algorithm.
GenSec	Generic Secret algorithm (PKCS #11 only).
HMAC	HMAC algorithm (CCA only).
KY768R2	CRYSTALS-Kyber 768 Round 2 Algorithm
KY768R3	CRYSTALS-Kyber 768 Round 3 Algorithm
KY1024R2	CRYSTALS-Kyber 1024 Round 2 Algorithm.
KY1024R3	CRYSTALS-Kyber 1024 Round 3 Algorithm.
LI2	CRYSTALS-Dilithium (6,5) Round 2 algorithm.
LI2-65R3	CRYSTALS-Dilithium (6,5) Round 3 algorithm.
LI2-87R2	CRYSTALS-Dilithium (8,7) Round 2 algorithm.
LI2-87R3	CRYSTALS-Dilithium (8,7) Round 3 algorithm.
MD2	MD2 hashing algorithm (PKCS #11 only).
MD5	MD5 hashing algorithm.
MLDH0404	Pre-hash ML-DSA (4,4)
MLDH0605	Pre-hash ML-DSA (6,5)

Table 156. SMF82t31val_ALG algorithm names (continued)	
Algorithm name	Description
MLDH0807	Pre-hash ML-DSA (8,7)
MLDP0404	Pure ML-DSA (4,4)
MLDP0605	Pure ML-DSA (6,5)
MLDP0807	Pure ML-DSA (8,7)
MLKE0768	ML-KEM 768
MLKE1024	ML-KEM 1024
PRNG	Pseudo-random number generator
PRNGFIPS	Pseudo-random number generator. Consistent with NIST SP800-90A.
RC4	RC4 algorithm (PKCS #11 only).
RPMD160	RPMD-160 hashing algorithm.
RSA512	RSA algorithm with a key bit length from 512 to 1023 bits.
RSA1024	RSA algorithm with a key bit length from 1024 to 2047 bits.
RSA2048	RSA algorithm with a key bit length from 2048 to 4095 bits.
RSA4096	RSA algorithm with a key bit length of 4096 to 8191 bits.
RSA8192	RSA algorithm with a key bit length of 8192 bits or greater.
SHA1	SHA-1 hashing algorithm.
SHA224	SHA-224 hashing algorithm.
SHA256	SHA-256 hashing algorithm.
SHA3-224	SHA3-224 hashing algorithm
SHA3-256	SHA3-256 hashing algorithm
SHA3-384	SHA3-384 hashing algorithm
SHA3-512	SHA3-512 hashing algorithm
SHA384	SHA-384 hashing algorithm.
SHA512	SHA-512 hashing algorithm.
SHAKE128	SHAKE128 hashing algorithm
SHAKE256	SHAKE256 hashing algorithm
TLS-PRF	TLS Pseudo-Random Function derivation protocol (PKCS #11 Only).
X25519	ECC curve25519 key exchange algorithm.
X448	ECC curve448 key exchange algorithm.

See [Appendix G, “Resource names for CCA and ICSF entry points,”](#) on page 503 for a list of possible values for the SMF82t31val_SRV tag.

Subtype 40 - CCA symmetric key lifecycle event

The main section for this subtype consists of a number of Tag-Length-Value (TLV) triplets. The following triplets may be contained in the record. The specific set of triplets is dependent on the type of event and the information that is available. This replaces subtype 9.

Table 157. Subtype 40 CCA symmetric key lifecycle event

Tag value		Name	Length	Format	Description
Dec	Hex				
256	100	SMF82_TAG_KEY_EVENT	1	binary	Key event. This field always occurs first in the record. X'10' Key record added to KDS. X'11' Key record updated in KDS. X'12' Key record deleted from KDS. X'13' Key record archived. X'14' Key record restored. X'15' Key record metadata changed. X'17' Key record pre-activated. X'18' Key record activated. X'19' Key record deactivated. X'1B' Key exported. X'20' Key generated. X'21' Key imported. Note: 1. When a key is exported, the key token that gets audited is the input or source token. 2. When a key is imported, the key token that gets audited is the output or target token.
257	101	SMF82_TAG_KDS_LABEL	72	EBCDIC	The label in the KDS.
258	102	SMF82_TAG_KDS_DSNAME	44	EBCDIC	The dataset name of the KDS associated with the event. If there is no associated KDS (for example, the event only involves a token), this field is not present.
259	103	SMF82_TAG_KEY_NAME	64	EBCDIC	The key name from the token. Applies to variable-length CCA tokens and TR-31 key blocks.

Table 157. Subtype 40 CCA symmetric key lifecycle event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
261	105	SMF82_TAG_KEY_FPRINT	1 - 64	binary	<p>One or more key fingerprints.</p> <p>The first byte is the number (<i>n</i>) of fingerprints present for the key. Following that are <i>n</i> type-length-value triplets. Within each of these triplets is a 1-byte fingerprint type, followed by a 1-byte length for the triplet, followed by the fingerprint.</p> <p>Fingerprint types:</p> <p>X'01' Ciphertext obtained from encrypting a data block filled with binary zeros in ECB mode.</p> <p>X'03' SHA-256 algorithm. See Appendix E in <i>z/OS Cryptographic Services ICSF Application Programmer's Guide</i> for more information.</p> <p>X'04' SHAVP1 algorithm. See Appendix E in <i>z/OS Cryptographic Services ICSF Application Programmer's Guide</i> for more information.</p> <p>X'05' CMACZERO (only present for compliant-tagged AES keys). See the Key Test2 (CSNBKYT2/CSNEKYT2) callable service in <i>z/OS Cryptographic Services ICSF Application Programmer's Guide</i> for more information.</p> <p>For example, X'010105010203' indicates that there is one fingerprint value (01) which is the ciphertext obtained from using the key to encrypt a data block of binary zeros in ECB mode (01). The fingerprint is 3 bytes in length (05 – 2) and the value is X'010203'.</p> <p>Note: If the event pertains to a record in the CKDS, the key fingerprint is only present when using a KDSR-format dataset.</p>
262	106	SMF82_TAG_SERVICE	8	EBCDIC	The service associated with the event.
264	108	SMF82_TAG_TOK_FMT	1	binary	<p>The format of the token.</p> <p>X'01' Fixed length CCA token.</p> <p>X'02' Variable length CCA token.</p> <p>X'03' TR-31 key block.</p> <p>X'04' RKX token.</p> <p>Note: When format is RKX token, no other key or token related fields are present.</p>
265	109	SMF82_TAG_KEY_SEC	1	binary	<p>Key security.</p> <p>X'01' No key present.</p> <p>X'02' Clear key.</p> <p>X'03' Key encrypted under master key.</p> <p>X'04' Key encrypted under key encrypting key.</p>

Table 157. Subtype 40 CCA symmetric key lifecycle event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
266	10A	SMF82_TAG_KEY_ALG	1	binary	Key algorithm. X'02' DES. X'03' AES. X'04' HMAC.
267	10B	SMF82_TAG_KEY_TYPE	2	binary	Key type. The key type from the token. Applies to variable-length CCA tokens only. See “Variable-length symmetric key token” in z/OS Cryptographic Services ICSF Application Programmer's Guide for the list of key types.
268	10C	SMF82_TAG_KEY_CV	8	binary	Key control vector. The first eight bytes of the control vector from the token. Applies to fixed-length DES CCA tokens only. See Appendix C in z/OS Cryptographic Services ICSF Application Programmer's Guide for information on how to interpret the control vector.
269	10D	SMF82_TAG_KEY_USAGE_CKDS	3 - 33	binary	Key usage fields. Consists of a 1 byte count followed by one or more 2-byte key usage fields. Applies to variable-length CCA tokens only. See Appendix B in z/OS Cryptographic Services ICSF Application Programmer's Guide for the list of key usage values for variable length tokens.
270	10E	SMF82_TAG_KEY_LEN	2	binary	The length of the key (in bits). Not present for HMAC tokens.
271	10F	SMF82_TAG_KEY_CP	16	EBCDIC	Key crypto period. The start date followed by the end date for the record in YYYYMMDD format (therefore, YYYYMMDDYYYYMMDD). If a date is not set, that portion of the key crypto period will be blanks. Applies to records in the KDS which have at least one date set.
280	118	SMF82_TAG_KEY_TIV	variable	binary	A key token identification value. A string of bytes present in the key token. Can be used to help uniquely identify a key token. Notes: 1. Only present when TOK_FMT is a fixed-length CCA token. 2. This is the 4-byte token validation value. For more information, see Appendix B. Key Token Formats (Sections “AES Key Token Formats” and “DES Key Token Formats”) in z/OS Cryptographic Services ICSF Application Programmer's Guide .
281	119	SMF82_TAG_KEY_COMP_TAG	0	N/A	The key is compliant-tagged. Note: May only be present when KEY_ALG is DES and TOK_FMT is a fixed-length CCA token or when KEY_ALG is AES and TOK_FMT is a variable-length CCA token, or when TOK_FMT is TR-31 key block. For DES key tokens, only when the key derivation function (KDF) is X'03' or greater is it considered compliant-tagged.
283	11B	SMF82_TAG_TR31_KB_VER_ID	1	EBCDIC	The TR-31 key block version ID. Only applies to TR-31 key blocks. See “X9.143 (TR-31) key block header and optional block data” on page 320 for possible TR-31 header values.
284	11C	SMF82_TAG_TR31_KEY_USAGE	2	EBCDIC	The TR-31 key block key usage. Only applies to TR-31 key blocks. See “X9.143 (TR-31) key block header and optional block data” on page 320 for possible TR-31 header values.

Table 157. Subtype 40 CCA symmetric key lifecycle event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
285	11D	SMF82_TAG_TR31_MODE_USE	1	EBCDIC	The TR-31 key block mode of use. Only applies to TR-31 key blocks. See “X9.143 (TR-31) key block header and optional block data” on page 320 for possible TR-31 header values.
286	11E	SMF82_TAG_TR31_KEY_VER	2	EBCDIC	The TR-31 key block key version number. Only applies to TR-31 key blocks. See “X9.143 (TR-31) key block header and optional block data” on page 320 for possible TR-31 header values.
287	11F	SMF82_TAG_TR31_EXPORT	1	EBCDIC	The TR-31 key block exportability. Only applies to TR-31 key blocks. See “X9.143 (TR-31) key block header and optional block data” on page 320 for possible TR-31 header values.

Subtype 41 - CCA asymmetric key lifecycle event

The main section for this subtype consists of a number of Tag-Length-Value (TLV) triplets. The following triplets may be contained in the record. The specific set of triplets is dependent on the type of event and the information that is available. This replaces subtype 13.

Table 158. Subtype 41 CCA asymmetric key lifecycle event

Tag value		Name	Length	Format	Description
Dec	Hex				
256	100	SMF82_TAG_KEY_EVENT	1	Binary	Key event. This field always occurs first in the record. X'10' Key token added to KDS. X'11' Key token updated in KDS. X'12' Key token deleted from KDS. X'13' Key token archived. X'14' Key token restored. X'15' Key token metadata changed. X'17' Key token pre-activated. X'18' Key token activated. X'19' Key token deactivated. X'1B' Key token exported. X'20' Key token generated. X'21' Key token imported. Note: 1. When a key is exported, the key token that gets audited is the input or source token. 2. When a key is imported, the key token that gets audited is the output or target token.

Table 158. Subtype 41 CCA asymmetric key lifecycle event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
257	101	SMF82_TAG_KDS_LABEL	72	EBCDIC	The 64-byte KDS label left-justified and padded on the right with blanks.
258	102	SMF82_TAG_KDS_DSNAME	44	EBCDIC	The dataset name of the KDS associated with the event. If there is no associated KDS (for example, the event only involves a token), this field is not present.
259	103	SMF82_TAG_KEY_NAME	64	EBCDIC	The key name from the token.
260	104	SMF82_TAG_OBJ_TYPE	1	Binary	<p>Object type.</p> <p>X'02' Public key.</p> <p>X'05' Certificate.</p> <p>X'0B' Public/Private key pair.</p> <p>X'0D' Trusted block.</p> <p>Notes:</p> <ul style="list-style-type: none"> When the object type is trusted block, no other key or token related information is present. When the object type is certificate, the only other key related information present is the key fingerprint (KEY_FPRINT).
261	105	SMF82_TAG_KEY_FPRINT	1 - 64	Binary	<p>One or more key fingerprints.</p> <p>The first byte is the number (<i>n</i>) of fingerprints present for the key. Following that are <i>n</i> type-length-value triplets. Within each of these triplets is a 1-byte fingerprint type, followed by a 1-byte length for the triplet, followed by the fingerprint.</p> <p>Fingerprint types:</p> <p>X'02' SHA-1 hash of the public key.</p> <p>For example, X'010205010203' indicates that there is one fingerprint value (01) which is the SHA-1 hash of the public key (02). The fingerprint is 3 bytes in length (05 – 2) and the value is X'010203'.</p> <p>Note: If the event pertains to a record in the PKDS, the key fingerprint is only present when using a KDSR-format dataset.</p>
262	106	SMF82_TAG_SERVICE	8	EBCDIC	The service associated with the event.
265	109	SMF82_TAG_KEY_SEC	1	Binary	<p>Key security.</p> <p>X'01' No key present.</p> <p>X'02' Clear key.</p> <p>X'03' Key encrypted under master key.</p> <p>X'04' Key encrypted under key encrypting key.</p>

Table 158. Subtype 41 CCA asymmetric key lifecycle event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
266	10A	SMF82_TAG_KEY_ALG	1	Binary	<p>Key algorithm.</p> <p>X'07' RSA.</p> <p>X'08' DSA.</p> <p>X'09' ECC.</p> <p>X'0E' ML-DSA, CRYSTALS-Dilithium.</p> <p>X'0F' ML-KEM, CRYSTALS-Kyber.</p> <p>Note: When the algorithm is DSA, the only other key or token information that is present is the object type.</p>
270	10E	SMF82_TAG_KEY_LEN	2	Binary	<p>The length of the key (in bits). For RSA, this is the length of the modulus. For ECC, this is the length of the private value. For CRYSTALS-Dilithium (6,5) Round 2, this is the size of t1 in bits.</p> <p>Note: This tag will not be present for ML-DSA, ML-KEM, CRYSTALS-Dilithium (8,7) Round 2 or 3, CRYSTALS-Dilithium (6,5) Round 3, and CRYSTALS-Kyber.</p>
271	10F	SMF82_TAG_KEY_CP	16	EBCDIC	<p>Key crypto period. The start date followed by the end date for the record in YYYYMMDD format (therefore, YYYYMMDDYYYYMMDD). If a date is not set, that portion of the key crypto period will be blanks. Applies to records in the KDS which have at least one date set.</p>
272	110	SMF82_TAG_KEY_USAGE_PKDS	4	Binary	<p>Key usage for private keys.</p> <p>Bit</p> <p>Meaning when set</p> <p>0 Undefined.</p> <p>1 Key management usage permitted.</p> <p>2 Signature usage permitted.</p> <p>3 Key translation permitted.</p> <p>4 Key agreement usage permitted.</p> <p>5 Nonrepudiation usage permitted.</p> <p>6 Key encipherment usage permitted.</p> <p>7 Data encipherment usage permitted.</p> <p>8 Key certificate sign usage permitted.</p> <p>9 Certificate Revocation List sign usage permitted.</p> <p>10 Only key encipher usage permitted during key agreement.</p> <p>11 Only key decipher usage permitted during key agreement.</p> <p>12-31 Reserved.</p>

Table 158. Subtype 41 CCA asymmetric key lifecycle event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
274	112	SMF82_TAG_KEY_EC_CURVE	1	Binary	ECC curve type. X'01' Prime curve. X'02' Brainpool curve. X'04' Edwards curve. X'05' Koblitz curve.
281	119	SMF82_TAG_KEY_COMP_TAG	0	N/A	The key is compliant-tagged.
282	11A	SMF82_TAG_QSA_ALG_PARAM	2	Binary	QSA Algorithm Parameter. Only applicable to ML-DSA, ML-KEM, CRYSTALS-Dilithium and CRYSTALS-Kyber. X'0404' ML-DSA (4,4). X'0605' ML-DSA, CRYSTALS-Dilithium (6,5). X'0807' ML-DSA, CRYSTALS-Dilithium (8,7). X'0768' ML-KEM, CRYSTALS-Kyber (768). X'1024' ML-KEM, CRYSTALS-Kyber (1024).
288	120	SMF82_TAG_QSA_ALG_ID	1	Binary	QSA Algorithm Identifier. Only applicable to ML-DSA, ML-KEM, CRYSTALS-Dilithium and CRYSTALS-Kyber. X'01' CRYSTALS-Dilithium Round 2. X'02' CRYSTALS-Kyber Round 2. X'03' CRYSTALS-Dilithium Round 3. X'04' CRYSTALS-Kyber Round 3. X'05' Pure ML-DSA. X'06' ML-KEM. X'07' Pre-hash ML-DSA.

Subtype 42 - PKCS#11 object lifecycle event

The main section for this subtype consists of a number of Tag-Length-Value (TLV) triplets. The following triplets may be contained in the record. The specific set of triplets is dependent on the type of event and the information that is available. This replaces subtype 23.

Table 159. Subtype 42 PKCS#11 object lifecycle event

Tag value		Name	Length	Format	Description
Dec	Hex				
256	100	SMF82_TAG_KEY_EVENT	1	Binary	Object event. This field always occurs first in the record. X'10' Object added to KDS. X'11' Object updated in KDS. X'12' Object deleted from KDS. X'13' Object archived. X'14' Object restored. X'15' Object metadata changed. X'17' Object pre-activated. X'18' Object activated. X'19' Object deactivated. X'1B' Object wrapped by another key.
257	101	SMF82_TAG_KDS_LABEL	72	EBCDIC	The 44-byte key handle left-justified and padded on the right with blanks. If the sequence number of the handle is 'FFFFFFF', this was a raw object.
258	102	SMF82_TAG_KDS_DSNAME	44	EBCDIC	The dataset name of the KDS.
259	103	SMF82_TAG_KEY_NAME	1 - 513	EBCDIC	The CKA_LABEL attribute from the object. If the CKA_Label is greater than 512 characters, the plus (+) symbol is placed at the 513th character to indicate truncation.
260	104	SMF82_TAG_OBJ_TYPE	1	Binary	Object type. X'01' Symmetric key. X'02' Public key. X'03' Private key. X'05' Certificate. X'06' Domain parameters. X'07' Data object. X'0C' PKCS #11 token.

Table 159. Subtype 42 PKCS#11 object lifecycle event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
261	105	SMF82_TAG_KEY_FPRINT	1 - 64	Binary	<p>One or more key fingerprints.</p> <p>The first byte is the number (<i>n</i>) of fingerprints present for the key. Following that are <i>n</i> type-length-value triplets. Within each of these triplets is a 1-byte fingerprint type, followed by a 1-byte length for the triplet, followed by the fingerprint.</p> <p>Fingerprint types:</p> <p>X'01' Ciphertext obtained from encrypting a data block filled with binary zeros in ECB mode.</p> <p>X'02' SHA-1 hash of the public key.</p> <p>For example, X'010105010203' indicates that there is one fingerprint value (01) which is the ciphertext obtained from using the key to encrypt 8 bytes of binary zeros in ECB mode (01). The fingerprint is 3 bytes in length (05 – 2) and the value is X'010203'.</p> <p>Note: If the event pertains to a record in the TKDS, the key fingerprint is only present when using a KDSR-format dataset.</p>
262	106	SMF82_TAG_SERVICE	8	EBCDIC	The service associated with the event.
265	109	SMF82_TAG_KEY_SEC	1	Binary	<p>Key security.</p> <p>X'02' Clear key.</p> <p>X'03' Key encrypted under master key.</p>
266	10A	SMF82_TAG_KEY_ALG	1	Binary	<p>Key algorithm.</p> <p>X'01' Generic symmetric.</p> <p>X'02' DES.</p> <p>X'03' AES.</p> <p>X'05' RC4.</p> <p>X'06' Blowfish.</p> <p>X'07' RSA.</p> <p>X'08' DSA.</p> <p>X'09' ECC.</p> <p>X'0A' Diffie-Hellman.</p> <p>X'0D' ChaCha20.</p> <p>X'0E' CRYSTALS-Dilithium.</p> <p>X'0F' CRYSTALS-Kyber.</p>
270	10E	SMF82_TAG_KEY_LEN	2	Binary	The length of the public key (in bits). For RSA, this is the length of the modulus. For CRYSTALS-Dilithium, this is the size of t1 in bits. For CRYSTALS-Kyber, this is the size of the public polynomial vector in bits.

Table 159. Subtype 42 PKCS#11 object lifecycle event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
271	10F	SMF82_TAG_KEY_CP	16	EBCDIC	Key crypto period. The start date followed by the end date for the record in YYYYMMDD format (therefore, YYYYMMDDYYYYMMDD). If a date is not set, that portion of the key crypto period will be blanks. Applies to records in the KDS which have at least one date set.
273	111	SMF82_TAG_KEY_USAGE_TKDS	4	Binary	<p>Key usage for private key, public key, and secret key objects.</p> <p>Bit</p> <p>Meaning when set</p> <p>0 Data encryption allowed.</p> <p>1 Data decryption allowed.</p> <p>2 Key derivation allowed.</p> <p>3 Sign allowed where signature is appendix.</p> <p>4 Verify allowed where signature is appendix.</p> <p>5 Sign allowed where data is recovered from signature.</p> <p>6 Verify allowed where data is recovered from signature.</p> <p>7 Key wrapping allowed.</p> <p>8 Key unwrapping allowed.</p> <p>9 FIPS 140-2 key attribute is ON. When honored, key usage must be FIPS 140-2 compliant.</p> <p>10-31 Reserved.</p>
274	112	SMF82_TAG_KEY_EC_CURVE	1	Binary	<p>ECC curve type.</p> <p>X'01' Prime curve.</p> <p>X'02' Brainpool curve.</p> <p>X'04' Edwards curve.</p>

Table 159. Subtype 42 PKCS#11 object lifecycle event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
279	117	SMF82_TAG_FIPS_INFO	4	Binary	<p>FIPS information related to the event. This is written only when the FIPSMODE is set to a 140-2 mode and FIPS approval checking was enforced for the event.</p> <p>Also see TLV SMF82_TAG_FIPS_INFO_V2, which has enhanced FIPS reporting for all FIPS modes.</p> <p>Bit</p> <p>Meaning when set</p> <p>0-1</p> <p>B'10' FIPSMODE(YES).</p> <p>B'01' FIPSMODE(COMPAT).</p> <p>B'00' FIPSMODE(NO).</p> <p>2 FIPS 140-2 algorithm approval evaluated due to ICSF FIPSMODE option. FIPSMODE(YES) or FIPSMODE(COMPAT) and the request was not exempt from FIPS checking.</p> <p>3 FIPS 140-2 algorithm approval evaluated due to key attribute or service call. The key object had the FIPS compliance flag on or FIPS compliance was requested by a service call parameter.</p> <p>4 Algorithm usage approved for FIPS 140-2.</p> <p>5 FIPS information TLV SMF82_TAG_FIPS_INFO_V2 (535/x217) is also present, which contains the information in this TLV and additional FIPS information.</p> <p>6-31 Reserved.</p>
282	11A	SMF82_TAG_QSA_ALG_PARAM	2	Binary	<p>QSA Algorithm Parameter.</p> <p>Only applicable to CRYSTALS-Dilithium and CRYSTALS-Kyber.</p> <p>X'0605' CRYSTALS-Dilithium (6,5).</p> <p>X'0807' CRYSTALS-Dilithium (8,7).</p> <p>X'0768' CRYSTALS-Kyber (768).</p> <p>X'1024' CRYSTALS-Kyber (1024).</p>
288	120	SMF82_TAG_QSA_ALG_ID	1	Binary	<p>QSA Algorithm Identifier.</p> <p>Only applicable to CRYSTALS-Dilithium and CRYSTALS-Kyber.</p> <p>X'01' CRYSTALS-Dilithium Round 2.</p> <p>X'02' CRYSTALS-Kyber Round 2.</p> <p>X'03' CRYSTALS-Dilithium Round 3.</p>

Table 159. Subtype 42 PKCS#11 object lifecycle event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
535	217	SMF82_TAG_FIPS_INFO_V2	SMF82_ TRIPL_L EN	Binary	<p>Enhanced FIPS information related to the event. This TLV is written for all FIPS modes except the case of FIPSMODE(NO) and the key FIPS attribute OFF. For examples, see 'Using SMF reporting for FIPS planning' in z/OS Cryptographic Services ICSF Writing PKCS #11 Applications.</p> <p>Bit(s) Meaning when set</p> <p>0-1 Defined when bits 16-23 indicate 140-2 mode.</p> <p>B'10' FIPSMODE(YES).</p> <p>B'01' FIPSMODE(COMPAT).</p> <p>B'00' FIPSMODE(NO).</p> <p>2 FIPS 140 algorithm approval evaluated due to ICSF FIPSMODE option. FIPSMODE(YES) or FIPSMODE(COMPAT) is in effect and the request was not exempt from FIPS compliance or FIPSMODE(140-3,*) is in effect.</p> <p>3 FIPS 140-2 algorithm approval evaluated due to key attribute or service call. The key object involved had the FIPS 140-2 attribute on or FIPS compliance was requested via a parameter on the service call.</p> <p>4 Algorithm approved for level of FIPS 140 reported in bits 16-23. OFF if the algorithm is not approved for the FIPSMODE in effect, for example:</p> <ul style="list-style-type: none"> FIPSEXEMPT or USEFIPS140-2 profile read access applied to the request. FIPSMODE is FIPS 140-3 INDICATE, but only FIPS 140-2 was met. <p>5 FIPSEXEMPT READ access - Request exempt from all FIPS 140-2 checking.</p> <p>6 USEFIPS1402 READ access - Algorithm checking done using 140-2 FIPSMODE(COMPAT) rules.</p> <p>7 Not used.</p> <p>8 FIPSMODE(mode,ENFORCE), where mode is reported in bits 16-23. Currently, mode is always 140-3.</p> <p>9 FIPSMODE (mode,INDICATE), where mode is reported in bits 16-23. Currently, mode is always 140-3.</p> <p>10 FIPSMODE(mode,HYRBRID), where mode is reported in bits 16-23. Currently, mode is always 140-3.</p> <p>11-15 Not used.</p> <p>16-23 The level of FIPS algorithm approval required by the FIPSMODE for clear keys.</p> <p>X'00' 140-2.</p> <p>X'01' 140-3.</p>

Table 159. Subtype 42 PKCS#11 object lifecycle event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
535 (cont)	217	SMF82_TAG_FIPS_INFO_V2	SMF82_ TRIPL_L EN	Binary	Bit(s) Meaning when set 24-31 FIPSMODE which could be met – The maximum level of FIPS 140 that the usage is approved for clear keys. X'00' 140-2. X'01' 140-3. 32-63 Reserved.

Note: Not written for secure keys.

For examples of the SMF82_TAG_FIPS_INFO_V2 tag-length-value (TLV), see 'Using SMF reporting for FIPS planning' in [z/OS Cryptographic Services ICSF Writing PKCS #11 Applications](#).

Subtype 44 - CCA symmetric key usage event

The main section for this subtype consists of a number of Tag-Length-Value (TLV) triplets. The following triplets may be contained in the record. The specific set of triplets is dependent on the type of event and the information that is available.

Table 160. Subtype 44 CCA symmetric key usage event

Tag value		Name	Length	Format	Description
Dec	Hex				
257	101	SMF82_TAG_KDS_LABEL	72	EBCDIC	The label in the KDS.
259	103	SMF82_TAG_KEY_NAME	64	EBCDIC	The key name from the token. Applies to variable-length CCA tokens and TR-31 key blocks.

Table 160. Subtype 44 CCA symmetric key usage event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
261	105	SMF82_TAG_KEY_FPRINT	1 - 64	binary	<p>One or more key fingerprints.</p> <p>The first byte is the number (<i>n</i>) of fingerprints present for the key. Following that are <i>n</i> type-length-value triplets. Within each of these triplets is a 1-byte fingerprint type, followed by a 1-byte length for the triplet, followed by the fingerprint.</p> <p>Fingerprint types:</p> <p>X'01' Ciphertext obtained from encrypting a data block filled with binary zeros in ECB mode.</p> <p>X'03' SHA-256 algorithm. See Appendix E in <i>z/OS Cryptographic Services ICSF Application Programmer's Guide</i> for more information.</p> <p>X'04' SHAVP1 algorithm. See Appendix E in <i>z/OS Cryptographic Services ICSF Application Programmer's Guide</i> for more information.</p> <p>X'05' CMACZERO (only present for compliant-tagged AES keys). See the Key Test2 (CSNBKYT2/CSNEKYT2) callable service in <i>z/OS Cryptographic Services ICSF Application Programmer's Guide</i> for more information.</p> <p>For example, X'010105010203' indicates that there is one fingerprint value (01) which is the ciphertext obtained from using the key to encrypt a data block of binary zeros in ECB mode (01). The fingerprint is 3 bytes in length (05 – 2) and the value is X'010203'.</p>
262	106	SMF82_TAG_SERVICE	8	EBCDIC	The service associated with the event.
264	108	SMF82_TAG_TOK_FMT	1	binary	<p>The format of the token.</p> <p>X'01' Fixed length CCA token.</p> <p>X'02' Variable length CCA token.</p> <p>X'03' TR-31 key block.</p> <p>X'04' RKX token.</p> <p>Note: When format is RKX token, no other key or token related fields are present.</p>
265	109	SMF82_TAG_KEY_SEC	1	binary	<p>Key security.</p> <p>X'01' No key present.</p> <p>X'02' Clear key.</p> <p>X'03' Key encrypted under master key.</p> <p>X'04' Key encrypted under key encrypting key.</p>

Table 160. Subtype 44 CCA symmetric key usage event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
266	10A	SMF82_TAG_KEY_ALG	1	binary	Key algorithm. X'02' DES. X'03' AES. X'04' HMAC.
267	10B	SMF82_TAG_KEY_TYPE	2	binary	Key type. The key type from the token. Applies to variable-length CCA tokens only. See “Variable-length symmetric key token” in z/OS Cryptographic Services ICSF Application Programmer's Guide for the list of key types.
268	10C	SMF82_TAG_KEY_CV	8	binary	Key control vector. The first eight bytes of the control vector from the token. Applies to fixed-length DES CCA tokens only. See Appendix C in z/OS Cryptographic Services ICSF Application Programmer's Guide for information on how to interpret the control vector.
269	10D	SMF82_TAG_KEY_USAGE_CKDS	3 - 33	binary	Key usage fields. Consists of a 1 byte count followed by one or more 2-byte key usage fields. Applies to variable-length CCA tokens only. See Appendix B in z/OS Cryptographic Services ICSF Application Programmer's Guide for the list of key usage values for variable length tokens.
270	10E	SMF82_TAG_KEY_LEN	2	binary	The length of the key (in bits). Not present for HMAC tokens.
275	113	SMF82_TAG_START_TOD	16	binary	Start time of the interval in STCKE format.
276	114	SMF82_TAG_END_TOD	16	binary	End time of the interval in STCKE format.
277	115	SMF82_TAG_USG_COUNT	4	binary	Number of usages accounted for in this record.
278	116	SMF82_TAG_KEY_OLD	0	binary	The key is internal, but not wrapped under the current master key. Additionally, if key store policy is enabled for CKDS, the key is wrapped under the old master key. Applies to token usage only.
280	118	SMF82_TAG_KEY_TIV	variable	binary	A key token identification value. A string of bytes present in the key token. Can be used to help uniquely identify a key token. Notes: 1. Only present when TOK_FMT is a fixed-length CCA token. 2. This is the 4-byte token validation value. For more information, see Appendix B. Key Token Formats (Sections “AES Key Token Formats” and “DES Key Token Formats”) in z/OS Cryptographic Services ICSF Application Programmer's Guide .
281	119	SMF82_TAG_KEY_COMP_TAG	0	N/A	The key is compliant-tagged. Note: May only be present when KEY_ALG is DES and TOK_FMT is a fixed-length CCA token, when KEY_ALG is AES and TOK_FMT is a variable-length CCA token, or when TOK_FMT is TR-31 key block. For DES key tokens, only when the key derivation function (KDF) is X'03' or greater is it considered compliant-tagged.
283	11B	SMF82_TAG_TR31_KB_VER_ID	1	EBCDIC	The TR-31 key block version ID. Only applies to TR-31 key blocks. See “X9.143 (TR-31) key block header and optional block data” on page 320 for possible TR-31 header values.

Table 160. Subtype 44 CCA symmetric key usage event (continued)					
Tag value		Name	Length	Format	Description
Dec	Hex				
284	11C	SMF82_TAG_TR31_KEY_USAGE	2	EBCDIC	The TR-31 key block key usage. Only applies to TR-31 key blocks. See “X9.143 (TR-31) key block header and optional block data” on page 320 for possible TR-31 header values.
285	11D	SMF82_TAG_TR31_MODE_USE	1	EBCDIC	The TR-31 key block mode of use. Only applies to TR-31 key blocks. See “X9.143 (TR-31) key block header and optional block data” on page 320 for possible TR-31 header values.
286	11E	SMF82_TAG_TR31_KEY_VER	2	EBCDIC	The TR-31 key block key version number. Only applies to TR-31 key blocks. See “X9.143 (TR-31) key block header and optional block data” on page 320 for possible TR-31 header values.
287	11F	SMF82_TAG_TR31_EXPORT	1	EBCDIC	The TR-31 key block exportability. Only applies to TR-31 key blocks. See “X9.143 (TR-31) key block header and optional block data” on page 320 for possible TR-31 header values.

Subtype 45 - CCA asymmetric key usage event

The main section for this subtype consists of a number of Tag-Length-Value (TLV) triplets. The following triplets may be contained in the record. The specific set of triplets is dependent on the type of event and the information that is available.

Table 161. Subtype 45 CCA asymmetric key usage event					
Tag value		Name	Length	Format	Description
Dec	Hex				
257	101	SMF82_TAG_KDS_LABEL	72	EBCDIC	The 64-byte KDS label left-justified and padded on the right with blanks.
259	103	SMF82_TAG_KEY_NAME	64	EBCDIC	The key name from the token.
260	104	SMF82_TAG_OBJ_TYPE	1	binary	Object type. X'02' Public key. X'05' Certificate. X'0B' Public/Private key pair. X'0D' Trusted block. Notes: <ul style="list-style-type: none"> When the object type is trusted block, no other key or token related information is present. When the object type is certificate, the only other key related information present is the key fingerprint (KEY_FPRINT).

Table 161. Subtype 45 CCA asymmetric key usage event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
261	105	SMF82_TAG_KEY_FPRINT	1 - 64	binary	<p>One or more key fingerprints.</p> <p>The first byte is the number (<i>n</i>) of fingerprints present for the key. Following that are <i>n</i> type-length-value triplets. Within each of these triplets is a 1-byte fingerprint type, followed by a 1-byte length for the triplet, followed by the fingerprint.</p> <p>Fingerprint types:</p> <p>X'02' SHA-1 hash of the public key.</p> <p>For example, X'010105010203' indicates that there is one fingerprint value (01) which is the ciphertext obtained from using the key to encrypt 8 bytes of binary zeros in ECB mode (01). The fingerprint is 3 bytes in length (05 – 2) and the value is X'010203'.</p>
262	106	SMF82_TAG_SERVICE	8	EBCDIC	The service associated with the event.
265	109	SMF82_TAG_KEY_SEC	1	binary	<p>Key security.</p> <p>X'01' No key present.</p> <p>X'02' Clear key.</p> <p>X'03' Key encrypted under master key.</p> <p>X'04' Key encrypted under key encrypting key.</p>
266	10A	SMF82_TAG_KEY_ALG	1	binary	<p>Key algorithm.</p> <p>X'07' RSA.</p> <p>X'08' DSA.</p> <p>X'09' ECC.</p> <p>X'0E' ML-DSA, CRYSTALS-Dilithium.</p> <p>X'0F' ML-KEM, CRYSTALS-Kyber.</p> <p>Note: When the algorithm is DSA, the only other key or token information present is the object type.</p>
270	10E	SMF82_TAG_KEY_LEN	2	binary	<p>The length of the public key (in bits). For CRYSTALS-Dilithium (6,5) Round 2, this is the size of t1 in bits.</p> <p>Note: This tag will not be present for ML-DSA, ML-KEM, CRYSTALS-Dilithium (8,7) Round 2 or 3, CRYSTALS-Dilithium (6,5) Round 3, and CRYSTALS-Kyber.</p>

Table 161. Subtype 45 CCA asymmetric key usage event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
272	110	SMF82_TAG_KEY_USAGE_PKDS	4	binary	<p>Key usage for private keys.</p> <p>Bit</p> <p>Meaning when set</p> <p>0 Undefined.</p> <p>1 Key management usage permitted.</p> <p>2 Signature usage permitted.</p> <p>3 Key translation permitted.</p> <p>4 Key agreement usage permitted.</p> <p>5 Nonrepudiation usage permitted.</p> <p>6 Key encipherment usage permitted.</p> <p>7 Data encipherment usage permitted.</p> <p>8 Key certificate sign usage permitted.</p> <p>9 Certificate Revocation List sign usage permitted.</p> <p>10 Only key encipher usage permitted during key agreement.</p> <p>11 Only key decipher usage permitted during key agreement.</p> <p>12-31 Reserved.</p>
274	112	SMF82_TAG_KEY_EC_CURVE	1	binary	<p>ECC curve type.</p> <p>X'01' Prime curve.</p> <p>X'02' Brainpool curve.</p> <p>X'04' Edwards curve.</p> <p>X'05' Koblitz curve.</p>
275	113	SMF82_TAG_START_TOD	16	binary	Start time of the interval in STCKE format.
276	114	SMF82_TAG_END_TOD	16	binary	End time of the interval in STCKE format.
277	115	SMF82_TAG_USG_COUNT	4	binary	Number of usages accounted for in this record.
278	116	SMF82_TAG_KEY_OLD	0	N/A	The key is internal, but not wrapped under the current master key. Applies to token usage only.
281	119	SMF82_TAG_KEY_COMP_TAG	0	N/A	The key is compliant-tagged.

Table 161. Subtype 45 CCA asymmetric key usage event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
282	11A	SMF82_TAG_QSA_ALG_PARAM	2	Binary	QSA Algorithm Parameter. Only applicable to ML-DSA, ML-KEM, CRYSTALS-Dilithium and CRYSTALS-Kyber. X'0404' ML-DSA (4,4). X'0605' ML-DSA, CRYSTALS-Dilithium (6,5). X'0807' ML-DSA, CRYSTALS-Dilithium (8,7). X'0768' ML-KEM, CRYSTALS-Kyber (768). X'1024' ML-KEM, CRYSTALS-Kyber (1024).
288	120	SMF82_TAG_QSA_ALG_ID	1	Binary	QSA Algorithm Identifier. Only applicable to ML-DSA, ML-KEM, CRYSTALS-Dilithium and CRYSTALS-Kyber. X'01' CRYSTALS-Dilithium Round 2. X'02' CRYSTALS-Kyber Round 2. X'03' CRYSTALS-Dilithium Round 3. X'04' CRYSTALS-Kyber Round 3. X'05' Pure ML-DSA. X'06' ML-KEM. X'07' Pre-hash ML-DSA.

Subtype 46 - PKCS#11 key usage event

The main section for this subtype consists of a number of Tag-Length-Value (TLV) triplets. The following triplets may be contained in the record. The specific set of triplets is dependent on the type of event and the information that is available. Created when the ICSF option AUDITPKCS11USG is specified.

Table 162. Subtype 46 PKCS#11 key usage event

Tag value		Name	Length	Format	Description
Dec	Hex				
257	101	SMF82_TAG_KDS_LABEL	72	EBCDIC	The 44-byte key handle left-justified and padded on the right with blanks. If the sequence number of the handle is 'FFFFFFF', this was a raw object.
259	103	SMF82_TAG_KEY_NAME	1 - 513	EBCDIC	The CKA_LABEL attribute from the object. If the CKA_Label is greater than 512 characters, the plus (+) symbol is placed at the 513th character to indicate truncation.

Table 162. Subtype 46 PKCS#11 key usage event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
260	104	SMF82_TAG_OBJ_TYPE	1	binary	Object type. X'01' Symmetric key. X'02' Public key. X'03' Private key. X'05' Certificate. X'06' Domain parameters. X'07' Data object. X'0C' PKCS #11 token.
261	105	SMF82_TAG_KEY_FPRINT	1 - 64	binary	One or more key fingerprints. The first byte is the number (<i>n</i>) of fingerprints present for the key. Following that are <i>n</i> type-length-value triplets. Within each of these triplets is a 1-byte fingerprint type, followed by a 1-byte length for the triplet, followed by the fingerprint. Fingerprint types: X'01' Ciphertext obtained from encrypting a data block filled with binary zeros in ECB mode. X'02' SHA-1 hash of the public key. For example, X'010105010203' indicates that there is one fingerprint value (01) which is the ciphertext obtained from using the key to encrypt 8 bytes of binary zeros in ECB mode (01). The fingerprint is 3 bytes in length (05 – 2) and the value is X'010203'.
262	106	SMF82_TAG_SERVICE	8	EBCDIC	The service associated with the event.
265	109	SMF82_TAG_KEY_SEC	1	binary	Key security. X'02' Clear key. X'03' Key encrypted under master key.

Table 162. Subtype 46 PKCS#11 key usage event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
266	10A	SMF82_TAG_KEY_ALG	1	binary	Key algorithm. X'01' Generic symmetric. X'02' DES. X'03' AES. X'05' RC4. X'06' Blowfish. X'07' RSA. X'08' DSA. X'09' ECC. X'0A' Diffie-Hellman. X'0D' ChaCha20. X'0E' CRYSTALS-Dilithium. X'0F' CRYSTALS-Kyber.
270	10E	SMF82_TAG_KEY_LEN	2	binary	The length of the public key (in bits). For RSA, this is the length of the modulus. For CRYSTALS-Dilithium, this is the size of t1 in bits. For CRYSTALS-Kyber, this is the size of the public polynomial vector in bits.

Table 162. Subtype 46 PKCS#11 key usage event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
273	111	SMF82_TAG_KEY_USAGE_TKDS	4	binary	<p>Key usage.</p> <p>Bit</p> <p>Meaning when set</p> <p>0 Data encryption allowed.</p> <p>1 Data decryption allowed.</p> <p>2 Key derivation allowed.</p> <p>3 Sign allowed where signature is appendix.</p> <p>4 Verify allowed where signature is appendix.</p> <p>5 Sign allowed where data is recovered from signature.</p> <p>6 Verify allowed where data is recovered from signature.</p> <p>7 Key wrapping allowed.</p> <p>8 Key unwrapping allowed.</p> <p>9 FIPS 140-2 key attribute is ON. When honored, key usage must be FIPS 140-2 compliant.</p> <p>10-31 Reserved.</p>
274	112	SMF82_TAG_KEY_EC_CURVE	1	binary	<p>ECC curve type.</p> <p>X'01' Prime curve.</p> <p>X'02' Brainpool curve.</p> <p>X'04' Edwards curve.</p>
275	113	SMF82_TAG_START_TOD	16	binary	Start time of the interval in STCKE format.
276	114	SMF82_TAG_END_TOD	16	binary	End time of the interval in STCKE format.
277	115	SMF82_TAG_USG_COUNT	4	binary	Number of usages accounted for in this record.

Table 162. Subtype 46 PKCS#11 key usage event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
279	117	SMF82_TAG_FIPS_INFO	4	binary	<p>FIPS information related to the event. This is written only when the FIPSMODE is set to a 140-2 mode and FIPS approval checking was enforced for the event.</p> <p>Also see TLV SMF82_TAG_FIPS_INFO_V2, which has enhanced FIPS reporting for all FIPS modes.</p> <p>Bit</p> <p>Meaning when set</p> <p>0-1</p> <p>B'10' FIPSMODE(YES).</p> <p>B'01' FIPSMODE(COMPAT).</p> <p>B'00' FIPSMODE(NO).</p> <p>2 FIPS 140-2 algorithm approval evaluated due to ICSF FIPSMODE option. FIPSMODE(YES) or FIPSMODE(COMPAT) and the request was not exempt from FIPS checking.</p> <p>3 FIPS 140-2 algorithm approval evaluated due to key attribute or service call. The key object had the FIPS compliance flag on or FIPS compliance was requested by a service call parameter.</p> <p>4 Algorithm usage approved for FIPS 140-2.</p> <p>5 FIPS information TLV SMF82_TAG_FIPS_INFO_V2 is also present, which contains the information in this TLV and additional FIPS information.</p> <p>6-31 Reserved.</p>
282	11A	SMF82_TAG_QSA_ALG_PARAM	2	Binary	<p>QSA Algorithm Parameter.</p> <p>Only applicable to CRYSTALS-Dilithium and CRYSTALS-Kyber.</p> <p>X'0605' CRYSTALS-Dilithium (6,5).</p> <p>X'0807' CRYSTALS-Dilithium (8,7).</p> <p>X'0768' CRYSTALS-Kyber (768).</p> <p>X'1024' CRYSTALS-Kyber (1024).</p>
288	120	SMF82_TAG_QSA_ALG_ID	1	Binary	<p>QSA Algorithm Identifier.</p> <p>Only applicable to CRYSTALS-Dilithium and CRYSTALS-Kyber.</p> <p>X'01' CRYSTALS-Dilithium Round 2.</p> <p>X'02' CRYSTALS-Kyber Round 2.</p> <p>X'03' CRYSTALS-Dilithium Round 3.</p>

Table 162. Subtype 46 PKCS#11 key usage event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
535	217	SMF82_TAG_FIPS_INFO_V2	SMF82_ TRIPL_L EN	Binary	<p>Enhanced FIPS information related to the event. This TLV is written for all FIPS modes except the case of FIPSMODE(NO) and the key FIPS attribute OFF. For examples, see 'Using SMF reporting for FIPS planning' in z/OS Cryptographic Services ICSF Writing PKCS #11 Applications.</p> <p>Bit(s) Meaning when set</p> <p>0-1 Defined when bits 16-23 indicate 140-2 mode.</p> <p>B'10' FIPSMODE(YES).</p> <p>B'01' FIPSMODE(COMPAT).</p> <p>B'00' FIPSMODE(NO).</p> <p>2 FIPS 140 algorithm approval evaluated due to ICSF FIPSMODE option. FIPSMODE(YES) or FIPSMODE(COMPAT) is in effect and the request was not exempt from FIPS compliance or FIPSMODE(140-3,*) is in effect.</p> <p>3 FIPS 140-2 algorithm approval evaluated due to key attribute or service call. The key object involved had the FIPS 140-2 attribute on or FIPS compliance was requested via a parameter on the service call.</p> <p>4 Algorithm approved for level of FIPS 140 reported in bits 16-23. OFF if the algorithm is not approved for the FIPSMODE in effect, for example:</p> <ul style="list-style-type: none"> FIPSEXEMPT or USEFIPS140-2 profile read access applied to the request. FIPSMODE is FIPS 140-3 INDICATE, but only FIPS 140-2 was met. <p>5 FIPSEXEMPT READ access - Request exempt from all FIPS 140-2 checking.</p> <p>6 USEFIPS1402 READ access - Algorithm checking done using 140-2 FIPSMODE(COMPAT) rules.</p> <p>7 Not used.</p> <p>8 FIPSMODE(mode,ENFORCE), where mode is reported in bits 16-23. Currently, mode is always 140-3.</p> <p>9 FIPSMODE (mode,INDICATE), where mode is reported in bits 16-23. Currently, mode is always 140-3.</p> <p>10 FIPSMODE(mode,HYRBRID), where mode is reported in bits 16-23. Currently, mode is always 140-3.</p> <p>11-15 Not used.</p> <p>16-23 The level of FIPS algorithm approval required by the FIPSMODE for clear keys.</p> <p>X'00' 140-2.</p> <p>X'01' 140-3.</p>

Table 162. Subtype 46 PKCS#11 key usage event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
535 (continued)	217	SMF82_TAG_FIPS_INFO_V2	SMF82_T TRIPL_L EN	Binary	Bit(s) Meaning when set 24-31 FIPSMODE which could be met – The maximum level of FIPS 140 that the usage is approved for clear keys. X'00' 140-2. X'01' 140-3. 32-63 Reserved.

Subtype 47 - PKCS#11 no key usage event

The main section for this subtype consists of a number of Tag-Length-Value (TLV) triplets. The following triplets may be contained in the record. The specific set of triplets is dependent on the type of event and the information that is available.

Table 163. Subtype 47 PKCS#11 no key usage event

Tag value		Name	Length	Format	Description
Dec	Hex				
262	106	SMF82_TAG_SERVICE	8	EBCDIC	The service associated with the event. Service is either CSF1PPRF or CSF1POWH.
275	113	SMF82_TAG_START_TOD	16	binary	Start time of the interval in STCKE format.
276	114	SMF82_TAG_END_TOD	16	binary	End time of the interval in STCKE format.
277	115	SMF82_TAG_USG_COUNT	4	binary	Number of usages accounted for in this record.
279	117	SMF82_TAG_FIPS_INFO	4	binary	FIPS 140-2 information related to the event. Bit Meaning when set 0-1 B'10' FIPSMODE(YES). B'01' FIPSMODE(COMPAT). B'00' FIPSMODE(NO). 2 FIPS 140-2 algorithm approval evaluated due to ICSF FIPSMODE option. FIPSMODE(YES) or FIPSMODE(COMPAT) and the request was not exempt. 3 FIPS 140-2 algorithm approval evaluated due to key attribute or service call. The key object had the FIPS compliance flag on or FIPS compliance was requested by a service call parameter. 4 Algorithm usage approved for FIPS 140-2. See note #1 below. 5 FIPS information TLV SMF82_TAG_FIPS_INFO_V2 is also present, which contains the information in this TLV and additional FIPS information. 6-31 Reserved.

Table 163. Subtype 47 PKCS#11 no key usage event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
535	217	SMF82_TAG_FIPS_INFO_V2	SMF82_ TRIPL_L EN	Binary	Enhanced FIPS information related to the event.
<p>Bit(s) Meaning when set</p> <p>0-1</p> <p>B'10' FIPSMODE(YES).</p> <p>B'01' FIPSMODE(COMPAT).</p> <p>B'00' FIPSMODE(NO).</p> <p>2 FIPS 140 algorithm approval evaluated due to ICSF FIPSMODE option. Always off for the no key usage subtype.</p> <p>3 FIPS 140 algorithm approval evaluated due to key attribute or service call. Always on - compliance was requested via a parameter on the service call.</p> <p>4 Algorithm usage approved for level FIPS 140 reported in bits 16-23. See note #1 below.</p> <p>5 Not applicable to this subtype.</p> <p>6 Not applicable to this subtype.</p> <p>7 Not used.</p> <p>8 FIPSMODE(mode,ENFORCE), where mode is reported in bits 16-23. Currently, mode is always 140-3.</p> <p>9 FIPSMODE (mode,INDICATE), where mode is reported in bits 16-23. Currently, mode is always 140-3.</p> <p>10 FIPSMODE(mode,HYBRID), where mode is reported in bits 16-23. Currently, mode is always 140-3.</p> <p>11-15 Not used.</p> <p>16-23 FIPSMODE level – The level of FIPS algorithm approval required by the FIPSMODE.</p> <p>X'00' 140-2.</p> <p>X'01' 140-3.</p> <p>24-31 Not applicable to this subtype.</p> <p>32-63 Reserved.</p>					

Note:

1. If the service being used is CSF1PPRF with RULE PRNGFIPS, when bit 4 is on, then random number generation is consistent with NIST SP800-90A.

Subtype 48 - Compliance warning event

The main section for this subtype consists of a number of Tag-Length-Value (TLV) triplets. The following triplets may be contained in the record. The specific set of triplets depends on the type of event and the information that is available.

Table 164. Subtype 48 Compliance warning event					
Dec	Hex	Name	Length	Format	Description
520	208	SMF82_TAG_COMP_RESULT	2	binary	<p>The compliance result of the operation. This tag is always present and always occurs first.</p> <p>When the value is hexadecimal zeros (X'0000'), the compliance of the request could not be evaluated. This is referred to as <i>compliance not evaluated</i> elsewhere in the documentation. The service either does not currently support compliant-tagged key tokens or there are key tokens being used in the service that cannot currently become compliant-tagged.</p> <p>The service may support compliant-tagged key tokens in the future or the key tokens that cannot currently become compliant may be able to become compliant-tagged in the future.</p>
522	20A	SMF82_TAG_COMP_LVL	1	binary	<p>The compliance level that the operation was checked against.</p> <p>X'01' PCI-HSM 2016.</p>
262	106	SMF82_TAG_SERVICE	8	EBCDIC	The service that was invoked.
523	20B	SMF82_TAG_COMP_TOK	variable	binary	A collection of triplets that contains information about one key that is used in the request. There is one COMP_TOK triplet for each key token that is used in the request.
The follow triplets make up the value portion of the COMP_TOK TLV triplet.					

Table 164. Subtype 48 Compliance warning event (continued)					
Dec	Hex	Name	Length	Format	Description
521	209	SMF82_TAG_COMP_CHK	2	binary	<p>The results of the compliance check performed against the key token.</p> <p>Bit</p> <p>Meaning when set</p> <p>0 Compliant key.</p> <p>1 Key was not evaluated for compliance. The key does not affect the compliance of the operation.</p> <p>2 Weak key or key wrapped by a weaker key. When this is an internal key, it is a weak key. When this is an external key, it is either a weak key or wrapped by a weaker key. When running with the September 2020 or later licensed internal code, either bit 7 or bit 8 will also be on to indicate definitively a weak key or a key wrapped by a weaker key.</p> <p>3 Key type or usage attributes is not compliant.</p> <p>4 Evaluation error.</p> <p>5 NOCV KEK.</p> <p>6 Compliance unknown. If this is an X.509 certificate, this means you are running without a CEX7C which has the June 2020 or later licensed internal code (LIC). Upgrading the coprocessors will yield an actual compliance result. An external key token may be reported as unknown when used with non-compliant tokens. Once the non-compliant tokens are made compliant, future records will report a known value for the external key token.</p> <p>7 Key wrapped by a weaker key. This bit gets set when running with the September 2020 or later licensed internal code. Otherwise, only bit 2 is set for a key wrapped by a weaker key. When this bit is set, bit 2 is also set.</p> <p>8 Weak key. This bit gets set when running with the September 2020 or later licensed internal code. Otherwise, only bit 2 is set for a weak key. When this bit is set, bit 2 is also set.</p> <p>9-15 Reserved.</p>
524	20C	SMF82_TAG_KDS_TYPE	1	binary	<p>The KDS type corresponding to the key token (for example, the KDS where this key might be stored). This has no bearing on whether the key token is in the KDS.</p> <p>X'01' CKDS.</p> <p>X'02' PKDS.</p>
257	101	SMF82_TAG_KDS_LABEL	64 or 72	EBCDIC	<p>The KDS label left justified and padded on the right with blanks. When KDS_TYPE is CKDS, the length is 72. When KDS_TYPE is PKDS, the length is 64.</p> <p>Note: Present only when a key label is passed to the service.</p>
259	103	SMF82_TAG_KEY_NAME	64	EBCDIC	<p>The key name from the token.</p> <p>Note: Applies to CCA variable-length tokens and TR-31 key blocks.</p>

Table 164. Subtype 48 Compliance warning event (continued)

Dec	Hex	Name	Length	Format	Description
260	104	SMF82_TAG_OBJ_TYPE	1	binary	<p>Object type.</p> <p>X'02' Public key.</p> <p>X'05' Certificate.</p> <p>X'0B' Public/Private key pair.</p> <p>X'0D' Trusted block.</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. Present only when KDS_TYPE is PKDS. 2. When object type is trusted block, no other key or token-related information is present. 3. When the object type is certificate, the only other key related information present is the key fingerprint (KEY_FPRINT).
261	105	SMF82_TAG_KEY_FPRINT	1 – 64	binary	<p>One or more key fingerprints.</p> <p>The first byte is the number (n) of fingerprints present for the key. Following that are n type-length-value triplets. Within each of these triplets is a 1-byte fingerprint type, followed by a 1-byte length for the triplet, followed by the fingerprint.</p> <p>Fingerprint types:</p> <p>X'01' Ciphertext that is obtained from encrypting a data block filled with binary zeros in ECB mode.</p> <p>X'02' SHA-1 hash of the public key¹.</p> <p>X'03' SHA-256 algorithm. For more information, see Appendix E, Cryptographic algorithms and processes, in z/OS Cryptographic Services ICSF Application Programmer's Guide.</p> <p>X'04' SHA1P1 algorithm. For more information, see Appendix E, Cryptographic algorithms and processes, in z/OS Cryptographic Services ICSF Application Programmer's Guide.</p> <p>X'05' CMACZERO (only present for compliant-tagged AES keys). See the Key Test2 (CSNBKYT2/CSNEKYT2) callable service in z/OS Cryptographic Services ICSF Application Programmer's Guide for more information.</p> <p>For example, X'010105010203' indicates that there is one fingerprint value (01) which is the ciphertext that is obtained from using the key to encrypt a data block of binary zeros in ECB mode (01). The fingerprint is 3 bytes in length (05 – 2) and the value is X'010203'.</p> <p>¹ The public key is converted to an ASN.1 DER-encoded subjectPublicKey BIT STRING as specified in RFC 3279. The key fingerprint is the hash of the subjectPublicKey (excluding the tag, length, and number of unused bits).</p>

Table 164. Subtype 48 Compliance warning event (continued)

Dec	Hex	Name	Length	Format	Description
264	108	SMF82_TAG_TOK_FMT	1	binary	<p>The format of the token.</p> <p>X'01' Fixed length CCA token.</p> <p>X'02' Variable length CCA token.</p> <p>X'03' TR-31 key block.</p> <p>X'04' RKX token.</p> <p>X'05' RSA DSI PKCS #1 V2 OAEP format (PKCSOAEP).</p> <p>X'06' RSA DSI PKCS #1 block type 02 format (PKCS-1.2).</p> <p>X'07' Zero padded (ZERO-PAD).</p> <p>X'08' PKA92 format (PKA92).</p> <p>X'09' EMV or Smart Card format (EMVCRT, EMVDDA, EMVDDAE, SCCOMCRT, SCCOMME, or SCVISA).</p> <p>X'0A' Azure key object (CKM_RAKW).</p> <p>X'0B' AESKW external format.</p> <p>Note: When format is X'04' or greater, no other key or token-related fields are present.</p>
265	109	SMF82_TAG_KEY_SEC	1	binary	<p>Key security.</p> <p>X'01' No key present.</p> <p>X'02' Clear key.</p> <p>X'03' Key encrypted under master key.</p> <p>X'04' Key encrypted under key encrypting key.</p>

Table 164. Subtype 48 Compliance warning event (continued)

Dec	Hex	Name	Length	Format	Description
266	10A	SMF82_TAG_KEY_ALG	1	binary	<p>Key algorithm.</p> <p>X'02' DES.</p> <p>X'03' AES.</p> <p>X'04' HMAC.</p> <p>X'07' RSA.</p> <p>X'08' DSA.</p> <p>X'09' ECC.</p> <p>X'0E' ML-DSA, CRYSTALS-Dilithium.</p> <p>X'0F' ML-KEM, CRYSTALS-Kyber.</p> <p>Note:</p> <p>When the algorithm is DSA, the only other key or token information present is the object type.</p>
267	10B	SMF82_TAG_KEY_TYPE	2	binary	<p>Key type.</p> <p>The key type from the token. See “Variable-length symmetric key token” in <i>z/OS Cryptographic Services ICSF Application Programmer's Guide</i> for the list of key types.</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. Present only when KDS_TYPE is CKDS. 2. Applies to variable-length CCA tokens only.
268	10C	SMF82_TAG_KEY_CV	8	binary	<p>Key control vector.</p> <p>The first 8 bytes of the control vector from the token.</p> <p>See Appendix C, Control vectors and changing control vectors with the CVT callable service, in <i>z/OS Cryptographic Services ICSF Application Programmer's Guide</i> for information on how to interpret the control vector.</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. Present only when KDS_TYPE is CKDS. 2. Applies to fixed-length DES CCA tokens only.
269	10D	SMF82_TAG_KEY_USAGE_CKDS	3 - 33	binary	<p>Key usage fields.</p> <p>Consists of a 1 byte count followed by one or more 2-byte key usage fields. Applies to variable-length CCA tokens only.</p> <p>See Appendix B, Key token formats, in <i>z/OS Cryptographic Services ICSF Application Programmer's Guide</i> for the list of key usage values for variable length tokens.</p> <p>Note: Present only when KDS_TYPE is CKDS.</p>
270	10E	SMF82_TAG_KEY_LEN	2	binary	<p>The length of the key (in bits).</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. Not present for HMAC tokens. 2. When KDS_TYPE is PKDS, this is the length of the public key. 3. This tag will not be present for ML-DSA, ML-KEM, CRYSTALS-Dilithium (8,7) Round 2 or 3, CRYSTALS-Dilithium (6,5) Round 3, and CRYSTALS-Kyber.

Table 164. Subtype 48 Compliance warning event (continued)					
Dec	Hex	Name	Length	Format	Description
272	110	SMF82_TAG_KEY_USAGE_PKDS	4	binary	<p>Key usage for private keys.</p> <p>Bit Meaning when set</p> <p>0 Undefined.</p> <p>1 Key management usage permitted.</p> <p>2 Signature usage permitted.</p> <p>3 Key translation permitted.</p> <p>4 Key agreement usage permitted.</p> <p>5 Nonrepudiation usage permitted.</p> <p>6 Key encipherment usage permitted.</p> <p>7 Data encipherment usage permitted.</p> <p>8 Key certificate sign usage permitted.</p> <p>9 Certificate Revocation List sign usage permitted.</p> <p>10 Only key encipher usage permitted during key agreement.</p> <p>11 Only key decipher usage permitted during key agreement.</p> <p>12-31 Reserved.</p> <p>Note: Present only when KDS_TYPE is PKDS.</p>
274	112	SMF82_TAG_KEY_EC_CURVE	1	binary	<p>ECC curve type.</p> <p>X'01' Prime curve.</p> <p>X'02' Brainpool curve.</p> <p>X'04' Edwards curve.</p> <p>X'05' Koblitz curve.</p> <p>Note: Present only when KDS_TYPE is PKDS and KEY_ALG is ECC.</p>
280	118	SMF82_TAG_KEY_TIV	variable	binary	<p>A key token identification value. A string of bytes present in the key token. Can be used to help uniquely identify a key token.</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. Present only when KDS_TYPE is CKDS and TOK_FMT is a fixed-length CCA token. 2. This is the 4-byte token validation value. For more information, see Appendix B, Key Token Formats (Sections “AES Key Token Formats” and “DES Key Token Formats”) in <i>z/OS Cryptographic Services ICSF Application Programmer's Guide</i>.

Table 164. Subtype 48 Compliance warning event (continued)

Dec	Hex	Name	Length	Format	Description
282	11A	SMF82_TAG_QSA_ALG_PARAM	2	Binary	QSA Algorithm Parameter. Only applicable to ML-DSA, ML-KEM, CRYSTALS-Dilithium and CRYSTALS-Kyber. X'0404' ML-DSA (4,4). X'0605' ML-DSA, CRYSTALS-Dilithium (6,5). X'0807' ML-DSA, CRYSTALS-Dilithium (8,7). X'0768' ML-KEM, CRYSTALS-Kyber (768). X'1024' ML-KEM, CRYSTALS-Kyber (1024).
283	11B	SMF82_TAG_TR31_KB_VER_ID	1	EBCDIC	The TR-31 key block version ID. Only applies to TR-31 key blocks. See “X9.143 (TR-31) key block header and optional block data” on page 320 for possible TR-31 header values.
284	11C	SMF82_TAG_TR31_KEY_USAGE	2	EBCDIC	The TR-31 key block key usage. Only applies to TR-31 key blocks. See “X9.143 (TR-31) key block header and optional block data” on page 320 for possible TR-31 header values.
285	11D	SMF82_TAG_TR31_MODE_USE	1	EBCDIC	The TR-31 key block mode of use. Only applies to TR-31 key blocks. See “X9.143 (TR-31) key block header and optional block data” on page 320 for possible TR-31 header values.
286	11E	SMF82_TAG_TR31_KEY_VER	2	EBCDIC	The TR-31 key block key version number. Only applies to TR-31 key blocks. See “X9.143 (TR-31) key block header and optional block data” on page 320 for possible TR-31 header values.
287	11F	SMF82_TAG_TR31_EXPORT	1	EBCDIC	The TR-31 key block exportability. Only applies to TR-31 key blocks. See “X9.143 (TR-31) key block header and optional block data” on page 320 for possible TR-31 header values.
288	120	SMF82_TAG_QSA_ALG_ID	1	Binary	QSA Algorithm Identifier. Only applicable to ML-DSA, ML-KEM, CRYSTALS-Dilithium and CRYSTALS-Kyber. X'01' CRYSTALS-Dilithium Round 2. X'02' CRYSTALS-Kyber Round 2. X'03' CRYSTALS-Dilithium Round 3. X'04' CRYSTALS-Kyber Round 3. X'05' Pure ML-DSA. X'06' ML-KEM. X'07' Pre-hash ML-DSA.

Table 164. Subtype 48 Compliance warning event (continued)					
Dec	Hex	Name	Length	Format	Description
525	20D	SMF82_TAG_KEY_DIR	1	binary	Key parameter direction. X'01' Input key. X'02' Output key.
526	20E	SMF82_TAG_KEY_AGE	1	binary	Key age. X'01' Pre-existing key. X'02' Newly generated key.
527	20F	SMF82_TAG_RC	4	binary	The return code from the attempt to compliance check the token. Present only when COMP_CHK indicates an evaluation error.
528	210	SMF82_TAG_RS	4	binary	The reason code to go along with the return code from the attempt to compliance check the token. Present only when COMP_CHK indicates an evaluation error.

The following tags may be present in the end user audit section:

- X500_IDN
- X500_SDN
- IDID_USRI
- IDID_USRF
- IDID_REG
- USRI

See [“Audit header and audit section” on page 414](#) for more details.

Subtype 49 - Master key event resulted in a new master key value being promoted to current

This record is written when a new master key is promoted to current master key by ICSF. This record is not written when the TKE set immediate operation is used to set the current master key. If TKE is used to set the new master key and ICSF promotes the master key to current, or TKE is used to drive a coordinated change master key in ICSF, the subtype 49 record is written.

This subtype consists of a number of tag-length-value (TLV) triplets. The following triplets may be contained in the record. The specific set of triplets depends on the type of event and the information that is available.

Table 165. Subtype 49 Master key event resulted in a new master key value being promoted to current					
Dec	Hex	Name	Length	Format	Description
529	211	SMF82_TAG_SYSNAME	8	EBCDIC	Name of system that wrote the SMF record

Table 165. Subtype 49 Master key event resulted in a new master key value being promoted to current (continued)

Dec	Hex	Name	Length	Format	Description
530	212	SMF82_TAG_MK_EVENT_FLAG	1	binary	Bit Meaning when set 0 ON when this system initiated a coordinated change master key. 1 ON when a change master key occurred on this system. 2 ON when a new master key is promoted to current master key outside of a change master key. Note: If no defined bits are ON in this field, the triplet may or may not be present in the SMF record.
258	102	SMF82_TAG_KDS_DSNAME	44	EBCDIC	The dataset name of the KDS, after the event has completed, corresponding to the event.
524	20C	SMF82_TAG_KDS_TYPE	1	binary	The KDS type: X'01' CKDS. X'02' PKDS. X'03' TKDS.

Table 165. Subtype 49 Master key event resulted in a new master key value being promoted to current
(continued)

Dec	Hex	Name	Length	Format	Description
531	213	SMF82_TAG_SERIAL_NUMS	variable	binary	<p>(1 byte master key type, 1 byte serial number count, array of serial numbers)</p> <p>The first byte indicates type of master key affected by the event. Master key event type:</p> <p>X'01' AES.</p> <p>X'02' DES.</p> <p>X'03' RSA.</p> <p>X'04' ECC.</p> <p>X'05' P11.</p> <p>X'06' Reserved.</p> <p>Following the first byte is a one-byte field (<i>num</i>) indicating the number of coprocessors that were affected by the event.</p> <p>Following <i>num</i> is a list of 8-byte serial numbers that were affected by the event. The list contains <i>num</i> serial numbers.</p>

Table 165. Subtype 49 Master key event resulted in a new master key value being promoted to current
(continued)

Dec	Hex	Name	Length	Format	Description
532	214	SMF82_TAG_MKVP	variable	binary	<p>(1 byte master key type, 1 byte count of valid MKVP nibbles displayed, MKVP value)</p> <p>The first byte indicates the key type. Key type:</p> <p>X'01' AES.</p> <p>X'02' DES.</p> <p>X'03' RSA.</p> <p>X'04' ECC.</p> <p>X'05' P11.</p> <p>X'06' Reserved.</p> <p>The second byte indicates the number of valid nibbles length (<i>len</i>) in the MKVP value that follows. <i>len</i> has been adjusted according to the MKCVLEN installation option (for example, the number of valid MKVP nibbles written may be truncated).</p> <p>Following the second byte is the MKVP for the master key that was made current as the result of the event. The MKVP is left justified and padded with zeroes.</p>
533	215	SMF82_TAG_DOMAIN	1	binary	Cryptographic domain.
534	216	SMF82_TAG_TOD	16	binary	<p>Time of event in STCKE format.</p> <p>This TOD will match the D ICSF,MKVPs and D ICSF,KDS MKVP date for a key type when the master key promotion was due to a master key change, the initialization of the KDS, or the update to the KDS to add the MKVP for that key type. When the sys1.samplib member CSFSMFR is used to format the SMF records, the formatted TOD matches the D ICSF command output and is in ISO 8601 format YYYY-MM-DD HH:MM:ss.</p> <p>Note: In the unusual and not recommended case where more than one type of new master key is loaded in a coprocessor or coprocessors and a KDS is being updated, more than one type of new MKVP could be promoted as a result of the update to the KDS. The TOD reported is for the most recently promoted MKVP for that coprocessor.</p>

Record type 1154 (X'482') Subtype 49 – ICSF Compliance Evidence

Record type 1154 Subtype 49 is written to record ICSF compliance evidence when an ENF86 signal is received. Record type 1154 Subtype 49 is mapped by the CSFZ1154 macro.

See [z/OS MVS System Management Facilities \(SMF\)](#) for a complete description of the SMF 1154 record and subtypes.

Record type 1154 (X'482') Subtype 49 header

Table 166. Record type 1154 Subtype 49 header					
Offsets		Name	Length	Format	Description
0	0	Smf1154_49_Trn	2	Binary	Number of triplets: 2
2	2	Smf1154_49_Rsv1	2		Reserved (X'00')
4	4	Smf1154_49_1_Offset	4	Binary	Offset to first data section 1
8	8	Smf1154_49_1_Length	2	Binary	Length of data section 1
10	A	Smf1154_49_1_Number	2	Binary	Number of repeated data section 1's: 1
12	C	Smf1154_49_2_Offset	4	Binary	Offset to first data section 2
16	10	Smf1154_49_2_Length	2	Binary	Length of data section 2
18	12	Smf1154_49_2_Number	2	Binary	Number of repeated data section 2's: 1

Record type 1154 (X'482') Subtype 49 Data section 1

Data section 1 reports various ICSF security settings for ICSF key data sets, ICSF resource classes, key store policy, and other configuration and audit options.

Table 167. Record type 1154 Subtype 49 Data section 1					
Offsets		Name	Length	Format	Description
0	0	SMF1154_49_1_VERSION	2	Binary	Version of this section: 1
2	2	SMF1154_49_1_PROTECTALLFAIL	1	Binary	RACF PROTECTALL setting: X'01' PROTECTALL(FAIL) X'00' PROTECTALL(WARN) or NOPROTECTALL
3	3	SMF1154_49_1_CHKAUTH	1	Binary	Setting of CHKAUTH keyword in ICSF options dataset.
4	4	SMF1154_49_1_RSV1	28		Reserved.
32	20	SMF1154_49_1_XFACILIT_ACT	1	Binary	XFACILIT class status: X'01' Active X'00' Not Active
33	21	SMF1154_49_1_XFACILIT_RACL	1	Binary	XFACILIT class RACLIST status: X'01' Raclisted X'00' Not Raclisted
34	22	SMF1154_49_1_KSPCTOKCHKLBLWARN	1	Binary	CSF.CKDS.TOKEN.CHECK.LABEL.WARN setting: X'01' Enabled X'00' Not Enabled

Table 167. Record type 1154 Subtype 49 Data section 1 (continued)

Offsets		Name	Length	Format	Description
35	23	SMF1154_49_1_KSPCTOKCHKLBLEFAIL	1	Binary	CSF.CKDS.TOKEN.CHECK.LABEL.FAIL setting: X'01' Enabled X'00' Not Enabled
36	24	SMF1154_49_1_KSPPTOKCHKLBLEWARN	1	Binary	CSF.PKDS.TOKEN.CHECK.LABEL.WARN setting: X'01' Enabled X'00' Not Enabled
37	25	SMF1154_49_1_KSPPTOKCHKLBLEFAIL	1	Binary	CSF.PKDS.TOKEN.CHECK.LABEL.FAIL setting: X'01' Enabled X'00' Not Enabled
38	26	SMF1154_49_1_KSPCTOKCHKDFTLBLE	1	Binary	CSF.CKDS.TOKEN.CHECK.DEFAULT.LABEL setting: X'01' Enabled X'00' Not Enabled
39	27	SMF1154_49_1_KSPPTOKCHKDFTLBLE	1	Binary	CSF.PKDS.TOKEN.CHECK.DEFAULT.LABEL setting: X'01' Enabled X'00' Not Enabled
40	28	SMF1154_49_1_KSP_CTOKNODUPS	1	Binary	CSF.CKDS.TOKEN.NODUPLICATES setting: X'01' Enabled X'00' Not Enabled
41	29	SMF1154_49_1_KSP_PTOKNODUPS	1	Binary	CSF.PKDS.TOKEN.NODUPLICATES setting: X'01' Enabled X'00' Not Enabled
42	2A	SMF1154_49_1_KSP_XKEYENABLEAES	1	Binary	CSF.XCSFKEY.ENABLE.AES setting: X'01' Enabled X'00' Not Enabled
43	2B	SMF1154_49_1_KSP_XKEYENABLEDES	1	Binary	CSF.XCSFKEY.ENABLE.DES setting: X'01' Enabled X'00' Not Enabled
44	2C	SMF1154_49_1_KSP_KEYSAUTHWARN	1	Binary	CSF.CSFKEYS.AUTHORITY.LEVELS.WARN setting: X'01' Enabled X'00' Not Enabled
45	2D	SMF1154_49_1_KSP_KEYSAUTHFAIL	1	Binary	CSF.CSFKEYS.AUTHORITY.LEVELS.FAIL setting: X'01' Enabled X'00' Not Enabled

Table 167. Record type 1154 Subtype 49 Data section 1 (continued)					
Offsets		Name	Length	Format	Description
46	2E	SMF1154_49_1_KSP_ARCHUSE	1	Binary	CSF.KDS.KEY.ARCHIVE.USE setting: X'01' Enabled X'00' Not Enabled
47	2F	SMF1154_49_1_KSP_ARCHDATADEC	1	Binary	CSF.KDS.KEY.ARCHIVE.DATA.DECRYPT setting: X'01' Enabled X'00' Not Enabled Note: This field is reserved (X'00') on ICSF FMID HCR77D1.
48	30	SMF1154_49_1_KSP_KGUPAUTHCHK	1	Binary	CSF.KGUP.CSFKEYS.AUTHORITY.CHECK setting: X'01' Enabled X'00' Not Enabled
49	31	SMF1154_49_1_KSP_ECCPVTKEYNAME	1	Binary	CSF.CSFKEYS.ECC.PRIVATEKEYNAME.ENABLE setting: X'01' Enabled X'00' Not Enabled Note: This field is reserved (X'00') on ICSF FMID HCR77D1.
50	32	SMF1154_49_1_RSV2	16		Reserved.
66	42	SMF1154_49_1_AKL_CLBL	1	Binary	AUDITKEYLIFECKDS(LABEL()) X'01' YES X'00' NO
67	43	SMF1154_49_1_AKL_CTOK	1	Binary	AUDITKEYLIFECKDS(TOKEN()) X'01' YES X'00' NO
68	44	SMF1154_49_1_AKL_PLBL	1	Binary	AUDITKEYLIFECPKDS(LABEL()) X'01' YES X'00' NO
69	45	SMF1154_49_1_AKL_PTOK	1	Binary	AUDITKEYLIFECPKDS(TOKEN()) X'01' YES X'00' NO
70	46	SMF1154_49_1_AKL_TTOKO	1	Binary	AUDITKEYLIFETKDS(TOKENOBJ()) X'01' YES X'00' NO
71	47	SMF1154_49_1_AKL_TSESSO	1	Binary	AUDITKEYLIFETKDS(SESSIONOBJ()) X'01' YES X'00' NO
72	48	SMF1154_49_1_RSV3	8		Reserved.

Table 167. Record type 1154 Subtype 49 Data section 1 (continued)					
Offsets		Name	Length	Format	Description
80	50	SMF1154_49_1_AKU_CLBL	1	Binary	AUDITKEYUSGCKDS(LABEL()) X'01' YES X'00' NO
81	51	SMF1154_49_1_AKU_CTOK	1	Binary	AUDITKEYUSGCKDS(TOKEN()) X'01' YES X'00' NO
82	52	SMF1154_49_1_AKU_PLBL	1	Binary	AUDITKEYUSGPKDS(LABEL()) X'01' YES X'00' NO
83	53	SMF1154_49_1_AKU_PTOK	1	Binary	AUDITKEYUSGPKDS(TOKEN()) X'01' YES X'00' NO
84	54	SMF1154_49_1_AKU_P11TOKO	1	Binary	AUDITPKCS11USG(TOKENOBJ()) X'01' YES X'00' NO
85	55	SMF1154_49_1_AKU_P11SESSO	1	Binary	AUDITPKCS11USG(SESSIONOBJ()) X'01' YES X'00' NO
86	56	SMF1154_49_1_RSV4	32		Reserved.
118	76	SMF1154_49_1_CC_SERVICES	1	Binary	Existence of installation defined services in the ICSF installation options dataset. X'01' At least one SERVICE() entry specified in the ICSF installation options dataset. X'00' No SERVICE() entries are defined in the ICSF installation options dataset.
119	77	SMF1154_49_1_CC_EXITS	1	Binary	Existence of installation exits in the ICSF installation options dataset. X'01' At least one EXIT() entry specified in the ICSF installation options dataset. X'00' No EXIT() entries in the ICSF installation options dataset.
120	78	SMF1154_49_1_RSV5	4		Reserved.

Table 167. Record type 1154 Subtype 49 Data section 1 (continued)

Offsets		Name	Length	Format	Description
124	7C	SMF1154_49_1_KDSFORMAT	3	Binary	<p>ICSF key data set (KDS) format:</p> <p>Byte:</p> <p>1 CKDS format.</p> <p>2 PKDS format.</p> <p>3 TKDS format.</p> <p>Byte meaning when set:</p> <p>X'00' Not defined.</p> <p>X'01' Empty KDS.</p> <p>X'02' Non-KDSR format.</p> <p>X'03' KDSR format.</p> <p>X'04' KDSRL format.</p> <p>Note: This value is applicable only on z/OS V2R5 (ICSF FMID HCR77D2).</p>
127	7F	SMF1154_49_1_CLASS	292 * 4		<p>ICSF class information.</p> <p>Four contiguous instances are each mapped by the SMF1154_49_CLASS definition in Table 168 on page 481.</p> <ul style="list-style-type: none"> The first instance is the CSFSERV class profile access information. The second instance is the CSFKEYS class profile access information. The third instance is the CRYPTOZ class profile access information. The fourth instance is the XCSFKEY class profile access information.
1295	50F	SMF1154_49_1_DFLTLBL	292 * 2		<p>Default label checking for CKDS and PKDS.</p> <p>Two contiguous instances each mapped by the SMF1154_49_DL_CLASS definition in Table 169 on page 482.</p> <ul style="list-style-type: none"> The first instance is the CKDS default label access controls. The second instance is the PKDS default label access controls.
1879	757	SMF1154_49_1_KDS	327 * 3		<p>CKDS, PKDS, and TKDS protection settings.</p> <p>Three contiguous instances each mapped by the SMF1154_49_KDS KDS access controls in Table 170 on page 483.</p> <ul style="list-style-type: none"> The first instance is the CKDS access controls. The second instance is the PKDS access controls. The third instance is the TKDS access controls.

Table 168. SMF1154_49_CLASS profile access

Offsets		Name	Length	Format	Description
0	0	SMF1154_49_CLS_NAME	8	EBCDIC	<p>Class Name.</p> <p>CSFSERV, CSFKEYS, CRYPTOZ, and XCSFKEY.</p>

Table 168. SMF1154_49_CLASS profile access (continued)

Offsets		Name	Length	Format	Description
8	8	SMF1154_49_CLS_ACTIVE	1	Binary	Class ACTIVE: X'01' YES X'00' NO
9	9	SMF1154_49_CLS_RACLISTED	1	Binary	Class RACLISTed: X'01' YES X'00' NO
10	A	SMF1154_49_CLS_PROFLEN	1	Binary	Length of profile name.
11	B	SMF1154_49_CLS_PROF	246	Char	Profile name. **' Or ***'
257	101	SMF1154_49_CLS_PROFUACC	1	Binary	Profile UACC setting. Bit Meaning When Set 0 ALTER access. 1 CONTROL access. 2 UPDATE access. 3 READ access. 4 EXECUTE access. 5-6 Reserved for IBM's use. 7 NONE access.
258	102	SMF1154_49_CLS_PROFWARN	1	Binary	Profile WARN setting. Identifies the data set as having the WARNING attribute on (bit 0 or bit 7 is on) or not having the WARNING attribute on.
259	103	SMF1154_49_CLS_PROFIDSPLAT	1	Binary	ID(*) setting: X'01' ID(*) Access. X'00' ID(*) Not defined.
260	104	SMF1154_49_CLS_RSV	32		Reserved.

Table 169. SMF1154_49_DL_CLASS KDS default label access controls

Offsets		Name	Length	Format	Description
0	0	SMF1154_49_DL_CLS_NAME	8	EBCDIC	Class name. CSFKEYS.
8	8	SMF1154_49_DL_CLS_ACTIVE	1	Binary	Class ACTIVE: X'01' YES X'00' NO

Table 169. SMF1154_49_DL_CLASS KDS default label access controls (continued)					
Offsets		Name	Length	Format	Description
9	9	SMF1154_49_DL_CLS_RACLISTED	1	Binary	Class RACLISTed: X'01' YES X'00' NO
10	A	SMF1154_49_DL_CLS_PROFLEN	1	Binary	Length of profile name.
11	B	SMF1154_49_DL_CLS_PROF	246	EBCDIC	Profile name. CSF-CKDS-DEFAULT and CSF-PKDS-DEFAULT.
257	101	SMF1154_49_DL_CLS_PROFUACC	1	Binary	Profile UACC setting. Bit Meaning When Set 0 ALTER access. 1 CONTROL access. 2 UPDATE access. 3 READ access. 4 EXECUTE access. 5-6 Reserved for IBM's use. 7 NONE access.
258	102	SMF1154_49_DL_CLS_PROFWARN	1	Binary	Profile WARN setting. Identifies the data set as having the WARNING attribute on (bit 0 or bit 7 is on) or not having the WARNING attribute on.
259	103	SMF1154_49_DL_CLS_PROFIDSPLAT	1	Binary	ID(*) setting: X'01' ID(*) Access. X'00' ID(*) Not defined.
260	104	SMF1154_49_DL_CLS_RSV	32		Reserved.

Table 170. SMF1154_49_KDS KDS access controls					
Offsets		Name	Length	Format	Description
0	0	SMF1154_49_KDS_TYPE	1	Binary	The KDS type: X'00' Not defined. X'01' CKDS. X'02' PKDS. X'03' TKDS.
1	1	SMF1154_49_KDS_NAME	44	EBCDIC	KDS name.
45	2D	SMF1154_49_KDS_PROFLEN	1	Binary	Length of the profile protecting the KDS. If X'00', no profile has been defined to protect the KDS.
46	2E	SMF1154_49_KDS_PROF	246	EBCDIC	The name of the profile protecting the KDS.

Table 170. SMF1154_49_KDS KDS access controls (continued)					
Offsets	Name		Length	Format	Description
292	124	SMF1154_49_KDS_PROFUACC	1	Binary	Profile UACC setting. Bit Meaning When Set 0 ALTER access. 1 CONTROL access. 2 UPDATE access. 3 READ access. 4 EXECUTE access. 5-6 Reserved for IBM's use. 7 NONE access.
293	125	SMF1154_49_KDS_PROFWARN	1	Binary	Profile WARN setting. Identifies the data set as having the WARNING attribute on (bit 0 or bit 7 is on) or not having the WARNING attribute on.
294	126	SMF1154_49_KDS_PROFIDSPLAT	1	Binary	ID(*) settings.
295	127	SMF1154_49_KDS_RSV	32		Reserved.

Record type 1154 (X'482') Subtype 49 Data section 2

Data section 2 reports the algorithm names and their counts used in ICSF services since the last ENF86 signal was received. You must have Cryptographic algorithm (ALG) usage statistics enabled for data section 2 to be included in the SMF type 1154 Subtype 49 record.

Table 171. Record type 1154 Subtype 49 Data section 2					
Offsets	Name		Length	Format	Description
0	0	Smf1154_49_2_Version	2	Binary	Version of this section: 1.
2	2	Smf1154_49_2_Rsv1	2		Reserved.
4	4	Smf1154_49_2_AlgsCount	4	Binary	Number of entries in Smf1154_49_2_Algs.
8	8	Smf1154_49_2_Algs	16 * Smf1154_49_2_AlgsCount		Algorithm count information since the ENF86 signal was received. Each instance is mapped by Table 172 on page 484 .

Table 172. Smf1154_49_2_Algs algorithm count information					
Offsets	Name		Length	Format	Description
0	0	Smf1154_49_2_Algs_Func	8	EBCDIC	Algorithm name. See Table 156 on page 437 .
8	8	Smf1154_49_2_Algs_Count	8	Binary	Number of times the algorithm was used during the ENF86 interval.

Appendix C. CICS-ICSF Attachment Facility

The purpose of the CICS-ICSF Attachment Facility is to enhance the performance of CICS transactions in the same region as a transaction using long-running ICSF services such as the PKA services and CKDS or PKDS update services.

Without the CICS-ICSF Attachment Facility, the application that requests a long-running ICSF service is placed into an OS WAIT. With the CICS-ICSF Attachment Facility, a long running service is transferred to an L8, and the CICS application is placed into a CICS WAIT, rather than an OS WAIT, for the duration of the operation.

Note: The CICS-ICSF Attachment Facility can only be used by 31-bit assembler stub functions. The CICS-ICSF Attachment Facility cannot be used when invoking ICSF APIs with C linkage or 64-bit assembler stub functions. See *Combining C or C++ and Assembler in z/OS XL C/C++ Programming Guide* for information on invoking the 31-bit assembler stubs from a C/C++ program.

Installing the CICS-ICSF Attachment Facility

Before you can use the CICS-ICSF Attachment Facility, the ICSF system programmer, or the CICS administrator needs to install it. This involves:

- Relinking the ICSF enabling routine, CSFATREN, and the ICSF TRUE, CSFATRUE, if ICSF was previously installed in an environment without the CICS-ICSF Attachment Facility
- Installing the proper load libraries in the PROC used to start CICS
- Updating the CICS System Definitions (CSD) data set to define the programs to CICS
- Enabling these programs

For information about CICS TRUE programs, refer to [CICS Transaction Server for z/OS, Version 5 Release 1 \(www.ibm.com/docs/en/cics-ts\)](http://www.ibm.com/docs/en/cics-ts).

Steps for installing the CICS-ICSF attachment facility

1. If ICSF was previously installed in an environment without the CICS-ICSF Attachment Facility (i.e., without being linked with the CICS SDFHLOAD data set), the ICSF system programmer will need to relink the ICSF TRUE, CSFATRUE, and the ICSF enabling routine, CSFATREN. This would be the case if, for example, (a) the DDDEF entries for ICSF do not have the SDFHLOAD DDDEF pointing to the CICS SDFHLOAD data set but instead have it pointing to an empty data set, or (b) z/OS (and hence ICSF) was installed using a ServerPac.

To relink the ICSF modules, first manually update the ICSF DDDEF for SDFHLOAD to point to the CICS SDFHLOAD data set. (Refer to ICSF sample CSFDDDEF shipped in SAMPLIB.) Then submit a job to relink the ICSF modules. This is an example of job control language for the relink.

```
//STEP01      EXEC PGM=IEWL,
//      PARM='LIST,XREF,LET,DCBS,RENT,REUS,AMODE(31),RMODE(ANY)'
//SYSLMOD DD DISP=SHR,DSN=yyy.SCSFSTUB (the ICSF load library)
//SYSLIB DD DISP=SHR,DSN=xxxxxx.SDFHLOAD
//SDFHLOAD DD DISP=SHR,DSN=xxxxxx.SDFHLOAD
//SCSFMOD0 DD DISP=SHR,DSN=yyy.SCSFMOD0 (the ICSF load library)
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(10,10))
//SYSPRINT DD SYSOUT=*
//SYSLIN DD *
            INCLUDE SDFHLOAD(DFHEAI)
            REPLACE CSFDHEAI(DFHEAI),CSF0EAI
            INCLUDE SCSFMOD0(CSFATREN)
            ENTRY DFHEAI
            NAME CSFATREN(R)
            INCLUDE SDFHLOAD(DFHEAI)
            REPLACE CSFDHEAI(DFHEAI),CSF0EAI
            INCLUDE SCSFMOD0(CSFATRUE)
            ENTRY DFHEAI
```

```

NAME CSFATRUE(R)
/*

```

2. Include the ICSF load module data set in the CICS startup job control language as shown in this example.

```

//DFHRPL DD DISP=SHR,DSN=xxxxx.SDFHLOAD
// DD DISP=SHR,DSN=yyy.SCSFSTUB (ICSF callable service stubs)
// DD DISP=SHR,DSN=yyy.SIEALNKE (ICSF shared libraries)
// DD ...
...
//SYSIN DD DISP=SHR,DSN=xxxxx.SYSIN(DFH$SIPx)
...

```

In the previous sample code, DFH\$SIPx includes the entry:

```

PLTPI=yy,

```

3. Customize the Program Load Table (PLT), to include the ICSF enabling routine CSFATREN in second stage initialization.

This is an example input deck for compiling a PLT for automatic enablement of the CICS-ICSF link. This is ASM code. Assemble it with the CICS macro library, but **without** the CICS translator.

```

//SYSIN DD *
*
* List of programs to be executed sequentially during system
* initialization. Required system initialization parm: PLTPI=yy
* DFHPLTCS should be defined in the CSD by CEDA or DFHCSDUP job
*
DFHPLT TYPE=INITIAL,SUFFIX=yy
*
* ----- Second stage of initialization -----
*
DFHPLT TYPE=ENTRY,PROGRAM=CSFATREN (Run enable of CSFATRUE)
*
* ----- Delimiter between Stages 2 and 3 -----
*
DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
*
* ----- Third stage of initialization -----
* (none)
*
DFHPLT TYPE=FINAL
END
/*

```

The previous code is an example only. Your CICS administrator can use it as a guide in customizing the PLT. For more information about coding the PLT, refer to [CICS Transaction Server for z/OS, Version 5 Release 1 \(www.ibm.com/docs/en/cics-ts\)](http://www.ibm.com/docs/en/cics-ts).

4. Link edit the PLT with these controls:

```

INCLUDE OBJLIB(DFHPLTy)
NAME DFHPLTy(R)

```

5. The CICS administrator should customize the system CSD to include:

- CSFATRUE
- CSFATREN
- A PLT to indicate that initialization is to call CSFATREN to enable the ICSF TRUE, CSFATRUE

This is an example of the job control language and input. In this example, xxxxx represents the local CICS prefix, and zzzzzzzz represents the PLT entry that was compiled previously.

```

//UPDATE JOB ...
//*- - - - -
//DEFINES EXEC PGM=DFHCSDUP,REGION=2M
//STEPLIB DD DISP=SHR,DSN=xxxxxx.SDFHLOAD
// DD DISP=SHR,DSN=zzzzzzzz
//DFHCSD DD DISP=SHR,DSN=xxxxxx.DFHCSD
//SYSPRINT DD SYSOUT=A

```

```
//SYSIN      DD *
*
DEFINE PROGRAM(CSFATREN) GROUP(ICSF)
                        DESCRIPTION(TRUE enablement routine)
                        LANGUAGE(ASSEMBLER)

*
DEFINE PROGRAM(CSFATRUE) GROUP(ICSF)
                        DESCRIPTION(ICSF interface TRUE)
                        LANGUAGE(ASSEMBLER)
                        CONCURRENCY(THREADSAFE)
                        API(OPENAPI)

*
DEFINE PROGRAM(DFHPLTy) GROUP(ICSF)
                        DESCRIPTION(PLT Program Init for CSFATRUE)
                        LANGUAGE(ASSEMBLER)
```

The PLT in the example runs the program CSFATREN during CICS initialization. CSFATREN automatically enables the ICSF TRUE, CSFATRUE. If CICS is already started, use a CICS Command Level Interpreter Transaction (CECI) to enable CSFATRUE. To do this, go into CECI and issue this statement:

```
ENABLE PROGRAM('CSFATRUE') TALENGTH(1536) LINKEDITMODE START
```

You can also do this in a single step with this statement:

```
CECI ENABLE PROGRAM('CSFATRUE') TALENGTH(1536) LINKEDITMODE START
```

6. If you have any existing CICS applications which invoke any of the ICSF services in the Wait List, then these applications must be re-linked with the current ICSF stubs.

Implementing the CICS wait list

The CICS Wait List can be implemented by means of a customer modifiable data set, pointed to by the Installation Options Data Set (WAITLIST parameter). The default WAITLIST includes all services which can complete asynchronously (for example, those services which perform I/O to a key data set and those services which are routed to a cryptographic processor). If the option is not specified, the default CICS Wait List will be utilized by ICSF when a CICS application invokes an ICSF callable service. If WAITLIST is specified, the data set specified by this parameter will be used to determine the names of the services to be placed on the CICS Wait List. A sample data set is provided by ICSF via member CSFWTL01 of SYS1.SAMPLIB. The sample data set contains the same entries as the default ICSF CICS Wait List -- for example, the data set contains the names of all ICSF callable services which, by default, will be driven through the CICS TRUE.

The WAITLIST option should be added to the Installation Options data set under these conditions.

- CICS customers who want to use the default CICS Wait List shipped with ICSF do not need to specify a WAITLIST keyword. If you have any existing CICS applications which invoke any of the ICSF services in the Wait List, then these applications must be re-linked with the current ICSF stubs.
- CICS customers who do not want to make use of CICS TRUE must either not enable the TRUE or specify a WAITLIST keyword and point to an empty wait list data set or you can specify WAITLIST(DUMMY) in the Installation Options data set.
- CICS customers who wish to modify the ICSF default CICS Wait List should modify the sample Wait List data set supplied in member CSFWTL01 of SYS1.SAMPLIB. The WAITLIST keyword in the Installation Options Data Set should be set to point to this data set. If you have any existing CICS applications which invoke any of the ICSF services in the Wait List, then these applications must be re-linked with the current ICSF stubs.

To ensure maximum performance, any existing CICS applications which invoke any of the ICSF services in the Wait List that were linked with ICSF stubs prior to HCR7770 should be re-linked with the current ICSF stubs.

If you already have the CICS-ICSF Attachment facility installed, there are a number of callable services which may potentially be routed to a coprocessor or may perform other asynchronous processing. If you have a modified CICS Wait List, you should ensure that the wait list data set includes all such

services, and any CICS applications which invoke any of these services are re-linked with the current ICSF stubs. As a model, you can use the default CICS Wait List that is shipped with ICSF which includes all services which have an asynchronous interface to ICSF or you can use a sample Wait List data set that is also shipped with ICSF. The sample CICS Wait List data set is contained in member CSFWTL01 of SYS1.SAMPLIB. The sample data set contains the same entries as the default ICSF CICS Wait List. If you have an application which invokes a UDX while running under CICS, then the name of the UDX generic service should be added to the CICS Wait List.

If you use a CICS Wait List data set, you need to identify the data set to ICSF through the WAITLIST(data_set_name) option in the ICSF Installation Options data set. The data set can be a member of a PARMLIB, a member of a partitioned data set, or a sequential data set. The data set should be allocated on a permanently resident volume and should adhere to:

- The format of each record in the data set must be fixed length or fixed block length.
- A physical line in the data set must be a LRECL of 80 characters long. The system ignores any characters in positions 73 to 80 of the line.
- You can delimit comments by "/" and "*" and include them anywhere in the text. A comment cannot span physical records.
- Only one service may be specified on a logical line.

Note: You can use the WAITLIST(DUMMY) parameter to specify a null CICS Wait List data set, or you can disable the CICS TRUE if you do not want to utilize the CICS TRUE. See [“Parameters in the installation options data set” on page 35](#) for additional information.

Appendix D. Helpful hints for ICSF first time startup

The purpose of this topic is to provide some helpful hints and resolutions for the problems that you may encounter when starting ICSF for the first time.

See [Appendix F, “Systems without Cryptographic features,” on page 499](#) if you're running in this environment.

Checklist for first-time startup of ICSF

This is a checklist for the first-time startup of ICSF.

Note: ALL crypto coprocessors cards must be loaded with the same level of code. Otherwise, unpredictable results can occur.

Step 1. Hardware setup

Note: The CP Assist for Cryptographic Functions feature is required for selection of the coprocessor in the activation profiles.

Process

LIC installed for CP Assist for Cryptographic Functions

Note: If using TKE, you must Permit each coprocessor for TKE Commands.

Responsible

CE or Client Operator Representative

Where

Support Element

Verify

Via CPC details

- CP Assist for Cryptographic Functions is 'Installed'
- CP Assist for Cryptographic Functions DES/TDES enablement (feature 3863) is 'Installed'

Via PCI Cryptographic Configuration Task

- Status for each coprocessor or accelerator is 'Configured'

Note: If using TKE, the status for each Coprocessor is "Permitted".

References

Support Element Operations Guide

Completed

Step 2. LPAR activation profiles

Process

PCI Crypto Page Setup

Responsible

CE or Client Operator Representative

Where

Support Element

Verify

Control Domain Index

Usage Domain Index

PCI Cryptographic Candidate List includes all CCA Crypto Express coprocessors and accelerators that CAN be online

PCI Cryptographic Online List includes all CCA Crypto Express coprocessors and accelerators that WILL be online when activation is complete (Selections in the Online List MUST be selected in the Candidate List)

References

Support Element Operations Guide

z/OS Cryptographic Services ICSF TKE Workstation User's Guide (LPAR Considerations)

IBM Z PR/SM Planning Guide

Completed

Note: If TKE is to be used, ALL cryptographic coprocessors that you want TKE to be able to control MUST be defined in the Online and Candidate Lists. Also, the Usage Domain for the TKE Host LPAR **MUST** be unique. While the same domain may be used by other LPARs as long as these LPARs do not share any of the same cards, the TKE Host domain must have access to all the cards so that prohibits any other LPAR from using the same domain.

Step 3. ICSF setup

Process

Install and Customize ICSF

Responsible

System Programmer and ICSF Administrator

Where

TSO and ISPF Panels

Verify

Customize SYS1.PARMLIB

- Add CSF.SCSFMODE0 and CSF.SCSFSTUB to the LNKLIST concatenation
- Update PROGxx to APF authorize CSF.SCSFMODE0 and CSF.SCSFSTUB
- Update IKJTSOxx for ICSF by adding CSFDAUTH and CSFDPKDS to the AUTHPGM and AUTHTSF parameter lists. To change the active IKJTSOxx member of SYS1.PARMLIB, use the PARMLIB UPDATE command.

CKDS and PKDS created

ICSF Startup Procedure created

Installation Options Dataset created

- The DOMAIN parameter in the installation options data set is optional. It is required if more than one domain is specified as the usage domain on the PR/SM panels or if running in native mode.
- CKDS and PKDS names specified
- COMPAT(NO)

Access provided to the ICSF panels

References

[Chapter 2, "Installation, initialization, and customization," on page 9](#)

Completed

Step 4. TKE setup

If you are not using TKE, proceed to the next step.

Process

Initialize the TKE Workstation.

Configure TCP/IP on the Host and the TKE Workstation.

Perform passphrase or smart card setup.

Setup the TKE Host Transaction Program:

- Create JCL to start the TKE Host Transaction Program.
- RACF (or equivalent product) Security Setup.
- Start the TKE Host Transaction Program.

Responsible

Network Programmer, System Programmer and TKE Administrator.

Where

ISPF Panels, TKE Workstation.

Verify

CSFTTKE is authorized in the AUTHCMD list of IKJTSoxx in SYS1.PARMLIB.

TKE Host Transaction Program (CSFTTCP) is defined in the RACF STARTED class (Note: If your installation has a Generic Userid associated to all started procedures, this is not necessary).

CSFTTKE profile is defined in the RACF FACILITY and RACF APPL classes.

The userid associated with the TKE Host Transaction Program (CSFTTCP) must be authorized to the CSFCRC, CSFKIM, CSFKRC, CSFKRD, CSFKRR, CSFKRW, CSFKYT, CSFKYT2, CSFPCI, CSFPKRC, CSFPKRW, and CSFPKI profiles in the CSFSERV class.

References

[*z/OS Cryptographic Services ICSF TKE Workstation User's Guide*](#) (See Topics: TKE Workstation Setup and Customization and TKE TCP/IP and Host Considerations.)

Completed

Step 5. ICSF startup

Process

Start ICSF

Responsible

Client Operator Representative or System Programmer

Where

Operator Console

References

[Chapter 2, "Installation, initialization, and customization," on page 9](#)

Completed

Step 6. Loading master keys and initializing the CKDS through ICSF panels

Note: When defining a master key by specifying master key parts, make sure that the key parts are recorded and saved in a secure location. When you are entering the key parts for the first time, be aware that you might need to reenter these same key values at a later date to restore master key values that have been cleared. If defining a master key by using a pass phrase, realize that the same pass phrase always produces the same master key values and is therefore as critical and sensitive as the master key values themselves. Make sure that you save the pass phrase so that you can later reenter it if needed. Because of the sensitive nature of the pass phrase, make sure that you secure it in a safe place.

If you are using TKE, proceed to the next step.

Process

Passphrase Initialization to load and SET master keys and initialize CKDS and PKDS

- OR -

Clear Master Key Entry

Note: Using the Coprocessor Management panel, the master keys can be loaded into all the coprocessors at the same time.

- Load DES New Master Key (optional)
- Load RSA New Master Key (optional)
- Load New AES master key if running on IBM z10 or newer servers with a CCA Crypto Express coprocessor and the Nov. 2008 or newer licensed internal code. (optional)
- Load New ECC master key if running on IBM z10 or newer servers with a CCA Crypto Express coprocessor and the Sept. 2011 or newer licensed internal code. (optional)
- Initialize CKDS
- Initialize the PKDS
- Enable PKA Callable Services control

Note: The PKA Callable Services control is disabled if the system has a CEX3C or newer with the Sept. 2011 or newer licensed internal code.

Responsible

ICSF Administrator and Key Officers

Where

ICSF Panels

Verify

In System Log (Systems with CCA Crypto Express coprocessors and accelerators):

```
CSFM608I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM608I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM654I KEY ARCHIVING USE CONTROL IS DISABLED.
CSFM015I FIPS 140 SELF CHECKS FOR PKCS11 SERVICES
SUCCESSFUL.
CSFM111I CRYPTOGRAPHIC FEATURE IS ACTIVE. CRYPTO EXPRESS5 ACCELERATOR 5Axx, SERIAL NUMBER
N/A.
CSFM111I CRYPTOGRAPHIC FEATURE IS ACTIVE. CRYPTO EXPRESS5 COPROCESSOR 5Czz, SERIAL NUMBER
ssssssss.
CSFM133I THERE ARE NO ACTIVE PKCS11 COPROCESSORS.
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.CSFCKDS IS NOT
INITIALIZED.
CSFM101E PKA KEY DATA SET, CSF.CSFCKDS IS NOT
INITIALIZED.
CSFM698I DOMAIN IN USE: 4
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE
AVAILABLE.
CSFM001I ICSF INITIALIZATION COMPLETE
```

Message CSFM111I is issued for each active Crypto Express coprocessor and accelerator.

Message CSFM122I is not issued when your system has any CEX3C coprocessors (with the Sept. 2011 or later LIC) online. The PKA callable services control will not be active. The availability of RSA callable services depend on the status of the RSA master key. CSFM130I is issued when the RSA master key is active and RSA callable services are available.

In System Log (without coprocessors and accelerators):

```
CSFM608I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM608I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM654I KEY ARCHIVING USE CONTROL IS DISABLED.
CSFM015I FIPS 140 SELF CHECKS FOR PKCS11 SERVICES SUCCESSFUL.
CSFM505I CRYPTOGRAPHY - THERE ARE NO ACTIVE CRYPTOGRAPHIC COPROCESSORS.
CSFM133I THERE ARE NO ACTIVE PKCS11 COPROCESSORS.
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.CSFCKDS IS NOT INITIALIZED.
CSFM101E PKA KEY DATA SET, CSF.CSFCKDS IS NOT INITIALIZED.
CSFM507I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC COPROCESSORS ONLINE.
```



```
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.  
CSFM698I DOMAIN IN USE: 4  
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.  
CSFM001I ICSF INITIALIZATION COMPLETE
```

References

For information on using the Pass Phrase Initialization Utility and managing master keys, refer to [z/OS Cryptographic Services ICSF Administrator's Guide](#).

Completed

Step 7. Customizing TKE and loading master keys

If you are not using TKE, proceed to the next step.

Process

TKE Administrator's and Key Officers

- Define host IDs
- Define roles
- Define coprocessor authorities
- Load new DES master key (optional)
- Load new RSA master key (optional)
- Load new AES master key (optional)
- Load new ECC master key if running on IBM z10 or newer servers with a CCA Crypto Express coprocessor and the Sept. 2011 or later licensed internal code. (optional)

Note: If you have more than one coprocessor, repeat the process for each, unless groups have been defined.

Responsible

ICSF Administrator

- Initialize CKDS and SET the DES and AES (if applicable) master keys
- Initialize PKDS and SET the RSA and ECC (if applicable) master keys
- Enable PKA Callable Services control

Note: The PKA Callable Services control is disabled if the system has a CEX3C or newer with the Sept. 2011 or newer licensed internal code.

Where

TKE Workstation and ICSF Panels

Verify

In System Log (Systems with Crypto Express coprocessors and accelerators):

```
CSFM608I A CKDS KEY STORE POLICY IS NOT DEFINED.  
CSFM608I A PKDS KEY STORE POLICY IS NOT DEFINED.  
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.  
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.  
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.  
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.  
CSFM654I KEY ARCHIVING USE CONTROL IS DISABLED.  
CSFM015I FIPS 140 SELF CHECKS FOR PKCS11 SERVICES  
SUCCESSFUL.  
CSFM111I CRYPTOGRAPHIC FEATURE IS ACTIVE. CRYPTO EXPRESS5 ACCELERATOR 5Axx, SERIAL NUMBER  
N/A.  
CSFM111I CRYPTOGRAPHIC FEATURE IS ACTIVE. CRYPTO EXPRESS5 COPROCESSOR 5Czz, SERIAL NUMBER  
ssssssss.  
CSFM133I THERE ARE NO ACTIVE PKCS11 COPROCESSORS.  
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.CSFCKDS IS NOT  
INITIALIZED.  
CSFM101E PKA KEY DATA SET, CSF.CSFCKDS IS NOT  
INITIALIZED.  
CSFM698I DOMAIN IN USE: 4  
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE
```

```
AVAILABLE.  
CSFM001I ICSF INITIALIZATION COMPLETE
```

Message CSFM111I is issued for each active Crypto Express coprocessors and accelerators.

Message CSFM122I will not be issued when your system has any CEX3C or newer coprocessors (with the Sept. 2011 or later LIC) online. The PKA callable services control will not be active. The availability of RSA callable services will depend on the status of the RSA master key. CSFM130I is issued when the RSA master key is active and RSA callable services are available.

In System Log (Systems without coprocessors or accelerators):

```
CSFM608I A CKDS KEY STORE POLICY IS NOT DEFINED.  
CSFM608I A PKDS KEY STORE POLICY IS NOT DEFINED.  
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.  
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.  
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.  
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.  
CSFM654I KEY ARCHIVING USE CONTROL IS DISABLED.  
CSFM015I FIPS 140 SELF CHECKS FOR PKCS11 SERVICES SUCCESSFUL.  
CSFM505I CRYPTOGRAPHY - THERE ARE NO ACTIVE CRYPTOGRAPHIC COPROCESSORS.  
CSFM131E CRYPTOGRAPHY - DES SERVICES ARE NOT AVAILABLE.  
CSFM131E CRYPTOGRAPHY - RSA SERVICES ARE NOT AVAILABLE.  
CSFM131E CRYPTOGRAPHY - ECC SERVICES ARE NOT AVAILABLE.  
CSFM131E CRYPTOGRAPHY - AES SERVICES ARE NOT AVAILABLE.  
CSFM133I THERE ARE NO ACTIVE PKCS11 COPROCESSORS.  
CSFM131E CRYPTOGRAPHY - SECURE KEY PKCS11 SERVICES ARE NOT AVAILABLE.  
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.CSFCKDS IS NOT INITIALIZED.  
CSFM101E PKA KEY DATA SET, CSF.CSFPKDS IS NOT INITIALIZED.  
CSFM507I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC COPROCESSORS ONLINE.  
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.  
CSFM698I DOMAIN IN USE: 4  
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.  
CSFM001I ICSF INITIALIZATION COMPLETE
```

References

For information on managing master keys, refer to [z/OS Cryptographic Services ICSF Administrator's Guide](#).

Completed

Step 8. CICS-ICSF Attachment Facility setup

If you are not using CICS, proceed to the next topic.

Process

Follow the instructions in [Appendix C, "CICS-ICSF Attachment Facility,"](#) on page 485 if desired.

Responsible

System Programmer

Where

Sample Jobs

References

[Appendix C, "CICS-ICSF Attachment Facility,"](#) on page 485

Completed

Step 9. Complete ICSF initialization

See ["Steps for initializing ICSF"](#) on page 33

Responsible

System Programmer

Where

Operator Console

Commonly encountered ICSF first time setup/initialization messages

These ICSF messages are commonly encountered during initialization and first time startup of ICSF.

- **CSFM124I MASTER KEY** *mk* ON *coprocessor-name cii*, **SERIAL NUMBER** *nnnnnnn*, **NOT INITIALIZED**
- The cryptographic coprocessor does not have the master key. When a master key is not set, then the cryptographic coprocessor may not be used for operations with the master key until the system administrator has provided the master key. This may be a normal situation for your installation. Have the system administrator enter the correct master key if appropriate.
- **CSFM410E ERROR IN OPTIONS DATA SET** - ICSF could not interpret the options data set. Check the CSF job output for diagnostic messages.

Appendix E. Using AMS REPRO encryption

This appendix provides information on using IDCAMS REPRO ENCIPHER and DECIPHER options with ICSF.

Steps for setting up ICSF

Perform these tasks to use the ENCIPHER and DECIPHER parameters with ICSF:

1. Define the key value that is used to encrypt and decrypt the data key. To define the key value, use one of these ICSF key administrative options:
 - Trusted Key Entry (TKE) workstation. For information about how to define the key value by using the TKE workstation, see *z/OS Cryptographic Services ICSF TKE Workstation User's Guide*.
 - Key generator utility program (KGUP). Use the KGUP panel ICSF - Create ADD, UPDATE, or DELETE Key Statement to define the key value. For more information about how to use KGUP panels, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

Be aware of the following restrictions:

- The length of the data encryption key is limited to 8 bytes, or 56-bit DES. Triple DES support is not available.
 - Key labels are limited to 8 characters because of the fixed size of REPRO storage areas.
 - The REPRO command's encryption algorithm variables are not documented, so you cannot use them to write decryption applications on another system. Therefore, cross-platform exchange is not possible.
2. Refresh ICSF's cryptographic key data set (CKDS) so that the key value can be used by REPRO.
 3. Ensure that ICSF can support PCF macro calls by specifying COMPAT(YES) in the ICSF installation options. For more information about how to specify ICSF installation options, see [Chapter 2, "Installation, initialization, and customization,"](#) on page 9.

If you had to change the ICSF installation options, you must restart ICSF.

4. Run the REPRO ENCIPHER or DECIPHER job.

Restrictions: The REPRO command's encryption algorithm variables are not documented, so you cannot use them to write decryption applications on another system. Therefore, cross-platform exchange is not possible.

Recommendation: Do not specify the REPRO parameter PRIVATEKEY because it exposes the clear data key value. Instead, specify either EXTERNALKEY or INTERNALKEY, and STOREDATAKEY.

Appendix F. Systems without Cryptographic features

This topic describes the processing of ICSF without a cryptographic coprocessor or accelerator.

Applications and programs

Applications requiring secure cryptography using encrypted keys will not be able to execute without a cryptographic coprocessor or accelerator. All cryptographic keys must be clear keys.

These applications and programs are not supported:

- Access Method Services Cryptographic option.
- CICS attachment facility.
- CKDS Conversion program.
- CSFEUTIL program for CKDS reencipher, refresh, and change master key functions.
- CSFPUTIL program for PKDS reencipher and refresh functions.
- Distributed Key Management System (DKMS).
- Key Generation Utility Program (KGUP) – Clear key can be generated.
- PCF applications.
- UDX (User Defined Extension) support.
- VTAM Session Level Encryption.
- If only the CPACF feature is installed, you will not be able to:
 1. Set master keys.
 2. Initialize the PKDS.
 3. Store keys in the PKDS.

Callable services

These services are available when there are no cryptographic coprocessors or accelerators:

- Character/Nibble Conversion (CSNBXBC and CSNBXCB)
- Code Conversion (CSNBXEA and CSNBXAE)
- Control Vector Generate (CSNBCVG)
- Decode (CSNBDCO)
- Encode (CSNBECO)
- Field level decipher (CSNBFLD)
- Field level encipher (CSNBFLE)
- ICSF Query Facility (CSFIQF and CSFIQF6) - The only rule available without a coprocessor is ICSFSTAT.
- ICSF Query Facility2 (CSFIQF2 and CSFIQF26)
- ICSF Query Algorithm (CSFIQA)
- MDC Generate (CSNBMDG and CSNBMDG1)
- One-Way Hash Generate (CSNBOWH and CSNBOWH1)
- PKA Key Token Build (CSNDPKB)
- PKA Public Key Extract (CSNDPKX)
- PKCS #11 Derive multiple keys (CSFPDMK)
- PKCS #11 Derive key (CSFPDVK)

- PKCS #11 Get attribute value (CSFPGAV)
- PKCS #11 Generate key pair (CSFPGKP)
- PKCS #11 Generate secret key (CSFPGSK)
- PKCS #11 Generate MAC (CSFPHMG)
- PKCS #11 Verify MAC (CSFPHMV)
- PKCS #11 One-way hash generate (CSFPOWH)
- PKCS #11 Private key sign (CSFPPKS)
- PKCS #11 Public key verify (CSFPPKV)
- PKCS #11 Pseudo-random function (CSFPPRF)
- PKCS #11 Set attribute value (CSFPSAV)
- PKCS #11 Secret key decrypt (CSFPSKD)
- PKCS #11 Secret key encrypt (CSFPSKE)
- PKCS #11 Token record create (CSFPTRC)
- PKCS #11 Token record delete (CSFPTRD)
- PKCS #11 Token record list (CSFPTRL)
- PKCS #11 Unwrap key (CSFPUWK)
- PKCS #11 Wrap key (CSFPWPK)
- Random Number Generate (CSNBRNG) and Random Number Generate Long (CSNBRNGL)
- Symmetric Key Decipher (CSNBSYD and CSNBSYD1) - Only clear keys are supported.
- Symmetric Key Encipher (CSNBSYE and CSNBSYE1) - Only clear keys are supported.
- Symmetric MAC Generate (CSNBSMG, CSNBSMG1, CSNESMG, and CSNESMG1)
- Symmetric MAC Verify (CSNBSMV, CSNBSMV1, CSNESMV, and CSNESMV1)
- X9.9 Data Editing (CSNB9ED)

These services are available when there are no cryptographic coprocessors and there are accelerators:

- Digital Signature Verify (CSNDDSV)
- PKA Decrypt (CSNDPKD)
- PKA Encrypt (CSNDPKE) ZERO-PAD formatting only

Note:

1. Installation defined callable services are supported only if you're using clear keys and using one of the supported callable services.
2. If running without an active PKCS #11 Cryptographic coprocessor, the PKCS #11 callable services are limited to clear keys only.

ICSF setup and initialization

If starting ICSF without any cryptographic features:

```
CSFM608I A CKDS KEY STORE POLICY IS DEFINED.
CSFM608I A PKDS KEY STORE POLICY IS DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM654I KEY ARCHIVING USE CONTROL IS DISABLED.
CSFM015I FIPS 140 SELF CHECKS FOR PKCS11 SERVICES SUCCESSFUL.
CSFM505I CRYPTOGRAPHY - THERE ARE NO ACTIVE CRYPTOGRAPHIC COPROCESSORS.
CSFM133I THERE ARE NO ACTIVE PKCS11 COPROCESSORS.
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.CKDS IS NOT INITIALIZED.
CSFM101E PKA KEY DATA SET, CSF.PKDS IS NOT INITIALIZED.
CSFM507I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC COPROCESSORS ONLINE.
```



```
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.  
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.  
CSFM001I ICSF INITIALIZATION COMPLETE
```

If starting ICSF with a cryptographic accelerator:

```
CSFM608I A CKDS KEY STORE POLICY IS DEFINED.  
CSFM608I A PKDS KEY STORE POLICY IS DEFINED.  
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.  
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.  
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.  
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.  
CSFM654I KEY ARCHIVING USE CONTROL IS DISABLED.  
CSFM015I FIPS 140 SELF CHECKS FOR PKCS11 SERVICES SUCCESSFUL.  
CSFM111I CRYPTOGRAPHIC FEATURE IS ACTIVE. CRYPTO EXPRESS4 ACCELERATOR 4A00,  
SERIAL NUMBER N/A.  
CSFM505I CRYPTOGRAPHY - THERE ARE NO ACTIVE CRYPTOGRAPHIC COPROCESSORS.  
CSFM133I THERE ARE NO ACTIVE PKCS11 COPROCESSORS.  
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.CKDS IS NOT INITIALIZED.  
CSFM101E PKA KEY DATA SET, CSF.PKDS IS NOT INITIALIZED.  
CSFM507I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC COPROCESSORS ONLINE.  
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.  
CSFM001I ICSF INITIALIZATION COMPLETE
```

Secure Sockets Layer (SSL)

System SSL applications are supported. SSL defines methods for data encryption, server authentication, message integrity, and client authentication for a TCP/IP connection. Security is provided on the link and callable services have been enhanced for DES, TDES and SHA-1 services.

TKE workstation

The Trusted Key Entry (TKE) workstation is not available with this hardware configuration.

Migrating from no active coprocessors to an active coprocessor environment

To migrate from an environment with no active coprocessors to an active coprocessor environment:

1. Add the DOMAIN parameter for the LPAR and configure the card as either a coprocessor or an accelerator, depending on the functionality you would like to exploit from your crypto card. For more information, see “Checklist for first-time startup of ICSF” on page 489.
2. If you currently have the ICSF started task configured and active, you have a clear-key CKDS to store any DES or AES data keys. Once you add/activate your crypto coprocessor, this clear-key CKDS is incompatible with your ICSF environment.

If you do not have keys stored in your current clear-key CKDS that you wish to carry forward, delete the clear-key CKDS and start with a new empty CKDS to be used for storing secure keys.

If you do have keys stored in your current clear-key CKDS that you wish to carry forward, there are two options, both of which have advantages and disadvantages.

- a. Use an authorized program to invoke ICSF callable services to read those keys you wish to transfer using the CKDS Key Record Read2 (CSNBKRR2 and CSNEKRR2) service and store them into a file or data set. You will write them to the new secure key CKDS after step 4.

Advantages:

Uses official ICSF callable services to manipulate key material, avoiding any risk of damaging the new CKDS.

Disadvantages:

Requires writing a program that runs in supervisor state or system key.

The key material is in the clear when stored in the external file or data set which requires the same protections as the CKDS itself.

- b. Create a new CKDS with the same LRECL as the current CKDS so you can use IDCAMS REPRO to copy records directly from the old CKDS to the new one.

Advantages:

No intermediate file or data set is required.

No requirement to write applications for this method.

Disadvantages:

If the new CKDS is not the same LRECL, there is a risk of making the new CKDS unusable.

3. Create a PKDS dataset and optionally, a TKDS dataset. For more information, see [“Steps to start ICSF for the first time”](#) on page 32.
4. Load master keys in the coprocessor for use with the CKDS and PKDS. For more information, see 'Managing CCA Master Keys' in [z/OS Cryptographic Services ICSF Administrator's Guide](#) and then initialize your newly created CKDS and PKDS with those master keys. For more information, see 'Initializing the key data sets at first-time startup' in [z/OS Cryptographic Services ICSF Administrator's Guide](#).
5. Restore the clear keys you want to preserve.
 - a. Use an authorized program to invoke ICSF callable services to restore those keys you want to transfer by reading them from the secure file created in step 2a using the CKDS Key Record Create2 (CSNBKRC2 and CSNEKRC2) service.
 - b. Use IDCAMS REPRO to copy the records directly from the old CKDS to the new CKDS.

Appendix G. Resource names for CCA and ICSF entry points

Table 173. Resource names for CCA and ICSF entry points						
Descriptive service name	CCA entry point names		ICSF entry point names		SAF resource name	Callable service exit name
	31-bit	64-bit	31-bit	64-bit		
Authentication Parameter Generate	CSNBAPG	CSNEAPG	CSFAPG	CSFAPG6	CSFAPG	CSFAPG
Cipher Text Translate2	CSNBCTT2	CSNECTT2	CSFCTT2	CSFCTT26	CSFCTT2	CSFCTT2
Cipher Text Translate2	CSNBCTT3	CSNECTT3	CSFCTT3	CSFCTT36	CSFCTT3	CSFCTT3
CKDS Key Record Create	CSNBKRC	CSNEKRC	CSFKRC	CSFKRC6	CSFKRC	CSFKRC
CKDS Key Record Create2	CSNBKRC2	CSNEKRC2	CSFKRC2	CSFKRC26	CSFKRC2	CSFKRC2
CKDS Key Record Delete	CSNBKRD	CSNEKRD	CSFKRD	CSFKRD6	CSFKRD	CSFKRD
CKDS Key Record Read	CSNBKRR	CSNEKRR	CSFKRR	CSFKRR6	CSFKRR	CSFKRR
CKDS Key Record Read2	CSNBKRR2	CSNEKRR2	CSFKRR2	CSFKRR26	CSFKRR2	CSFKRR2
CKDS Key Record Write	CSNBKRW	CSNEKRW	CSFKRW	CSFKRW6	CSFKRW	CSFKRW
CKDS Key Record Write2	CSNBKRW2	CSNEKRW2	CSFKRW2	CSFKRW26	CSFKRW2	CSFKRW2
Clear Key Import	CSNBCKI	CSNECKI	CSFCKI	CSFCKI6	CSFCKI	CSFCKI
Clear PIN Encrypt	CSNBCPE	CSNECPE	CSFCPE	CSFCPE6	CSFCPE	CSFCPE
Clear PIN Generate	CSNBPGN	CSNEPGN	CSFPGN	CSFPGN6	CSFPGN	CSFPGN
Clear PIN Generate Alternate	CSNBCPA	CSNECPA	CSFCPA	CSFCPA6	CSFCPA	CSFCPA
Control Vector Generate	CSNBCVG	CSNECVG	CSFCVG	CSFCVG6	N/A	N/A
Control Vector Translate	CSNBCVT	CSNECVT	CSFCVT	CSFCVT6	CSFCVT	CSFCVT
Coordinated KDS Administration	N/A	N/A	CSFCRC	CSFCRC6	CSFCRC	N/A
Cryptographic Usage Statistic	N/A	N/A	CSFSTAT	CSFSTAT6	N/A	N/A
Cryptographic Variable Encipher	CSNBCVE	CSNECVE	CSFCVE	CSFCVE6	CSFCVE	CSFCVE

Table 173. Resource names for CCA and ICSF entry points (continued)

Descriptive service name	CCA entry point names		ICSF entry point names		SAF resource name	Callable service exit name
CVV Key Combine	CSNBCKC	CSNECKC	CSFCKC	CSFCKC6	CSFCKC	CSFCKC
Data Key Export	CSNBDKX	CSNEDKX	CSFDKX	CSFDKX6	CSFDKX	CSFDKX
Data Key Import	CSNBDKM	CSNEDKM	CSFDKM	CSFDKM6	CSFDKM	CSFDKM
Decipher	CSNBDEC	CSNEDEC	CSFDEC	CSFDEC6	CSFDEC	CSFDEC
Decipher	CSNBDEC1	CSNEDEC1	CSFDEC1	CSFDEC16	CSFDEC1	CSFDEC1
Decode	CSNBDCO	CSNEDCO	CSFDCO	CSFDCO6	CSFDCO	CSFDCO
Derive ICC MK	CSNBDCM	CSNEDCM	CSFDCM	CSFDCM6	CSFDCM	CSFDCM
Derive Session Key	CSNBDSK	CSNEDSK	CSFDSK	CSFDSK6	CSFDSK	CSFDSK
Digital Signature Generate	CSNDDSG	CSNFDSG	CSFDSG	CSFDSG6	CSFDSG	CSFDSG
Digital Signature Verify	CSNDDSV	CSNFDSV	CSFDSV	CSFDSV6	CSFDSV	CSFDSV
Diversified Key Generate	CSNBDKG	CSNEDKG	CSFDKG	CSFDKG6	CSFDKG	CSFDKG
Diversified Key Generate2	CSNBDKG2	CSNEDKG2	CSFDKG2	CSFDKG26	CSFDKG2	CSFDKG2
Diversify Directed Key	CSNBDDK	CSNEDDK	CSFDDK	CSFDDK6	CSFDDK	CSFDDK
DK Deterministic PIN Generate	CSNBDDPG	CSNEDDPG	CSFDDPG	CSFDDPG6	CSFDDPG	CSFDDPG
DK Migrate PIN	CSNBDMPP	CSNEDMP	CSFDMP	CSFDMP6	CSFDMP	CSFDMP
DK PAN Modify in Transaction	CSNBDMPT	CSNEDPMT	CSFDPMT	CSFDPMT6	CSFDPMT	CSFDPMT
DK PAN Translate	CSNBDMPT	CSNEDPT	CSFDPT	CSFDPT6	CSFDPT	CSFDPT
DK PIN Change	CSNBDMPC	CSNEDPC	CSFDPC	CSFDPC6	CSFDPC	CSFDPC
DK PIN Verify	CSNBDMPV	CSNEDPV	CSFDPV	CSFDPV6	CSFDPV	CSFDPV
DK PRW Card Number Update	CSNBDMNU	CSNEDPNU	CSFDPNU	CSFDPNU6	CSFDPNU	CSFDPNU
DK PRW Card Number Update2	CSNBDMCU2	CSNBECU2	CSFDCU2	CSFDCU26	CSFDCU2	CSFDCU2
DK PRW CMAC Generate	CSNBDMPCG	CSNEDPCG	CSFDPCG	CSFDPCG6	CSFDPCG	CSFDPCG
DK Random PIN Generate	CSNBDRPG	CSNEDRPG	CSFDRPG	CSFDRPG6	CSFDRPG	CSFDRPG
DK Random PIN Generate2	CSNBDRG2	CSNBERG2	CSFDRG2	CSFDRG26	CSFDRG2	CSFDRG2
DK Regenerate PRW	CSNBDRP	CSNEDRP	CSFDRP	CSFDRP6	CSFDRP	CSFDRP
ECC Diffie-Hellman	CSNDEDH	CSNFEDH	CSFEDH	CSFEDH6	CSFEDH	CSFEDH

Table 173. Resource names for CCA and ICSF entry points (continued)

Descriptive service name	CCA entry point names		ICSF entry point names		SAF resource name	Callable service exit name
EMV Scripting Service	CSNBESC	CSNEESC	CSFESC	CSFESC6	CSFESC	CSFESC
EMV Transaction (ARQC/ARPC) Service	CSNBEAC	CSNEEAC	CSFEAC	CSFEAC6	CSFEAC	CSFEAC
EMV Verification Functions	CSNBEVF	CSNEEVF	CSFEVF	CSFEVF6	CSFEVF	CSFEVF
Encipher	CSNBENC	CSNEENC	CSFENC	CSFENC6	CSFENC	CSFENC
Encipher	CSNBENC1	CSNEENC1	CSFENC1	CSFENC16	CSFENC1	CSFENC1
Encode	CSNBECO	CSNEECO	CSFECO	CSFECO6	CSFECO	CSFECO
Encrypted PIN Generate	CSNBEPG	CSNEEPG	CSFEPG	CSFEPG6	CSFEPG	CSFEPG
Encrypted PIN Translate	CSNBPTR	CSNEPTR	CSFPTR	CSFPTR6	CSFPTR	CSFPTR
Encrypted PIN Translate2	CSNBPTR2	CSNEPTR2	CSFPTR2	CSFPTR26	CSFPTR2	CSFPTR2
Encrypted PIN Translate Enhanced	CSNBPTRE	CSNEPTRE	CSFPTR	CSFPTR6	CSFPTR	CSFPTR
Encrypted PIN Verify	CSNBPVR	CSNEPVR	CSFPVR	CSFPVR6	CSFPVR	CSFPVR
Encrypted PIN Verify2	CSNBPVR2	CSNEPVR2	CSFPVR2	CSFPVR26	CSFPVR2	CSNBPVR2
Field Level Decipher	CSNBFLD	CSNEFLD	CSFFLD	CSFFLD6	N/A	N/A
Field Level Encipher	CSNBFLE	CSNEFLE	CSFFLE	CSFFLE6	N/A	N/A
Format Preserving Algorithms Decipher	CSNBFFXD	CSNEFFXD	CSFFFXD	CSFFFXD6	CSFFFXD	CSFFFXD
Format Preserving Algorithms Encipher	CSNBFFXE	CSNEFFXE	CSFFFXE	CSFFFXE6	CSFFFXE	CSFFFXE
Format Preserving Algorithms Translate	CSNBFFXT	CSNEFFXT	CSFFFXT	CSFFFXT6	CSFFFXT	CSFFFXT
FPE Decipher	CSNBFPED	CSNEFPED	CSFFPED	CSFFPED6	CSFFPED	CSFFPED
FPE Encipher	CSNBFPEE	CSNEFPEE	CSFFPEE	CSFFPEE6	CSFFPEE	CSFFPEE
FPE Translate	CSNBFPET	CSNEFPET	CSFFPET	CSFFPET6	CSFFPET	CSFFPET
Generate Issuer MK	CSNBGIM	CSNEGIM	CSFGIM	CSFGIM6	CSFGIM	CSFGIM
HMAC Generate	CSNBHMG	CSNEHMG	CSFHMG	CSFHMG6	CSFHMG	CSFHMG
HMAC Generate	CSNBHMG1	CSNEHMG1	CSFHMG1	CSFHMG16	CSFHMG1	CSFHMG1
HMAC Verify	CSNBHMV	CSNEHMGV	CSFHMGV	CSFHMGV6	CSFHMGV	CSFHMGV
HMAC Verify	CSNBHMGV1	CSNEHMGV1	CSFHMGV1	CSFHMGV16	CSFHMGV1	CSFHMGV1
ICSF Multi-Purpose Service	N/A	N/A	CSFMPS	CSFMPS6	CSFMPS	CSFMPS
ICSF Query Algorithm	N/A	N/A	CSFIQA	CSFIQA6	CSFIQA	N/A

Table 173. Resource names for CCA and ICSF entry points (continued)

Descriptive service name	CCA entry point names		ICSF entry point names		SAF resource name	Callable service exit name
ICSF Query Facility	N/A	N/A	CSFIQF	CSFIQF6	CSFIQF	N/A
ICSF Query Facility2	N/A	N/A	CSFIQF2	CSFIQF26	N/A	CSFIQF2
Key Data Set List	N/A	N/A	CSFKDSL	CSFKDSL6	CSFKDSL	CSFKDSL
Key Data Set Metadata Read	N/A	N/A	CSFKDMR	CSFKDMR6	CSFKDMR	CSFKDMR
Key Data Set Metadata Write	N/A	N/A	CSFKDMW	CSFKDMW6	CSFKDMW	CSFKDMW
Key Data Set Record Retrieve	N/A	N/A	CSFRRT	CSFRRT6	CSFRRT (see notes)	N/A
Key Data Set Update	N/A	N/A	CSFKDU	CSFKDU6	CSFKDU (see notes)	N/A
Key Encryption Translate	CSNBKET	CSNEKET	CSFKET	CSFKET6	CSFKET	CSFKET
Key Export	CSNBKEX	CSNEKEX	CSFKEX	CSFKEX6	CSFKEX	CSFKEX
Key Generate	CSNBKGN	CSNEKGN	CSFKGN	CSFKGN6	CSFKGN	CSFKGN
Key Generate2	CSNBKGN2	CSNEKGN2	CSFKGN2	CSFKGN26	CSFKGN2	CSFKGN2
Key Import	CSNBKIM	CSNEKIM	CSFKIM	CSFKIM6	CSFKIM	CSFKIM
Key Part Import	CSNBKPI	CSNEKPI	CSFKPI	CSFKPI6	CSFKPI	CSFKPI
Key Part Import2	CSNBKPI2	CSNEKPI2	CSFKPI2	CSFKPI26	CSFKPI2	CSFKPI2
Key Test	CSNBKYT	CSNEKYT	CSFKYT	CSFKYT6	CSFKYT	CSFKYT
Key Test2	CSNBKYT2	CSNEKYT2	CSFKYT2	CSFKYT26	CSFKYT2	CSFKYT2
Key Test Extended	CSNBKYTX	CSNEKYTX	CSFKYTX	CSFKYTX6	CSFKYTX	CSFKYTX
Key Token Build	CSNBKTB	CSNEKTB	CSFKTB	CSFKTB6	N/A	N/A
Key Token Build2	CSNBKTB2	CSNEKTB2	CSFKTB2	CSFKTB26	N/A	N/A
Key Token Wrap	N/A	N/A	CSFWRP	CSFWRP6	CSFWRP	N/A
Key Translate	CSNBKTR	CSNEKTR	CSFKTR	CSFKTR6	CSFKTR	CSFKTR
Key Translate2	CSNBKTR2	CSNEKTR2	CSFKTR2	CSFKTR26	CSFKTR2	CSFKTR2
MAC Generate	CSNBMGN	CSNEMGN	CSFMGN	CSFMGN6	CSFMGN	CSFMGN
MAC Generate	CSNBMGN1	CSNEMGN1	CSFMGN1	CSFMGN16	CSFMGN1	CSFMGN1
MAC Generate2	CSNBMGN2	CSNEMGN2	CSFMGN2	CSFMGN26	CSFMGN2	CSFMGN2
MAC Generate2	CSNBMGN3	CSNEMGN3	CSFMGN3	CSFMGN36	CSFMGN3	CSFMGN3
MAC Verify	CSNBMVR	CSNEMVR	CSFMVR	CSFMVR6	CSFMVR	CSFMVR
MAC Verify	CSNBMVR1	CSNEMVR1	CSFMVR1	CSFMVR16	CSFMVR1	CSFMVR1
MAC Verify2	CSNBMVR2	CSNEMVR2	CSFMVR2	CSFMVR26	CSFMVR2	CSFMVR2
MAC Verify2	CSNBMVR3	CSNEMVR3	CSFMVR3	CSFMVR36	CSFMVR3	CSFMVR3

Table 173. Resource names for CCA and ICSF entry points (continued)

Descriptive service name	CCA entry point names		ICSF entry point names		SAF resource name	Callable service exit name
MDC Generate	CSNBMDG	CSNEMDG	CSFMDG	CSFMDG6	CSFMDG	CSFMDG
MDC Generate	CSNBMDG1	CSNEMDG1	CSFMDG1	CSFMDG16	CSFMDG1	CSFMDG1
Multi-MAC Scheme	CSNBMMS	CSNEMMS	CSFMMS	CSFMMS6	CSFMMS	CSFMMS
Multiple Clear Key Import	CSNBCKM	CSNECKM	CSFCKM	CSFCKM6	CSFCKM	CSFCKM
Multiple Secure Key Import	CSNBSKM	CSNESKM	CSFSKM	CSFSKM6	CSFSKM	CSFSKM
One-Way Hash Generate	CSNBOWH	CSNEOWH	CSFOWH	CSFOWH6	CSFOWH	CSFOWH
One-Way Hash Generate	CSNBOWH1	CSNEOWH1	CSFOWH1	CSFOWH16	CSFOWH1	CSFOWH1
PCI Interface	N/A	N/A	CSFPCI	CSFPCI6	CSFPCI	CSFPCI
PIN Change/Unblock	CSNBPCU	CSNEPCU	CSFPCU	CSFPCU6	CSFPCU	CSFPCU
PKA Decrypt	CSNDPKD	CSNFPKD	CSFPKD	CSFPKD6	CSFPKD	CSFPKD
PKA Encrypt	CSNDPKE	CSNFPKE	CSFPKE	CSFPKE6	CSFPKE	CSFPKE
PKA Key Generate	CSNDPKG	CSNFPKG	CSFPKG	CSFPKG6	CSFPKG	CSFPKG
PKA Key Import	CSNDPKI	CSNFPKI	CSFPKI	CSFPKI6	CSFPKI	CSFPKI
PKA Key Token Build	CSNDPKB	CSNFPKB	CSFPKB	CSFPKB6	N/A	N/A
PKA Key Token Change	CSNDKTC	CSNFKTC	CSFPKTC	CSFPKTC6	CSFPKTC	CSFPKTC
PKA Key Translate	CSNDPKT	CSNFPKT	CSFPKT	CSFPKT6	CSFPKT	CSFPKT
PKA Public Key Extract	CSNDPKX	CSNFPKX	CSFPKX	CSFPKX6	CSFPKX	CSFPKX
PKCS #11 Derive Key	N/A	N/A	CSFPDVK	CSFPDVK6	CSF1DVK ¹	N/A
PKCS #11 Derive Multiple Keys	N/A	N/A	CSFPDMK	CSFPDMK6	CSF1DMK ¹	N/A
PKCS #11 Generate Keyed MAC	N/A	N/A	CSFPHMG	CSFPHMG6	CSF1HMG ¹	N/A
PKCS #11 Generate Key Pair	N/A	N/A	CSFPGKP	CSFPGKP6	CSF1GKP ¹	N/A
PKCS #11 Generate Secret Key	N/A	N/A	CSFPGSK	CSFPGSK6	CSF1GSK ¹	N/A
PKCS #11 Get Attribute Value	N/A	N/A	CSFPGAV	CSFPGAV6	CSF1GAV ¹	N/A
PKCS #11 One-Way Hash, Sign, or Verify	N/A	N/A	CSFPOWH	CSFPOWH6	CSFOWH	N/A
PKCS #11 Private Key Sign	N/A	N/A	CSFPPKS	CSFPPKS6	CSF1PKS ¹	N/A

Table 173. Resource names for CCA and ICSF entry points (continued)

Descriptive service name	CCA entry point names		ICSF entry point names		SAF resource name	Callable service exit name
PKCS #11 Private Key Structure Decrypt	N/A	N/A	CSFPPD2	CSFPPD26	CSFPKD	N/A
PKCS #11 Private Key Structure Sign	N/A	N/A	CSFPPS2	CSFPPS26	CSFDSG	N/A
PKCS #11 Pseudo-Random Function	N/A	N/A	CSFPPRF	CSFPPRF6	CSFRNG	N/A
PKCS #11 Public Key Structure Encrypt	N/A	N/A	CSFPPE2	CSFPPE26	CSFPKE	N/A
PKCS #11 Public Key Structure Verify	N/A	N/A	CSFPPV2	CSFPPV26	CSFDSV	N/A
PKCS #11 Public Key Verify	N/A	N/A	CSFPPKV	CSFPPKV6	CSF1PKV ¹	N/A
PKCS #11 Secret Key Decrypt	N/A	N/A	CSFPSKD	CSFPSKD6	CSF1SKD ¹	N/A
PKCS #11 Secret Key Encrypt	N/A	N/A	CSFPSKE	CSFPSKE6	CSF1SKE ¹	N/A
PKCS #11 Secret Key Reencrypt	N/A	N/A	CSFPSKR	CSFPSKR6	CSF1SKR ¹	N/A
PKCS #11 Set Attribute Value	N/A	N/A	CSFPSAV	CSFPSAV6	CSF1SAV ¹	N/A
PKCS #11 Token Record Create	N/A	N/A	CSFPTRC	CSFPTRC6	CSF1TRC ¹	N/A
PKCS #11 Token Record Delete	N/A	N/A	CSFPTRD	CSFPTRD6	CSF1TRD ¹	N/A
PKCS #11 Token Record List	N/A	N/A	CSFPTRL	CSFPTRL6	CSF1TRL ¹	N/A
PKCS #11 Unwrap Key	N/A	N/A	CSFPUWK	CSFPUWK6	CSF1UWK ¹	N/A
PKCS #11 Verify Keyed MAC	N/A	N/A	CSFPHMV	CSFPHMV6	CSF1HMV ¹	N/A
PKCS #11 Wrap Key	N/A	N/A	CSFPWPK	CSFPWPK6	CSF1WPK ¹	N/A
PKDS Key Record Create	CSNDKRC	CSNFKRC	CSFPKRC	CSFPKRC6	CSFPKRC	CSFPKRC
PKDS Key Record Delete	CSNDKRD	CSNFKRD	CSFPKRD	CSFPKRD6	CSFPKRD	CSFPKRD
PKDS Key Record Read	CSNDKRR	CSNFKRR	CSFPKRR	CSFPKRR6	CSFPKRR	CSFPKRR
PKDS Key Record Read2	CSNDKRR2	CSNFKRR2	CSFPRR2	CSFPRR26	CSFPRR2	CSFPRR2

Table 173. Resource names for CCA and ICSF entry points (continued)

Descriptive service name	CCA entry point names		ICSF entry point names		SAF resource name	Callable service exit name
PKDS Key Record Write	CSNDKRW	CSNFKRW	CSFPKRW	CSFPKRW6	CSFPKRW	CSFPKRW
Prohibit Export	CSNBPEX	CSNEPEX	CSFPEX	CSFPEX6	CSFPEX	CSFPEX
Prohibit Export Extended	CSNBPEXX	CSNEPEXX	CSFPEXX	CSFPEXX6	CSFPEXX	CSFPEXX
Public Infrastructure Certificate	CSNDPIC	CSNFPIC	CSFPIC	CSFPIC6	CSFPIC	CSFPIC
Random Number Generate	CSNBRNG	CSNERNG	CSFRNG	CSFRNG6	CSFRNG	CSFRNG
Random Number Generate	CSNBRNGL	CSNERNGL	CSFRNGL	CSFRNGL6	CSFRNGL	CSFRNGL
Recover PIN from Offset	CSNBPFO	CSNEPFO	CSFPFO	CSFPFO6	CSFPFO	CSFPFO
Remote Key Export	CSNDRKX	CSNFRKX	CSFRKX	CSFRKX6	CSFRKX	CSFRKX
Restrict Key Attribute	CSNBRKA	CSNERKA	CSFRKA	CSFRKA6	CSFRKA	CSFRKA
Retained Key Delete	CSNDRKD	CSNFRKD	CSFRKD	CSFRKD6	CSFRKD	CSFRKD
Retained Key List	CSNDRKL	CSNFRKL	CSFRKL	CSFRKL6	CSFRKL	CSFRKL
SAF ACEE Selection	N/A	N/A	CSFACEE	CSFACEE6	N/A (see notes)	N/A (see notes)
Secure Key Import	CSNBSKI	CSNESKI	CSFSKI	CSFSKI6	CSFSKI	CSFSKI
Secure Key Import2	CSNBSKI2	CSNESKI2	CSFSKI2	CSFSKI26	CSFSKI2	CSFSKI2
Secure Messaging for Keys	CSNBSKY	CSNESKY	CSFSKY	CSFSKY6	CSFSKY	CSFSKY
Secure Messaging for PINs	CSNBSPN	CSNESPN	CSFSPN	CSFSPN6	CSFSPN	CSFSPN
SET Block Compose	CSNDSBC	CSNFSBC	CSFSBC	CSFSBC6	CSFSBC	CSFSBC
SET Block Decompose	CSNDSBD	CSNFSBD	CSFSBD	CSFSBD6	CSFSBD	CSFSBD
Symmetric Algorithm Decipher	CSNBSAD	CSNESAD	CSFSAD	CSFSAD6	CSFSAD	N/A
Symmetric Algorithm Decipher	CSNBSAD1	CSNESAD1	CSFSAD1	CSFSAD16	CSFSAD1	N/A
Symmetric Algorithm Encipher	CSNBSAE	CSNESAE	CSFSAE	CSFSAE6	CSFSAE	N/A
Symmetric Algorithm Encipher	CSNBSAE1	CSNESAE1	CSFSAE1	CSFSAE16	CSFSAE1	N/A
Symmetric Key Decipher	CSNBSYD	CSNESYD	CSFSYD	CSFSYD6	N/A	N/A

Table 173. Resource names for CCA and ICSF entry points (continued)

Descriptive service name	CCA entry point names		ICSF entry point names		SAF resource name	Callable service exit name
Symmetric Key Decipher	CSNBSYD1	CSNESYD1	CSFSYD1	CSFSYD16	N/A	N/A
Symmetric Key Encipher	CSNBSYE	CSNESYE	CSFSYE	CSFSYE6	N/A	N/A
Symmetric Key Encipher	CSNBSYE1	CSNESYE1	CSFSYE1	CSFSYE16	N/A	N/A
Symmetric Key Export	CSNDSYX	CSNFSYX	CSFSYX	CSFSYX6	CSFSYX	CSFSYX
Symmetric Key Export with Data	CSNDSXD	CSNFSXD	CSFSXD	CSFSXD6	CSFSXD	CSFSXD
Symmetric Key Generate	CSNDSYG	CSNFSYG	CSFSYG	CSFSYG6	CSFSYG	CSFSYG
Symmetric Key Import	CSNDSYI	CSNFSYI	CSFSYI	CSFSYI6	CSFSYI	CSFSYI
Symmetric Key Import2	CSNDSYI2	CSNFSYI2	CSFSYI2	CSFSYI26	CSFSYI2	CSFSYI2
Symmetric MAC Generate	CSNBSMG	CSNESMG	CSFSMG	CSFSMG6	N/A	CSFSMG
Symmetric MAC Generate	CSNBSMG1	CSNESMG1	CSFSMG1	CSFSMG16	N/A	CSFSMG1
Symmetric MAC Verify	CSNBSMV	CSNESMV	CSFSMV	CSFSMV6	N/A	CSFSMV
Symmetric MAC Verify	CSNBSMV1	CSNESMV1	CSFSMV1	CSFSMV16	N/A	CSFSMV1
TR-31 Create	CSNBT31C	CSNET31C	CSFT31C	CSFT31C6	CSFT31C	CSFT31C
TR-31 Import	CSNBT31I	CSNET31I	CSFT31I	CSFT31I6	CSFT31I	CSFT31I
TR-31 Optional Data Build	CSNBT31O	CSNET31O	CSFT31O	CSFT31O6	N/A	N/A
TR-31 Optional Data Read	CSNBT31R	CSNET31R	CSFT31R	CSFT31R6	N/A	N/A
TR-31 Parse	CSNBT31P	CSNET31P	CSFT31P	CSFT31P6	N/A	N/A
TR-31 Translate	CSNBT31X	CSNET31X	CSFT31X	CSFT31X6	CSFT31X	CSFT31X
TR-34 Bind-Begin	CSNDT34B	CSNFT34B	CSFT34B	CSFT34B6	CSFT34B	CSFT34B
TR-34 Bind-Complete	CSNDT34C	CSNFT34C	CSFT34C	CSFT34C6	CSFT34C	CSFT34C
TR-34 Key Distribution	CSNDT34D	CSNFT34D	CSFT34D	CSFT34D6	CSFT34D	CSFT34D
TR-34 Key Receive	CSNDT34R	CSNFT34R	CSFT34R	CSFT34R6	CSFT34R	CSFT34R
Transaction Validation	CSNBTRV	CSNETRV	CSFTRV	CSFTRV6	CSFTRV	CSFTRV
Trusted Block Create	CSNDTBC	CSNFTBC	CSFTBC	CSFTBC6	CSFTBC	CSFTBC
Unique Key Derive	CSNBUKD	CSNEUKD	CSFUKD	CSFUKD6	CSFUKD	CSFUKD

Table 173. Resource names for CCA and ICSF entry points (continued)						
Descriptive service name	CCA entry point names		ICSF entry point names		SAF resource name	Callable service exit name
VISA CVV Service Generate	CSNBCSG	CSNECSG	CSFCSG	CSFCSG6	CSFCSG	CSFCSG
VISA CVV Service Verify	CSNBCSV	CSNECSV	CSFCSV	CSFCSV6	CSFCSV	CSFCSV

Notes:

- Key Data Set Update (CSFKDU and CSFKDU6) and Key Data Set Record Retrieve (CSFRRT and CSFRRT6) will only be granted access with an explicitly defined covering profile.
- SAF ACEE Selection (CSFACEE and CSFACEE6) does not have SAF checking or callable service exit support on its own. The service specified in the *service_name* parameter determines SAF checking and callable service exit capability.
- N/A is shown in a column when the callable service:
 - Does not have CCA entry points (CCA entry point names columns).
 - Does not call SAF to determine access to a CSFSERV resource (SAF resource name column).
 - Does not allow a callable service exit to be defined (Callable service exit name column).
- ¹ CSF1xxx is just another name for the CSFPxxx service.

Appendix H. Accessibility

Accessible publications for this product are offered through [IBM Documentation for z/OS \(www.ibm.com/docs/en/zos\)](http://www.ibm.com/docs/en/zos).

If you experience difficulty with the accessibility of any z/OS documentation see [How to Send Feedback to IBM](#) to leave documentation feedback.

Notices

This information was developed for products and services that are offered in the USA or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America*

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan*

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for IBM Documentation. Use of hyperlinks in other output formats of this information is at your own risk.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

*IBM Corporation
Site Counsel
2455 South Road*

Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or

reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at ibm.com/privacy and IBM's Online Privacy Statement at ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMSdfp, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those

products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: [IBM Lifecycle Support for z/OS \(www.ibm.com/software/support/systemsz/lifecycle\)](http://www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at [Copyright and Trademark information \(www.ibm.com/legal/copytrade.shtml\)](http://www.ibm.com/legal/copytrade.shtml).

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Index

A

- abends [158](#)
- access control checking
 - udx [215](#)
- Access Method Services Cryptographic Option and ICSF [145](#)
- accessibility
 - contact IBM [513](#)
- active coprocessor environment [501](#)
- activity report
 - defining on a DD statement [238](#)
 - description [238](#)
- addressing mode
 - no restrictions on ICSF's caller [144](#)
- AMS DEFINE CLUSTER command [13](#), [17](#), [21](#)
- AMS IMPORT/EXPORT commands [13](#), [17](#), [21](#)
- AMS REPRO command [13](#), [17](#), [21](#)
- AMS REPRO encryption [227](#)
- ARM
 - Automatic Restart Manager [114](#)
- assistive technologies [513](#)
- Automatic Restart Manager
 - ARM [114](#)

B

- BEGIN installation option [40](#)

C

- callable services exit
 - CSF_SERVICE_EXIT [186](#)
- changing parameters in installation options data set
 - specifying option parameters and values [40](#)
- changing the master key in compatibility or coexistence mode [229](#)
- CHECKAUTH installation option [40](#)
- choosing compatibility modes during migration [229](#)
- CICS
 - WAITLIST installation option [54](#)
- CICS-ICSF Attachment
 - Facility
 - installing [485](#)
- CICSAUDIT installation option [41](#)
- CIPHER macro
 - SVC description [7](#)
- CKDS
 - create [12](#)
 - primary space required [12](#)
 - secondary space required [13](#)
- CKDS (cryptographic key data set)
 - conversion from PCF CKDS to ICSF CKDS [230](#)
 - creating [13](#)
 - description [5](#)
 - header record format [243–245](#)
 - record format [245–247](#), [319](#), [320](#)

- CKDS entry retrieval installation exit
 - environment [188](#)
 - input [189](#)
 - installing [189](#)
 - purpose and use [188](#)
 - return codes [190](#)
- CKDS refresh
 - SMF record type [82](#) [148](#)
- CKDSN installation option [41](#)
- CKTAUTH [41](#)
- coexistence mode
 - changing the master key [228](#), [229](#)
 - description [227](#), [228](#)
- coexistence, definition [57](#)
- command
 - syntax diagrams [116](#)
- command syntax notation [116](#)
- COMPAT installation option [41](#), [227](#)
- compatibility mode
 - and the Access Method Services Cryptographic Option [145](#)
 - changing the master key [227](#), [229](#)
 - description [227](#)
- COMPENC installation option [42](#)
- compliance warning event
 - SMF record type [82](#) [154](#)
- compliance warnings [108](#)
- COMPLIANCEWARN installation option [42](#)
- component trace [157](#)
- configure on/off cryptographic coprocessors [139](#)
- contact
 - z/OS [513](#)
- controlling access to CSFDUTIL [155](#)
- controlling access to secure tokens [156](#)
- controlling access to the callable services [155](#)
- controlling access to the cryptographic keys [155](#)
- controlling access to the key generator utility program [155](#)
- controlling the program environment [154](#)
- conversion program
 - activity report [238](#)
 - bypassing entries [233](#)
 - converting key types [235](#)
 - data sets [238](#)
 - including information in a key entry [234](#)
 - installation exit [231](#)
 - JCL for submitting [237](#)
 - override file [231](#)
 - running [237](#)
- conversion program installation exit
 - PCF
 - purpose and use [191](#)
 - return codes [193](#)
- converting a PCF CKDS [230](#)
- Converting to KDSR format [102](#)
- coprocessor environment [501](#)
- CP Assist for Cryptographic Functions
 - description [1](#)

- Creating an
 - creating an ICSF CTRACE Configuration Data Set [26](#)
 - Creating an ICSF CTRACE Configuration Data Set [26](#)
- creating the CKDS
 - allocating space for the CKDS [12](#)
 - reading the CKDS into storage [33](#)
 - using the AMS DEFINE CLUSTER command [13](#)
- creating the installation options data set
 - guidelines [24](#)
- creating the PKDS
 - allocating space for the PKDS [17](#)
- creating the startup procedure
 - specifying the installation options data set [28](#)
- creating the TKDS
 - allocating space for the TKDS [21](#)
- creation of [13](#)
- crypto education [xxi](#)
- cryptographic communication vector table [400](#)
- cryptographic communication vector table extension [402](#)
- Cryptographic Coprocessor clear master key entry
 - SMF record type [82](#) [149](#)
- cryptographic coprocessor retained key create or delete
 - SMF record type [82](#) [149](#)
- cryptographic coprocessor timing
 - SMF record type [82](#) [150](#)
- cryptographic coprocessor TKE command request or reply
 - SMF record type [82](#) [149](#)
- cryptographic coprocessors
 - bringing offline [139](#)
 - bringing online [139](#)
 - disabling [139](#)
- csf [28](#)
- CSF_SERVICE_EXIT [186](#)
- csf2 [29](#)
- CSFAPRPC processing routine [216](#)
- CSFCKDS exit [188](#)
- CSFCONVX exit [190](#)
- CSFESECI exit [197](#)
- CSFESECK exit [197](#)
- CSFESECS exit [197](#)
- CSFESECT exit [197](#)
- CSFEXIT1 exit [170](#)
- CSFEXIT2 exit [170](#)
- CSFEXIT3 exit [170](#)
- CSFEXIT4 exit [170](#)
- CSFEXIT5 exit [170](#)
- CSFKGUP exit [201](#)
- CSFPARM data set [29](#)
- CSFPRM00 [25](#)
- CSFSRRW exit [193](#)
- CSFVINP data set [238](#)
- CSFVNEW data set [238](#)
- CSFVOVR data set [238](#)
- CSFVRPT data set [238](#)
- CSFVSRC data set [238](#)
- CTRACE installation option [42](#)

D

- DEFAULTWRAP installation option [43](#)
- DEFINE CLUSTER command [13](#), [17](#), [21](#)
- defining conversion program data sets [238](#)
- disabling cryptographic coprocessors [139](#)
- Display ICSF command [119](#)

- DOMAIN installation option [43](#)
- duplicate key tokens
 - SMF record type [82](#) [151](#)
- dynamic CKDS update
 - SMF record type [82](#) [148](#)
- dynamic PKDS update
 - SMF record type [82](#) [148](#)
- dynamic service update [104](#), [140](#), [141](#), [143](#), [144](#)

E

- ECC token
 - associated data format for [364](#)
- EMK macro
 - SVC description [7](#)
- END installation option [44](#)
- ENF signals [162](#)
- event recording [145](#)
- exit
 - CKDS entry retrieval installation exit [166](#), [188](#)
 - description [165](#)
 - entry and return specifications [167](#)
 - identifier on ICSF [44](#)
 - key generator utility program installation exit [167](#), [201](#)
 - mainline installation exits [165](#), [170](#)
 - PCF conversion program installation exit [166](#), [190](#)
 - security installation exits [197](#)
 - service installation exits [166](#), [178](#)
 - single-record, read-write installation exit [166](#), [193](#)
- EXIT installation option [44](#)
- exit name table [175](#)
- external key token
 - PKA
 - RSA private [330](#)

F

- FIPSMODE installation option [45](#)
- FMID
 - applicable z/OS releases
 - [4](#)
 - hardware [4](#)
 - servers [4](#)
- formatting control blocks
 - using IPCS [158](#)

G

- GENKEY macro
 - SVC description [7](#)

H

- hash services [62](#), [63](#)

I

- ICSF
 - dispatching priority [55](#), [144](#)
- ICSF (Integrated Cryptographic Service Facility)
 - CSFZSM82 mapping macro [411](#)
 - record type [1154](#) [477](#)
 - record type [82](#) [411](#)

ICSF CTRACE Configuration Data Set [26](#)

ICSF initialization

SMF record type [82](#) [147](#)

ICSF installation options data set

deprecated parameters [61](#)

ICSF migration actions [58](#)

ICSF operator commands

Display ICSF [119](#)

SETICSF [130](#)

icsf sysplex group

SMF record type [82](#) [150](#)

ICSFMIG77A1_COPROCESSOR_ACTIVE [58](#)

ICSFMIG77A1_TKDS_OBJECT [59](#)

initializing ICSF

creating the PKDS [17](#)

creating the TKDS [21](#)

creation of [17](#), [21](#)

selecting ICSF startup options

creating the installation options data set [24](#)

creating the startup procedure [28](#)

starting ICSF [33](#)

installation option keyword

CHECKAUTH [40](#)

CICSAUDIT [41](#)

CKDSN [41](#)

CKTAUTH [41](#)

COMPAT [41](#), [227](#)

COMPENC [42](#)

COMPLIANCEWARN [42](#)

CTRACE [42](#)

DEFAULTWRAP [43](#)

DOMAIN [43](#)

EXIT [44](#)

FIPSMODE [45](#)

KEYAUTH [48](#)

MASTERKCVLEN [48](#)

MAXLEN [48](#)

MAXSESSOBJECTS [48](#)

PKDSCACHE [49](#)

PKDSN [49](#)

REASONCODES [49](#)

SERVICE [49](#)

SERVICELIBS [50](#)

SERVSCSFMOD0 [50](#)

SERVSIEALNKE [50](#)

SSM [50](#)

SYSPLEXCKDS [51](#)

SYSPLEXTKDS [52](#)

TKDSN [53](#)

TRACKCLASSUSAGE [53](#)

UDX [54](#)

USERPARM [54](#)

WAITLIST [54](#)

installation option parameter

BEGIN [40](#)

END [44](#)

installation options

performance considerations [144](#)

installation options data set

changing option parameters and values [40](#)

creating [25](#)

example [25](#)

specifying the installation options data set [28](#)

installation steps [9](#)

installation-defined service

access control checking [215](#)

defining [216](#)

description [213](#)

entry and exit code example [215](#)

executing [217](#)

link editing [215](#)

parameter checking [215](#)

writing [213](#)

Integrity [388](#)

internal key token

aes; [282](#)

DES [284](#)

PKA

RSA private [344](#), [346–349](#), [359](#), [365](#)

IPCS support

contention issues [162](#)

K

KDSR

format [279](#)

KDSR record

format [279](#)

key generator utility program exit parameter block [203–211](#)

key generator utility program installation exit

calling points [201](#)

environment [202](#)

installing [202](#)

processing [202](#)

purpose and use [201](#)

return codes [211](#)

SET statement [211](#)

key part entry

SMF record type [82](#) [148](#)

key store policy

SMF record type [82](#) [151](#)

Key store policy [99](#)

key token

aes; internal [282](#)

DES

null [289](#)

DES internal [284](#)

PKA

null [329](#)

RSA 1024-bit private internal [346–349](#)

RSA 2048-bit Chinese Remainder Theorem private

internal [357–359](#)

RSA private external [330](#)

RSA private internal [344](#), [359](#), [365](#)

RSA public [329](#)

QSA [365](#)

KEYAUTH installation option [48](#)

keyboard

navigation [513](#)

PF keys [513](#)

shortcut keys [513](#)

KGUP [98](#)

L

link editing

callable services [215](#)

M

- mainline installation exit
 - environment [170](#)
 - exit parameter block [172](#)
 - input [171](#)
 - installing [171](#)
 - parameters [173](#), [177](#)
 - purpose and use [170](#)
- mapping macro
 - CSFZSM82 (ICSF) [412](#)
- master key event
 - SMF record type 82 [154](#), [417–423](#), [425](#), [427–435](#), [440](#), [443](#), [446](#), [452](#), [455](#), [458](#), [464](#), [466](#), [473](#)
- MASTERKCVLEN installation option [48](#)
- MAXLEN installation option [48](#)
- MAXSESSOBJECTS installation option [48](#)
- message recording [154](#)
- migrating from PCF [227](#)
- Migrating to the common record format (KDSR) key data set [102](#)
- migration
 - terminology [57](#)
- migration actions
 - Cryptographic Services [58](#)
- migration process [108](#)
- MKVP date examples [128](#)
- MODIFY command [116](#)
- modifying ICSF [116](#)

N

- navigation
 - keyboard [513](#)
- noncompatibility mode
 - description [227](#), [229](#)
- null key token
 - format [289](#), [329](#)

O

- operator commands
 - ICSF [118](#)
- override file
 - defining on a DD statement [238](#)

P

- panels
 - accessing [30](#)
 - ICSF Coprocessor Management [139](#)
- parameter checking
 - callable services [215](#)
- PCF
 - application [227–229](#)
 - macro [227](#)
 - migration to ICSF [227](#)
- PCF conversion program installation exit
 - environment [191](#)
 - input [192](#)
 - installing [191](#)
 - purpose and use [191](#)
- PCI Cryptographic Coprocessor configuration

- PCI Cryptographic Coprocessor configuration (*continued*)
 - SMF record type 82 [150](#)
- PCI-HSM 2016 compliance mode [107](#)
- performance
 - problems [55](#), [144](#)
- PKA key token
 - record format
 - RSA 1024-bit private internal [346–349](#)
 - RSA 2048-bit Chinese Remainder Theorem private internal [357–359](#)
 - RSA private external [330](#)
 - RSA private internal [344](#), [359](#), [365](#)
 - RSA public [329](#)
- PKDS (public key data set)
 - creating [17](#)
 - description [5](#)
 - header record format [248](#)
 - record format [249](#), [250](#)
- PKDSCACHE installation option [49](#)
- PKDSN installation option [49](#)
- private external key token
 - RSA [330](#)
- private internal key token
 - RSA [344](#), [346–349](#), [359](#), [365](#)
- protected key
 - retrieving early in IPL [116](#)
- public key data set
 - improving security and reliability for the PKDS [17](#)
- public key data set refresh
 - SMF record type 82 [151](#)
- public key token
 - RSA [329](#)

Q

- QSA key token format [365](#)

R

- read-write exit parameter block [195](#), [196](#)
- REASONCODES installation option [49](#)
- recording events [145](#)
- RETKEY macro
 - SVC description [7](#)
- return codes
 - from PCF macros
 - migration consideration [227](#)
- RKX key-token [288](#)
- RMF
 - header record format [404](#)
- RSA 1024-bit private internal key token [346–349](#)
- RSA key token formats [329](#)
- RSA private external key token [330](#)
- RSA private internal Chinese Remainder Theorem key token [357–359](#)
- RSA private internal key token [344](#), [359](#), [365](#)
- RSA public token [329](#)
- running ICSF
 - in coexistence mode [228](#)
 - in compatibility mode [227](#)
 - in noncompatibility mode [229](#)
- running the conversion program
 - creating a job to run the conversion program [237](#)

running the conversion program (*continued*)
defining conversion program data sets [238](#)

S

scheduling changes for cryptographic keys [156](#)
secondary parameter block [183](#)
section sequence, trusted block [387](#)
security considerations [154](#)
security installation exit
 environment [198](#)
 input [199](#)
 installing [198](#)
 purpose and use [197](#)
 return codes [200](#)
selecting ICSF startup options
 creating the installation options data set [24](#)
 creating the startup procedure [28](#)
service installation exit
 environment [178](#)
 exit parameter block [181](#)
 input [180](#)
 installing [179](#)
 parameters [185](#)
 purpose and use [178](#)
 return codes [185](#)
SERVICE installation option
 syntax [216](#)
service names used in SMF records [416](#)
service stub
 description [213](#)
 example [218](#)
 linking [217](#)
 writing [216](#)
SERVICELIBS installation option [50](#)
SERVSCSFMOD0 installation option [50](#)
SERVSIEALNKE installation option [50](#)
SETICSF command [130](#)
shortcut keys [513](#)
single-record, read-write installation
 exit
 conversion program invocation [231](#)
 input [195](#)
 installing [194](#)
 purpose and use [194](#)
 return codes [196](#)
SMF record type 82
 subtype 1 [147](#), [417](#)
 subtype 13 [148](#), [420](#)
 subtype 14 [149](#), [421](#)
 subtype 15 [149](#), [422](#)
 subtype 16 [149](#), [423](#)
 subtype 18 [150](#), [425](#)
 subtype 20 [150](#), [427](#)
 subtype 21 [150](#), [428](#)
 subtype 22 [150](#), [429](#)
 subtype 23 [151](#), [430](#)
 subtype 24 [151](#), [430](#)
 subtype 25 [151](#), [430](#)
 subtype 26 [151](#), [431](#)
 subtype 27 [432](#)
 subtype 28 [433](#)
 subtype 29 [434](#)
 subtype 30 [434](#)

SMF record type 82 (*continued*)
 subtype 31 [435](#)
 subtype 40 [440](#)
 subtype 41 [443](#)
 subtype 42 [446](#)
 subtype 44 [452](#)
 subtype 45 [455](#)
 subtype 46 [458](#)
 subtype 47 [464](#)
 subtype 48 [154](#), [466](#)
 subtype 49 [154](#), [473](#)
 subtype 7 [148](#), [418](#)
 subtype 8 [148](#), [419](#)
 subtype 9 [148](#), [420](#)
SMF recording [145](#), [211](#)
specifying the installation options data set [28](#)
SSM installation option [50](#)
START command [33](#)
starting ICSF
 creating the startup procedure [28](#)
 entering the ICSF START command [33](#), [113](#)
 IPL-time [114](#)
startup procedure [9](#), [28](#)
steps in installation [9](#)
stopping ICSF [113](#)
summary of changes [xxv](#)
SVC [143](#) [7](#)
syntax diagrams
 how to read [116](#)
SYS1.PARMLIB
 customizing [10](#)
 description [9](#)
 storing startup procedure [30](#)
SYS1.PROCLIB
 description [9](#)
 storing startup procedure [29](#)
SYS1.SAMPLIB
 CSFPRM00 [25](#)
 description [9](#)
SYSPLEXCKDS installation option [51](#)
SYSPLEXPADS installation option [52](#)
SYSPLEXTKDS installation option [52](#)

T

testing ICSF [229](#)
TKDS
 SMF record type 82 [151](#)
TKDS (public key data set)
 creating [21](#)
TKDS (token data set)
 description [102](#)
 format [250](#)
TKDS (token key data set)
 description [6](#)
TKDSN installation option [53](#)
token data set (TKDS)
 description [102](#)
 format [250](#)
token key data set
 improving security and reliability for the TKDS [21](#)
token validation value (TVV) [283](#)
TRACEENTRY option and ICSF [64](#)
TRACKCLASSUSAGE installation option [53](#)

trademarks [518](#)
trusted block create
 SMF record type 82 [150](#)
trusted blocks [386](#)

U

udx
 access control checking [215](#)
UDX installation option [54](#)
user interface
 ISPF [513](#)
 TSO/E [513](#)
USERPARM installation option [54](#)
using different configurations [137](#)
using the conversion program override file [231](#)

V

VERBX [159](#)
virtual storage constraint relief
 for the caller of ICSF [144](#)
VSAM data set
 creating [13](#)
VTAM session-level encryption
 and ICSF [144](#)

W

WAITLIST installation option [54](#)

X

X9.143 (TR-31)
 key block header [320](#)
 key block support [98](#)
 optional block data [320](#)



SC14-7507-70

