ISC Cryptographic Development Kit (CDK)

FIPS 140-3 Non-Proprietary Security Policy

Software Version: 8.1.2.3

Document Version: 4.1.8

Issue Date: March 14, 2025

Authors: Michael J. Markowitz, Roger S. Schlafly, Jonathan C. Schulze-Hewett

Abstract: This document is a non-proprietary FIPS 140-3 Security Policy for ISC's Cryptographic Development Kit (CDK). It applies to CDK Version 8.1.2.3 and to all subsequent versions until otherwise indicated in new editions. It describes how the CDK meets the security requirements of FIPS 140-3 and how to run the CDK in the Approved mode. This policy was prepared as part of the FIPS 140-3 Level 1 validation of the module.

Notices

Document History

Version	Date	Change	Author
1.0.12	2002-05-22	First submitted version	Michael Markowitz
2.0.0	2003-02-04	Tweaks to language per lab	Jonathan Schulze-Hewett
3.0.0	2003-03-17	Added methods available to CO and Users	Jonathan Schulze-Hewett
3.1.0	2003-03-21	Tweaks to language per lab	Jonathan Schulze-Hewett
3.4.0	2003-04-30	Added HMAC-SHA-1 to CO/User table, removed CTR from the list of modes for EES	Jonathan Schulze-Hewett
3.5.0	2003-05-01	Revisions	Michael Markowitz
3.6.0	2003-05-07	Revised footer explaining CTR mode applicability	Michael Markowitz
3.7.0	2003-05-12	Added second footer explaining CTR mode applicability or lack thereof	Jonathan Schulze-Hewett
3.8.0	2003-06-04	Added footer explaining DES variants	Jonathan Schulze-Hewett
3.9.0	2003-07-25	Revisions	Jonathan Schulze-Hewett
3.10.0	2003-08-25	Final 140-1 document	Michael Markowitz
4.0.0	2016-02-23	Updated for CDK 8/FIPS 140-2	Jonathan Schulze-Hewett
4.0.1	2016-04-08	Revisions as per lab comments	Michael Markowitz, Jonathan Schulze-Hewett
4.0.2	2016-05-23	Revisions as per lab comments	Jonathan Schulze-Hewett
4.0.3	2016-06-23	Revisions as per lab comments	Jonathan Schulze-Hewett
4.0.4	2016-08-09	Revisions as per lab comments	Jonathan Schulze-Hewett
4.0.5	2016-09-09	Revisions as per lab comments	Jonathan Schulze-Hewett
4.0.6	2016-09-13	Changed RSA self-test from PCT to KAT Updated the filename of the CDK DLL Revisions as per lab comments	Jonathan Schulze-Hewett
4.0.7	2016-09-30	Modified the algorithm tables to comply with CMVP's October 2016 requirements	Jonathan Schulze-Hewett
4.0.8	2016-10-28	Moved HMAC-SHA-3 into approved from table 4 to table 3 and Revisions as per lab comments	Jonathan Schulze-Hewett
4.0.9	2016-12-16	Moved Skipjack (EES) into non-approved table per lab comments.	Jonathan Schulze-Hewett
4.0.10	2017-03-27	Revisions per lab comments	Jonathan Schulze-Hewett
4.0.11	2017-03-28	Revisions per lab comments	Jonathan Schulze-Hewett
4.0.12	2017-04-18	Revisions per lab comments	Jonathan Schulze-Hewett
4.0.13	2017-06-06	AES-GCM clarifications	Jonathan Schulze-Hewett
4.0.14	2017-10-13	AES-GCM clarifications	Jonathan Schulze-Hewett
4.0.15	2018-06-01	Updated for 1SUB version change	Jonathan Schulze-Hewett
4.1.0	2022-02-21	Updated for CDK 8.1	Jonathan Schulze-Hewett
4.1.1	2022-05-23	Updated based on SP 800-140Br2 guidance	Jonathan Schulze-Hewett
4.1.2	2023-02-07	Updated based on additional functionality introduced in 8.1	Jonathan Schulze-Hewett
4.1.3	2024-01-31	Updated based on lab feedback	Jonathan Schulze-Hewett
4.1.4	2024-02-28	Updated based on lab feedback	Jonathan Schulze-Hewett
4.1.5	2024-08-05	Updated for KAS-IFC-SSC CAST version change	Jonathan Schulze-Hewett
4.1.6	2024-11-15	Updated based on lab feedback	Jonathan Schulze-Hewett
4.1.7	2025-03-03	Updated based on lab feedback	Jonathan Schulze-Hewett
4.1.8	2025-03-14	Updated based on lab feedback	Jonathan Schulze-Hewett

References

Reference	Full Specification Name
[ANS X9.30 Part 1]	Public Key Cryptography Using Irreversible Algorithms – Part 1: The Digital Signature Algorithm (DSA)
[ANS X9.30 Part 2]	Public Key Cryptography Using Irreversible Algorithms – Part 2: The Secure Hash Algorithm (SHA-1)
[ANS X9.31]	Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA)
[FIPS 46-3]	Data Encryption Standard (DES) (withdrawn)
[FIPS 81]	DES Modes of Operation (withdrawn)
[FIPS 140-3]	Security Requirements for Cryptographic modules, March 22, 2019
[FIPS 180-4]	Secure Hash Standard (SHS)
[FIPS 185]	Escrowed Encryption Standard (obsolete)
[FIPS 186-4]	Digital Signature Standard
[FIPS 197]	Advanced Encryption Standard
[FIPS 198-1]	The Keyed-Hash Message Authentication Code (HMAC)
[FIPS 202]	SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions
[IEEE P1619-2007]	Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices
[ISO/IEC 10118-3:1998]	Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions
[ISO/IEC 19790:2012]	Information technology – Security techniques – Security requirements for cryptographic modules
[RFC 2437]	PKCS #1: RSA Cryptography Specifications, Version 2.0
[RFC 2104]	HMAC: Keyed-Hashing for Message Authentication
[RFC 3447]	Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1
[RFC 3610]	Counter with CBC-MAC (CCM)
[RFC 4493]	The AES-CMAC Algorithm
[SP 800-20]	Modes of Operation Validation System for the Triple Data Encryption Algorithm (TMOVS): Requirements and Procedures (withdrawn)
[SP 800-38A]	Recommendation for Block Cipher Modes of Operation: Methods and Techniques
[SP 800-38B]	Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication
[SP 800-38C]	Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality
[SP 800-38D]	Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC
[SP 800-38E]	Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices
[SP 800-56A R3]	Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography
[SP 800-67 R2]	Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher
[SP 800-89]	Recommendation for Obtaining Assurances for Digital Signature Applications
[SP 800-90A R1]	Recommendation for Random Number Generation Using Deterministic Random Bit Generators
[SP 800-90B]	Recommendation for the Entropy Sources Used for Random Bit Generation
[SP 800-107 R1]	Recommendation for Applications Using Approved Hash Algorithms
[SP 800-131A R2]	Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths
[SP 800-133 R2]	Recommendation for Cryptographic Key Generation

Table of Contents

1. General

1.1 Overview

Information Security Corporation's Cryptographic Development Kit (CDK) Version 8.1.2.3 is a software module. The software module is a shared library that contains cryptographic primitives that are cryptographic software building blocks which may be used by application developers to build security-enhanced features into their own applications. The CDK provides public-key algorithms, as well as symmetric ciphers, hashing functions, and related cryptographic and PKI operations.

The CDK was designed and implemented to meet FIPS 140-3 level 1 security requirements.

1.1.1 Document Organization

ISC's submission for FIPS 140-3 validation includes this security policy document and:

- Vendor evidence (Entropy statement, Testing statement, Crypto Officer's Guide, Cryptographic Key Management Document, Evaluator's Guide, and the CDK User's Guide),
- Finite state machine model diagram and explanation,
- Proprietary source code and build configurations for various target platforms.

1.2 Security Levels

The following table lists the validation level met by the CDK for each area in FIPS 140-3. The CDK meets the requirements for an overall FIPS 140-3 level 1 validation.

ISO/IEC 24759 Section 6. [Number Below]	FIPS 140-3 Section Title	Security Level
1	General	1
2	Cryptographic Module Specification	1
3	Cryptographic Module Interfaces	1
4	Roles, Services, and Authentication	1
5	Software/Firmware Security	1
6	Operational Environment	1
7	Physical Security	N/A
8	Non-Invasive Security	N/A
9	Sensitive Security Parameter Management	1
10	Self-Tests	1
11	Life-Cycle Assurance	1
12	Mitigation of Other Attacks	N/A

Table 1 – Security Levels

The “Physical Security” section is not applicable as the module is a software only, level 1, module. The “Non-Invasive Security” and “Mitigation of Other Attacks” sections are not relevant as the CDK is a software module and does not implement any countermeasures towards special attacks.

1.3 References

Federal Information Processing Standards Publication (FIPS PUB) 140-3, *Security Requirements for Cryptographic Modules*, details U.S. Government requirements for cryptographic modules. Below are hyperlinks to websites containing more information on NIST cryptographic programs, FIPS 140-3, and the CDK.

NIST Cryptographic Module Validation Program (CMVP)	https://csrc.nist.gov/projects/cryptographic-module-validation-program
FIPS 140-3 Security Requirements	https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-3.pdf
ISC CDK	https://infoseccorp.com/cdks.html
NIST Validation Lists for Cryptographic Standards – this site contains the technical implementations of the algorithms that have been validated to conform to the NIST approved algorithm standards	https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/validation-search

2. Cryptographic Module Specification

2.1 Module Description and Overview

The CDK cryptographic module is a multi-chip standalone software module running on a standalone general-purpose computing platform. The CDK provides cryptographic services to applications through a C++ language Application Program Interface (API). The "cryptographic boundary" is defined as the binary shared link library (cdkc8123Sx64.dll or libcdkc.so.81.2.3).

The CDK was tested on the following operational environments on the general-purpose computer (GPC) platforms shown in Table 2. These are the TOEPP (Tested Operational Environment's Physical Perimeter) of the module.

#	Operating System	Hardware Platform	Processor	PAA/Acceleration
1	Windows 10 64-bit	Dell Inspiron 15	Intel(R) Core(TM) i7-11390H (Tiger Lake)	AES-NI
2	Windows 10 64-bit	Dell Inspiron 15	Intel(R) Core(TM) i7-11390H (Tiger Lake)	None
3	CentOS 7.7 64-bit	Dell Inspiron 15	Intel(R) Core(TM) i7-11390H (Tiger Lake)	AES-NI
4	CentOS 7.7 64-bit	Dell Inspiron 15	Intel(R) Core(TM) i7-11390H (Tiger Lake)	None
5	Raspberry Pi OS 32-bit	Raspberry Pi 4 Model B	Broadcom BCM2711	None
6	Raspberry Pi OS 64-bit	Raspberry Pi 4 Model B	Broadcom BCM2835	None

Table 2 – Tested Operational Environments

There are no Vendor Affirmed Operational Environments at this time.

2.2 Cryptographic Algorithms

The CDK supports a wide variety of cryptographic algorithms and can be configured to run in an Approved mode or a Non-Approved mode. The keys and CSPs used for cryptographic operations are not shared between the modes of operation. Whenever possible, all Approved algorithms designed for a particular cryptographic function (such as encryption, message and entity authentication, hashing, *etc.*) are provided.

2.2.1 Algorithms and Parameters Allowed in the Approved mode

2.2.1.1 Approved Algorithms

The Approved cryptographic algorithms implemented in the CDK and the corresponding NIST standards (or alternate standards referenced by NIST) are listed in Table 3 along with CAVP certificate numbers. When the CDK is run in the Approved mode, only algorithms in Table 3, Table 4, and Table 5 can be used.

CAVP Cert	Algorithm and Standard	Mode/Method	Description / Key Size(s) / Key Strength(s)	Use / Function
A5798	AES-CBC [FIPS 197, SP 800-38A]	CBC	Direction: Decrypt, Encrypt Key Length: 128, 192, 256	Data Encryption/ Decryption
A5798	AES-CCM [FIPS 197, SP 800-38C]	CCM	Direction: Decrypt, Encrypt Key Length: 128, 192, 256	Data Encryption / Decryption
A5798	AES-CFB128 [FIPS 197, SP 800-38A]	CFB128	Direction: Decrypt, Encrypt Key Length: 128, 192, 256	Data Encryption / Decryption
A5798	AES-CFB8 [FIPS 197, SP 800-38A]	CFB8	Direction: Decrypt, Encrypt Key Length: 128, 192, 256	Data Encryption / Decryption
A5798	AES-CMAC [FIPS 197, SP 800-38B]	CMAC	Direction: Generation, Verification Key Length: 128, 192, 256	Message Authentication
A5798	AES-CTR [FIPS 197, SP 800-38A]	CTR	Direction: Decrypt, Encrypt Key Length: 128, 192, 256	Data Encryption / Decryption
A5798	AES-ECB [FIPS 197, SP 800-38A]	ECB	Direction: Decrypt, Encrypt Key Length: 128, 192, 256	Data Encryption / Decryption
A5798	AES-GCM [FIPS 197, SP 800-38D]	GCM[1]	Direction: Decrypt, Encrypt Key Length: 128, 192, 256	Data Encryption / Decryption
A5798	AES-KW [FIPS 197, SP 800-38F]	KW	Direction: Decrypt, Encrypt Cipher: Cipher, Inverse Key Length: 128, 192, 256	Key Wrapping / Unwrapping
A5798	AES-KWP [FIPS 197, SP 800-38F]	KWP	Direction: Decrypt, Encrypt Cipher: Cipher, Inverse Key Length: 128, 192, 256	Key Wrapping / Unwrapping
A5798	AES-OFB [FIPS 197, SP 800-38A]	OFB	Direction: Decrypt, Encrypt Key Length: 128, 192, 256	Data Encryption / Decryption
A5798	AES-XTS Testing Revision 2.0 [FIPS 197, SP 800-38E]	XTS[2]	Direction: Decrypt, Encrypt Key Length: 128, 256	Data Encryption / Decryption
A5798	Conditioning Component AES-CBC-MAC [SP800-90B]	CBC-MAC	Key Length: 128, 192, 256	Conditioning

A5798	cSHAKE-128 [SP 800-185]	cSHAKE-128	Message Length: 0-65536 Increment 8 Output Length: 16-65536 Increment 8	Extendable Output Function
A5798	cSHAKE-256 [SP 800-185]	cSHAKE-128	Message Length: 0-65536 Increment 8 Output Length: 16-65536 Increment 8	Extendable Output Function
A5798	DSA SigVer [FIPS186-4]	FIPS 186-4	Capabilities: L: 1024 N: 160, L: 2048 N: 224, L: 2048 N: 256, L: 3072 N: 256 Hash Algorithm: SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256	Signature Verification
A5798	Deterministic ECDSA SigGen [FIPS186-5]	FIPS 186-5	Curve: B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521 Hash Algorithm: SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512	Signature Generation
A5798	ECDSA KeyGen [FIPS186-4]	Secret Generation Mode: Testing Candidates	Curve: B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521	Key Pair Generation
A5798	ECDSA KeyVer [FIPS186-4]	FIPS 186-4	Curve: B-163, B-233, B-283, B-409, B-571, K-163, K-233, K-283, K-409, K-571, P-192, P-224, P-256, P-384, P-521	Key Pair Verification
A5798	ECDSA SigGen [FIPS186-4]	FIPS 186-4	Curve: B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521 Hash Algorithm: SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512	Signature Generation
A5798	ECDSA SigVer [FIPS186-4]	FIPS 186-4	Curve: B-163, B-233, B-283, B-409, B-571, K-163, K-233, K-283, K-409, K-571, P-192, P-224, P-256, P-384, P-521 Hash Algorithm: SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512	Signature Verification

[1] Internal IV generation only. The AES-GCM IV is generated internally randomly (scenario 2) or as a counter (scenario 1) per IG C.H.

[2] Only Approved for Storage Applications.

A5798	HMAC DRBG [SP 800-90A]	Mode: SHA2-256, SHA2-512	Entropy Input: 256-131072 Increment 8	Deterministic Random Bit Generation
A5798	HMAC-SHA-1 [FIPS 198-1]	HMAC-SHA-1	Key Length: 112-524288 Increment 8 MAC: 32-160	Message Authentication
A5798	HMAC-SHA2-224 [FIPS 198-1]	HMAC-SHA2-224	Key Length: 112-524288 Increment 8 MAC: 32-224	Message Authentication
A5798	HMAC-SHA2-256 [FIPS 198-1]	HMAC-SHA2-256	Key Length: 112-524288 Increment 8 MAC: 32-256	Message Authentication
A5798	HMAC-SHA2-384 [FIPS 198-1]	HMAC-SHA2-384	Key Length: 112-524288 Increment 8 MAC: 32-384	Message Authentication
A5798	HMAC-SHA2-512 [FIPS 198-1]	HMAC-SHA2-512	Key Length: 112-524288 Increment 8 MAC: 32-512	Message Authentication
A5798	HMAC-SHA2-512/224 [FIPS 198-1]	HMAC-SHA2-512/224	Key Length: 112-524288 Increment 8 MAC: 32-224	Message Authentication
A5798	HMAC-SHA2-512/256 [FIPS 198-1]	HMAC-SHA2-512/256	Key Length: 112-524288 Increment 8 MAC: 32-256	Message Authentication
A5798	HMAC-SHA3-224 [FIPS 198-1]	HMAC-SHA3-224	Key Length: 112-524288 Increment 8 MAC: 32-224	Message Authentication
A5798	HMAC-SHA3-256 [FIPS 198-1]	HMAC-SHA3-256	Key Length: 112-524288 Increment 8 MAC: 32-256	Message Authentication
A5798	HMAC-SHA3-384 [FIPS 198-1]	HMAC-SHA3-384	Key Length: 112-524288 Increment 8 MAC: 32-384	Message Authentication
A5798	HMAC-SHA3-512 [FIPS 198-1]	HMAC-SHA3-512	Key Length: 112-524288 Increment 8 MAC: 32-512	Message Authentication
A5798	KAS-ECC-SSC[3] [SP800-56Ar3]	ephemeralUnified, onePassDh, staticUnified	Domain Parameter Generation Methods: B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521	Key Agreement
A5798	KAS-IFC-SSC[3] [SP 800-56Br2]	KAS1	Modulo: 2048, 3072, 4096, 6144, 8192	Key Agreement
A5798	KDA HKDF [SP800-56Cr2]	HMAC Algorithm: SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512	Shared Secret Length: 224-1024 Increment 8	Key Derivation
A5798	KDA OneStep	Auxiliary Function Name: SHA3-512	Shared Secret Length: 224-1024 Increment 8	Key Derivation

[3] SSP establishment methodology provides between 112 and 256-bits of encryption strength.

	[SP800-56Cr2]			
A5798	KDF ANS 9.63 (CVL) [SP 800-135r1]	Hash Algorithm: SHA2-224, SHA2-256, SHA2-384, SHA2-512	Field Size: 224, 571	Key Derivation *No part of the ANS 9.63 protocol, other than the KDF, has been tested by the CAVP and CMVP.*
A5798	KDF IKEv1 (CVL) [SP 800-135r1]	Authentication Method: Digital Signature, Pre-shared Key, Public Key Encryption Hash Algorithm: SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	Diffie-Hellman Shared Secret Length: 224-8192 Increment 8	Key Derivation *No part of the IKEv1 protocol, other than the KDF, has been tested by the CAVP and CMVP.*
A5798	KDF IKEv2 (CVL) [SP 800-135r1]	Hash Algorithm: SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	Diffie-Hellman Shared Secret Length: 224-8192 Increment 8	Key Derivation *No part of the IKEv2 protocol, other than the KDF, has been tested by the CAVP and CMVP.*
A5798	KDF SNMP (CVL) [SP 800-135r1]	Engine ID: 12345678912345678900, abcdef0123456789abcdef1234567890	Password Length: 64, 8192	Key Derivation *No part of the SNMP protocol, other than the KDF, has been tested by the CAVP and CMVP.*
A5798	KDF SSH (CVL) [SP 800-135r1]	Hash Algorithm: SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	Cipher: AES-128, AES-192, AES-256, TDES	Key Derivation *No part of the SSH protocol, other than the KDF, has been tested by the CAVP and CMVP.*
A5798	KDF TLS (CVL) [SP 800-135r1]	TLS Version: v1.0/1.1	Hash Algorithm: SHA2-256, SHA2-384, SHA2-512	Key Derivation *No part of the TLS 1.0/1.1 protocol, other than the KDF, has been tested by the CAVP and CMVP.*
A5798	KDF TLS (CVL) [SP 800-135r1]	TLS Version: 1.2	Hash Algorithm: SHA2-256, SHA2-384, SHA2-512	Key Derivation *No part of the TLS 1.2 protocol, other than the KDF, has been tested by the CAVP and CMVP.*
A5798	KDF TPM (CVL)	TPM 1.2	HMAC with SHA-1	Key Derivation

	[SP 800-135r1]			*No part of the TPM 1.2 protocol, other than the KDF, has been tested by the CAVP and CMVP.*
A5798	KMAC-128 [SP 800-185]	Message Length: 0-65536 Increment 8; MAC Length: 32-65536 Increment 8	Key Data Length: 128-524288 Increment 8	XOF
A5798	KMAC-256 [SP 800-185]	Message Length: 0-65536 Increment 8; MAC Length: 32-65536 Increment 8	Key Data Length: 128-524288 Increment 8	XOF
A5798	KTS-IFC[4] [SP 800-56Br2]	KTS-OAEP-basic	Modulo: 2048, 3072, 4096, 6144, 8192	OAEP
A5798	ParallelHash-128 [SP 800-185]	Message Length: 0-65536 Increment 8	Output Length: 16-65536 Increment 8	XOF
A5798	ParallelHash-256 [SP 800-185]	Message Length: 0-65536 Increment 8	Output Length: 16-65536 Increment 8	XOF
A5798	PBKDF[5] [SP 800-132]	HMAC Algorithm: SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512	Password Length: 8-128 Increment 8	Key Derivation
A5798	RSA KeyGen [FIPS186-4]	Key Generation Mode: B.3.3 Primality Tests: Table C.2	Modulo: 2048, 3072, 4096, 6144, 8192[6]	Key Pair Generation
A5798	RSA SigGen [FIPS186-4]	Signature Type: ANSI X9.31, PKCS 1.5, PKCSPSS	Modulo: 2048, 3072, 4096, 6144, 8192[6]	Digital Signature Generation
A5798	RSA SigVer [FIPS186-4]	Signature Type: ANSI X9.31, PKCS 1.5, PKCSPSS	Modulo: 1024[7][Legacy], 2048, 3072, 4096, 6144, 8192[6]	Digital Signature Verification
A5798	RSA Signature Primitive (CVL)	RSA Signature Primitive	Private Key Format: standard Public Exponent Mode: random	Digital Signature Generation

[4] SSP establishment methodology provides between 112 and 256-bits of encryption strength.

[5] The module implements PBKDF in a manner that is compliant with option 1a from SP 800-132, section 5.4. With the minimum password length of 8 characters, the probability of randomly guessing this parameter is 1 in 256^8. The module supports a variable iteration count as low as 1, but based on SP 800-132, section 5.2 a minimum of 1,000 is recommended. Keys derived from passwords, as shown in SP 800-132, may only be used in storage applications.

[6] Modulo sizes 6144 and 8192 are supported but have not been CAVP tested.

[7] Legacy usage only. These legacy algorithms can only be used on data that was generated prior to the Legacy Date specified in FIPS 140-3 IG C.M.

	[FIPS 186-5, RFC 3447]			
A5798	SHA-1 [FIPS 180-4]	Function: SHA1	Message Length: 0-65536 Increment 8; Large Message Sizes: 1gigabytes	Message Digest
A5798	SHA2-224 [FIPS 180-4]	Function: SHA2	Message Length: 0-65536 Increment 8; Large Message Sizes: 1gigabytes	Message Digest
A5798	SHA2-256 [FIPS 180-4]	Function: SHA2	Message Length: 0-65536 Increment 8; Large Message Sizes: 1gigabytes	Message Digest
A5798	SHA2-384 [FIPS 180-4]	Function: SHA2	Message Length: 0-65536 Increment 8; Large Message Sizes: 1gigabytes	Message Digest
A5798	SHA2-512 [FIPS 180-4]	Function: SHA2	Message Length: 0-65536 Increment 8; Large Message Sizes: 1gigabytes	Message Digest
A5798	SHA2-512/224 [FIPS 180-4]	Function: SHA2	Message Length: 0-65536 Increment 8; Large Message Sizes: 1gigabytes	Message Digest
A5798	SHA2-512/256 [FIPS 180-4]	Function: SHA2	Message Length: 0-65536 Increment 8; Large Message Sizes: 1gigabytes	Message Digest
A5798	SHA3-224 [FIPS 202]	Function: SHA3	Message Length: 0-65536 Increment 8	Message Digest
A5798	SHA3-256 [FIPS 202]	Function: SHA3	Message Length: 0-65536 Increment 8	Message Digest
A5798	SHA3-384 [FIPS 202]	Function: SHA3	Message Length: 0-65536 Increment 8	Message Digest
A5798	SHA3-512 [FIPS 202]	Function: SHA3	Message Length: 0-65536 Increment 8	Message Digest
A5798	SHAKE-128 [FIPS 202]	SHAKE-128	Output Length: 16-65536 Increment 8	XOF
A5798	SHAKE-256 [FIPS 202]	SHAKE-256	Output Length: 16-65536 Increment 8	XOF
A5798	TDES-CBC [SP 800-38A, SP 800-67r2]	CBC	Keying Option: 1, 2^7[Legacy] Direction: Decrypt	Data Decryption
A5798	TDES-CFB64 [SP 800-38A, SP 800-67r2]	CFB64	Keying Option: 1, 2^7[Legacy] Direction: Decrypt	Data Decryption
A5798	TDES-CFB8 [SP 800-38A, SP 800-67r2]	CFB8	Keying Option: 1, 2^7[Legacy] Direction: Decrypt	Data Decryption
A5798	TDES-CTR [SP 800-38A, SP 800-67r2]	CTR	Keying Option: 1, 2^7[Legacy] Direction: Decrypt Incremental Counter	Data Decryption
A5798	TDES-ECB [SP 800-38A, SP 800-67r2]	ECB	Keying Option: 1, 2^7[Legacy] Direction: Decrypt	Data Decryption
A5798	TDES-OFB [SP 800-38A,	OFB	Keying Option: 1, 2^7[Legacy] Direction: Decrypt	Data Decryption

	SP 800-67r2]			
A5798	TLS v1.2 KDF RFC7627 [SP 800-135r1]	TLS v1.2 KDF RFC7627	Hash Algorithm: SHA2-256, SHA2-384, SHA2-512	Key Derivation *No part of the TLS 1.2 protocol, other than the KDF, has been tested by the CAVP and CMVP.*
A5798	TLS v1.3 KDF [SP 800-135r1, RFC8446]	TLS v1.3 KDF	HMAC Algorithm: SHA2-256, SHA2-384 KDF Running Modes: DHE, PSK, PSK-DHE	Key Derivation *No part of the TLS 1.3 protocol, other than the KDF, has been tested by the CAVP and CMVP.*
A5798	TupleHash-128 [SP 800-185]	Message Length: 0-65536 Increment 8	Output Length: 16-65536 Increment 8	XOF
A5798	TupleHash-256 [SP 800-185]	Message Length: 0-65536 Increment 8	Output Length: 16-65536 Increment 8	XOF
Security Function Implementations (SFIs)				
AES-KW A5798	KTS [SP 800-38F]	SP 800-38F. KTS (key wrapping and unwrapping) per IG D.G.	128, 192, and 256-bit keys provide between 128 and 256 bits of encryption strength	Key Wrap/Unwrap
AES-KWP A5798	KTS [SP 800-38F]	SP 800-38F. KTS (key wrapping and unwrapping) per IG D.G.	128, 192, and 256-bit keys provide between 128 and 256 bits of encryption strength	Key Wrap/Unwrap
KTS-IFC A5798	KTS [SP 800-56Brev2]	SP 800-56Brev2. KTS-IFC (key encapsulation and un-encapsulation) per IG D.G.	2048, 3072, 4096, 6144, and 8192-bit modulus providing 112, 128, 152, 176, or 200 bits of encryption strength	Key Encapsulation/Un-encapsulation
Entropy Source				
E21	ESV [SP 800-90B]	Non-Physical, Non-IID	Entropy Input obtained by DRBG is 48 bytes. Nonce obtained by DRBG is 16 bytes. Due to conditioning, the output from this entropy source is expected to contain full entropy.	Seeding the Module's DRBG

Table 3 - Approved Algorithms

The CDK provides the Approved methods in Table 4 for which there are no algorithm tests, but whose use is nevertheless allowed in the Approved mode. The proper implementation and functionality of these mechanisms is "Vendor Affirmed."

Algorithm	Caveat	Use/Function
CKG	Vendor Affirmed	Cryptographic Key Generation

[SP 800-133 rev2]		SP 800- 133rev2 (Section 4, 5.1, 6.1, and 6.3) and IG D.H. Unmodified output DRBG used for generation of symmetric keys and seeds for asymmetric keys. DRBGs are instantiated to 256 bits of security strength

Table 4 - Vendor Affirmed Algorithms

The module uses the unmodified output of the approved DRBG for the generation of symmetric keys and seeds for asymmetric keys.

2.2.1.2 SP 800-56Arev3 Assurances

As required per SP 800-56Arev3, the CDK conditionally performs the necessary checks when generating, importing, or using domain parameters and keys according to sections 5.5.2, 5.6.2, and/or 5.6.3 of the special publication.

2.2.1.3 HMAC Usage

If HMAC is used for the protection of data, the operator must ensure that a key length of at least 112 bits minimum is used.

2.2.1.4 AES-GCM Notes

2.2.1.4.1 IV Construction

In the Approved mode only internal 96-bit IV generation is allowed. The CDK supports both SP800-38D Section 8.1 and 8.2 for internal IV generation and therefore complies with both Scenario 1 and Scenario 2 of IG C.H.

2.2.1.4.1.1 TLS – Section 8.1 SP800-38D, Scenario 1 IG C.H

The CDK does not implement the TLS protocol. The CDK implements cryptographic operations that can be used to implement the TLS protocol. The CDK's AES GCM TLS internal IV generation is in compliance with TLS 1.2 per RFC 5288, TLS 1.3 per RFC 8446, and in support of the GCM cipher suites listed in SP800-52rev2. For TLS 1.2 the AES GCM IV is generated internally using a deterministic counter as the nonce_explicit value and takes as input a 16-bit salt. For TLS 1.3 the AES GCM IV is generated internally by XOR'ing an internally maintained counter with the 12-byte IV. In each case, the resulting IV is exactly 96-bits in length.

2.2.1.4.1.2 Random – Section 8.2 SP800-38D, Scenario 2 IG C.H

The CDK implements random internal IV generation that uses the module's Approved DRBG. The seed used by the CDK's DRBG is provided by the CDK's jitter entropy component. The IV is exactly 96-bits in length.

2.2.1.4.2 Power Loss

In the event that module power is lost and restored, the application using the CDK must ensure that any of its AES-GCM keys used for encryption are re-established or re-generated.

2.2.1.4.3 Limits

The CDK enforces the following limits on the number of encryption operations that can be performed with GCM:

- TLS IV 64-bit Deterministic Counter with 16-bit salt – if the counter exceeds 2^{64} the CDK returns a `CDK_INVALID_IV` error with value `1064`.
- Random IV – if the number of AES operations exceeds 2^{32}-1 the CDK returns `CDK_INPUT_LENGTH_ERR` error with value `1151`.

2.2.2 Non-Approved Algorithms Allowed in the Approved Mode of Operation

The CDK does not support any non-approved algorithms that are allowed in the Approved mode.

2.2.3 Non-Approved-mode Algorithms

When run in the Approved mode the following additional algorithms, modes, and sizes are allowed to be used internally with No Security Claimed. Note these algorithms are not accessible through the CDK's API and calling applications are unable to use them.

Algorithm	Caveat	Use/Function
MD5	Only allowed as the PRF in TLS v1.0 and v1.1 per IG 2.4.A	SP 800-135rev1 Section 4.2.1 describes the use of MD5 in conjunction with SHA-1 in the key derivation function, concluding that the TLS 1.0/1.1 KDF may be used within the context of the TLS protocol (with provisions for validation of the companion approved functions, SHA-1 and HMAC). This use of MD5 does not conflict with the security of the approved security functions

Table 5 - Non-Approved Algorithms Allowed in the Approved Mode of Operation with No Security Claimed

When run in the Non-Approved mode all the algorithms, modes, and sizes described above are available as well as the following additional algorithms, modes, and sizes which are **not allowed to be used in the Approved mode**.

Algorithm/Function	Use/Function
AES	Encryption/decryption using the CFB64 mode
AES GCM	Encryption using external IV generation by calling init() or initExt(); Encryption/decryption using XPN by calling initXPN()
AES GCM SIV	Encrypt/decrypting using the AES GCM SIV mode
ANSI x9.63 KDF, TLS 1.3 KDF	Key derivation – using SHA-1 or SHA-3
ChaCha20	Encryption/decryption
DES	Encryption/decryption using the DES, DESX, or DES40 variants
Diffie-Hellman	Key establishment [512-1024 bits] Key generation [512-1024 bits]
DRBG	HMAC-SHA-1, HMAC-SHA2-224, HMAC-SHA2-384, HMAC-SHA2-512/256, HMAC-SHA3-224, HMAC-SHA3-384
DSA	Digital signature generation [512-1024 bits] Key generation [512-1024-bits]

Elliptic Curve Diffie-Hellman	Key establishment using keys whose size is less than 224-bits Key generation of keys with size less than 224-bits Any use of non-NIST approved curves (including Montgomery Curves) (Non-compliant less than 112 bits of encryption strength)
ECDSA	Digital signature generation using keys whose size is less than 224-bits or SHA-1 Key generation of keys with size less than 224-bits Any use of non-NIST approved curves (secp112r1, secp112r2, secp128r1, secp128r2, secp160k1, secp160r1, secp160r2, brainpoolP160r1, brainpoolP160t1, brainpoolP192r1, brainpoolP192t1, brainpoolP224r1, brainpoolP224t1, brainpoolP256r1. brainpoolP256t1, brainpoolP320r1, brainpoolP320t1, brainpoolP384r1, brainpoolP384t1, brainpoolP512r1, brainpoolP512t1, numsp256d1, numsp384d1, numsp512d1, frp256v1, sm2p256v1, gostRFC7091, gostParamSetA, gostParamSetB, curve22519, ed448, oakley1, oakley2, ipsec3, ipsec4
EDDSA	Digital signature generation using Edwards curves (Ed25519, Ed448)
PBKDF	Key derivation using key lengths less than 112 bits or salt lengths less than 128 bits
RSA	Digital signature generation using keys whose size is less than 2048-bits or SHA-1 Key generation of keys with size less than 2048-bits Key wrapping using PKCS #1 v1.5 padding Key wrapping using keys whose size is less than 2048-bits (Non-compliant less than 112 bits of encryption strength)
SHA-0	Hashing – any use (API input 0 to the SHA constructor)
SHS	Hashing – using ISC's incorrect, non-compliant, versions of SHA2-256, SHA2-384, SHA2-512, or SHA2-224 corresponding to API input 12, 13, 15, or 17 to the SHA2 constructor
Skipjack	Encryption/decryption
Triple-DES	Encryption/decryption using 1-key (API input key length of 64-bits) Encryption using 2-key (API input key length of 128-bits) Encryption/decryption using the CFB32 mode Encryption using 3-key (API input key length of 192-bits)

Table 6 - Non-Approved Algorithms Not Allowed in the Approved Mode of Operation

2.3 CDK Modes and Approval Indicators

2.3.1 Running the CDK in the Approved mode

When the CDK is run in the Approved mode, only FIPS 140-3 approved algorithms are allowed to be used. The CDK will error if a non-approved algorithm is used.

In order to operate in the Approved mode, the CO must ensure that applications loaded by the operating system call the "Configure" service (`enableFIPS()` method) at startup.

The CO may use the "Show Status" service (`isFIPS()` method) in their application to determine whether or not the CDK is operating in the Approved mode. The CO's application must provide an indication of the mode of operation by either calling the "Show Status" service (`isFIPS()` function) and outputting a custom message, or by outputting the output of the "Show Status" service (`StrVersion()` method).

2.3.2 Running the CDK in the Non-Approved Mode

If the "Configure" service (`enableFIPS()` function) is not called, the CDK will operate in the Non-Approved mode. The "Show Status" service (`isFIPS()` function) will return false to indicate that the CDK is not operating in the Approved mode. The output of the "Show Status" service

(`StrVersion()` function) will not include the statement that the module is operating in the Approved mode.

2.3.3 Running the CDK in Degraded Mode

The CDK does not support a degraded mode of operation.

2.3.4 Approval Indicators

As noted above, the isFIPS() function will return true or false to indicate whether or not the CDK is operating in the Approved mode and only allowing approved algorithms to be used. Additionally, in the Approved or Non-Approved mode individual algorithms can be queried:

- by calling the isFIPS() member method, if the algorithm is represented by a C++ class
- by calling isHashFIPS() or isKDFFIPS() with appropriate inputs

2.4 Cryptographic Boundary

The following diagram (Figure 1) illustrates the TOEPP and the relationship between a typical software application (such as the supplied CDK test program), the ISC CDK, the computer's operating system, the system's BIOS, and the physical general-purpose computer (GPC) on which it all executes. The cryptographic boundary is the ISC CDK shared library itself as shown inside of the red dashed lines labels cryptographic boundary.

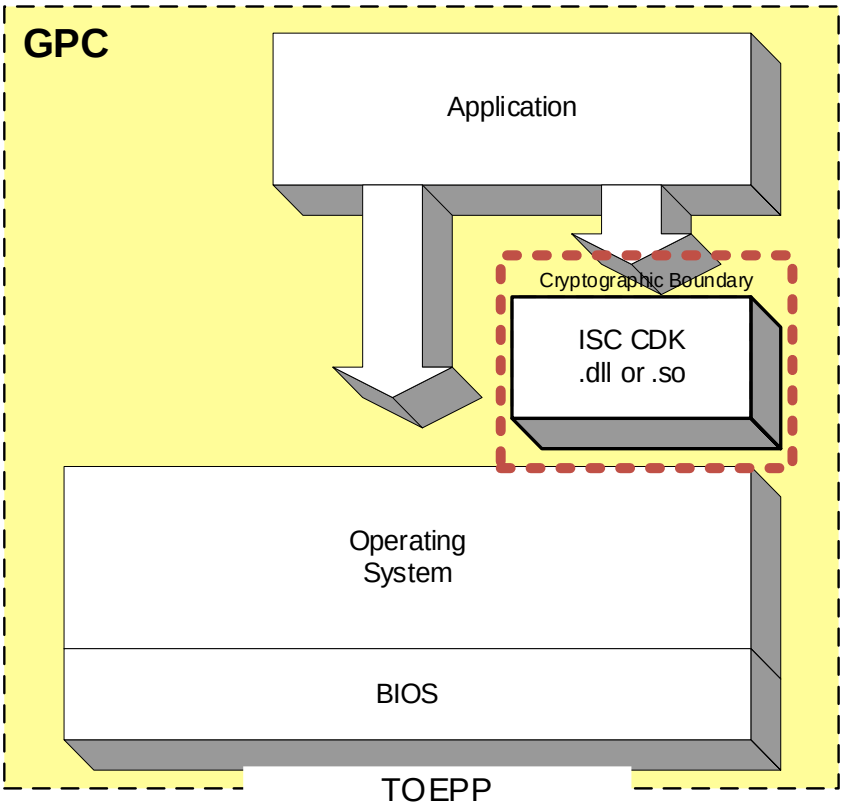

Figure 1 — Cryptographic Boundary

The following diagram (Figure 2) is a block diagram displaying the most important components of the CDK software. (Certain dependencies between the various components are suppressed for simplicity.)

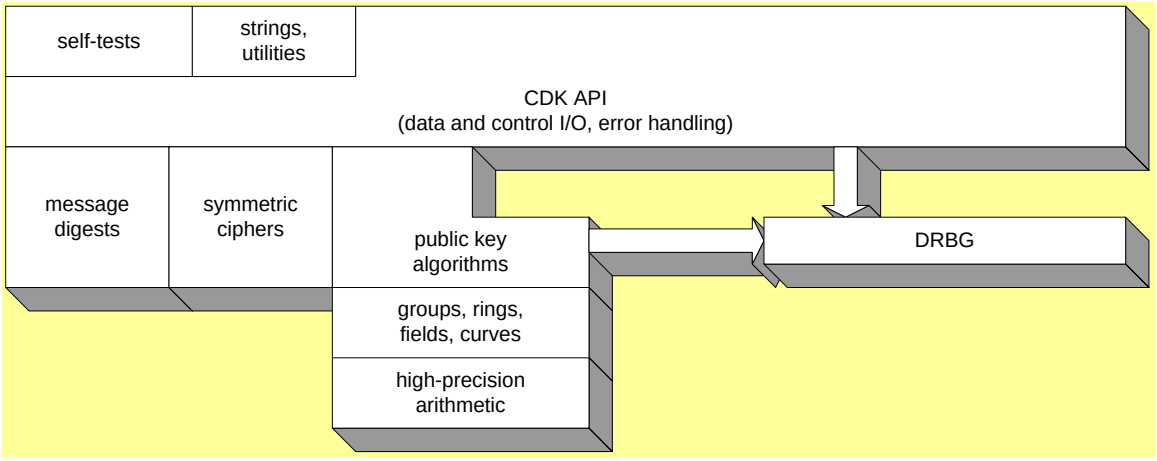

Figure 2 — Important Components of the CDK

3. Cryptographic Module Interfaces

As a FIPS 140-3 multi-chip standalone module, the CDK has a physical power interface and physical input and output data paths, which are the computer system's standard input/output ports and power interface. The input/output ports on the computer are used for connecting external devices such as monitors and keyboards however these devices are outside the cryptographic boundary of the CDK. The CDK does not support a control output interface.

The CDK software is written in C++; its logical interfaces are the application program interfaces (API) defined by C++ classes and global methods. The calling program inputs control and data to the CDK through the input fields of the API and receives output data and/or status information through the output parameters of the API. Vendor documentation describes what output indicates an error and what output constitutes successful completion of the operation.

A "show status" service is provided by the static `Algorithm::isErrorState(), isFIPS()`, and `StrVersion()` methods which may be called at any time to determine if the CDK is in the hard error state and whether or not the CDK is operating in the Approved mode. If the CDK enters the hard error state, an error code is returned through the API interface, and no data output is returned.

Methods performing key generation do not output intermediate key values. Methods performing key zeroization only return status output describing success or failure of the operation.

Below is a table that maps the logical interfaces to the physical interfaces, with the exclusion of the Control Output interface and Power Interface which the module does not support:

Physical Port	Logical Interface	Data that passes over port/interface
Standard Input Port (e.g. Keyboard)	Data Input	Data passed to the API calls to be used by the Module
Standard Output Port (e.g. Monitor)	Data Output	Data returned from API calls, generated by the Module
N/A	Control Input	API calls
Standard Output Port (e.g. Monitor)	Status Output	C++ exceptions, the Algorithm::isErrorState() function, the isFIPS() function, and the StrVersion() function

Table 7 – Ports and Interfaces

4. Roles, Services, and Authentication

The CDK module supports one role: “Crypto-Officer” (*CO*). The CO is the human being who configures an application that uses services provided by the CDK.

The CDK provides no maintenance access interface and therefore does not support a *Maintenance* role. FIPS 140-3 Level 1 cryptographic modules are not required to employ authentication as a means of controlling access to the module. Such authentication mechanisms are not supported by the CDK for the CO role. No other roles are supported.

The CO configures the computer system, operating system, and the application using services provided by the CDK to operate in a secure Approved mode, if that is desired (this may include configuring the system on which the application is installed as part of the installation process). Additional conditions for meeting FIPS 140-3 requirements are provided in a separate document: Crypto Officer’s Guide.

Self-test services are described in Section 9 of this document.

Bypass services are not provided.

Tables 8, 9 and 10 provide details on the services available to each role, and each role’s access rights with respect to those services.

Role	Service	Input	Output
CO	Configure	Configuration Parameters	Status
CO	Integrity Self-Test	None	Status
CO	Perform Self-Tests	None	Status
CO	Show Status	None	Status
CO	Zeroize	Key	Status
CO	Symmetric Key Generation using DRBG	Key Size	Key, Status
CO	Random Number Generation	Size	Value, Status
CO	Asymmetric Key Generation	Key Type, Size	Key, Status
CO	Symmetric Encrypt/Decrypt	Key, Data	Ciphertext, Status
CO	Symmetric Digest	Key, Data	Digest Value, Status
CO	Message Digest	Data	Digest Value, Status
CO	Keyed Hash	Key, Data	Digest Value, Status
CO	Key Agreement	Keys	Shared Secret, Status
CO	Key Transport	Keys	Ciphertext, Status
CO	Digital Signature	Key, Digest Value, Digest Type	Signature, Status
CO	Signature Verification	Key, Signature, Message	Status
CO	Extendable Output Function	Data	Extended Output Value, Status
CO	Key Derivation	Key, Shared Secret, Password, Data	Key, Status

Table 8 - Roles, Service Commands, Input and Output

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and/or SSPs	Indicator
Configure	Initialize and configure module	All algorithms and modes	All SSPs	CO	R, W	0 upon Success
Integrity Self-Test	Check module integrity	HMAC-SHA2-256 #A5798	HMAC Integrity Key	CO	None	0 upon Success

Perform Self-Tests	Check module algorithm correctness	See Section 10 for algorithms tested.	None	CO	None	0 upon Success
Show Status	Return codes and/or strings[8]	None	None	CO	None	0 upon Success
Zeroize	Erase key or critical security parameter	None	All SSPs	CO	Z	0 upon Success
Symmetric Key Generation using DRBG	Generate a random key	Conditioning Component AES-CBC-MAC, AES-CCM, AES-CFB128, AES-CFB8, AES-CMAC, AES-CTR, AES-ECB, AES-GCM, AES-OFB, AES-XTS Testing Revision 2.0, CKG, HMAC DRBG, HMAC-SHA2-256, HMAC-SHA2-512, TDES-CBC, TDES-CFB64, TDES-CFB8, TDES-CTR, TDES-ECB, TDES-OFB, ESV #A5798	AES GCM IV, AES GCM Key, AES Key, AES XTS Key, DRBG Key Value, DRBG Seed, DRBG 'V' Value, Entropy Input String, MAC Key, Triple-DES Key	CO	G, R, E	0 upon Success
Random Number Generation	Generate a random number	HMAC DRBG, ESV #A5798	DRBG Key Value, DRBG Seed, DRBG 'V' Value, Entropy Input String	CO	E	0 upon Success
Asymmetric Key Generation	Generate an asymmetric public and private key	CKG, DSA PQGGen [FIPS186-4], DSA PQGVer [FIPS186-4], ECDSA KeyGen [FIPS186-4], ECDSA KeyVer [FIPS186-4], HMAC DRBG, RSA KeyGen [FIPS186-4], ESV #A5798	DRBG Key Value, DRBG Seed, DRBG 'V' Value, DSA Public Key, ECC DH Private Key, ECC DH Public Key, ECDSA Private Key, ECDSA Public Key, RSA Signature Private Key, RSA Signature Public Key	CO	G, R, E	0 upon Success
Symmetric Encrypt/Decrypt	Encrypt/decrypt data using a symmetric algorithm	AES-CBC, AES-CCM, AES-CFB128, AES-	AES GCM IV, AES GCM Key, AES Key, AES	CO	E	0 upon Success

[8] The Show Status service also satisfies the "Show Module Version Information" Service.

		CFB8, AES-CTR, AES-ECB, AES-GCM, AES-OFB, AES-XTS Testing Revision 2.0, TDES-CBC, TDES-CFB64, TDES-CFB8, TDES-CTR, TDES-ECB, TDES-OFB #A5798	XTS Key, Triple-DES Key			
Symmetric Digest	Digest data	Conditioning Component AES-CBC-MAC, AES-CMAC #A5798	AES MAC Key	CO	E	0 upon Success
Message Digest	Digest data	ParallelHash-128, ParallelHash-256, SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512, TupleHash-128, TupleHash-256 #A5798	None	CO	None	0 upon Success
Keyed Hash	Digest data	HMAC-SHA-1, HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512, HMAC-SHA2-512/224, HMAC-SHA2-512/256, HMAC-SHA3-224, HMAC-SHA3-256, HMAC-SHA3-384, HMAC-SHA3-512, KMAC-128, KMAC-256 #A5798	MAC Key	CO	E	0 upon Success
Key Agreement	Derive a shared key	KAS-ECC-SSC, KAS-IFC-SSC #A5798	ECC DH Private Key, ECC DH Public Key, Shared Secret value	CO	G, R, E	0 upon Success
Key Transport	Encrypt a data encryption key with a key encryption key	AES-KW, AES-KWP, KTS-IFC #A5798	AES Key Wrap key, RSA Key Wrap Private Key, RSA Key Wrap Public Key	CO	R, E	0 upon Success

Digital Signature	Create a digital signature	ECDSA SigGen [FIPS186-4], Deterministic ECDSA SigGen [FIPS186-5], RSA SigGen [FIPS186-4], RSA Signature Primitive (CVL) #A5798	ECDSA Private Key, RSA Signature Private Key	CO	E	0 upon Success
Signature Verification	Verify a digital signature	DSA SigVer [FIPS186-4], ECDSA SigVer [FIPS186-4], RSA SigVer [FIPS186-4] #A5798	DSA Public Key, ECDSA Public Key, RSA Signature Public Key	CO	E	0 upon Success
Extendable Output Function	Extend bit strings to any desired length	cSHAKE-128, cSHAKE-256, SHAKE-128, SHAKE-256, ParallelHash-128, ParallelHash-256 #A5798	None	CO	None	0 upon Success
Key Derivation	Derive a key	KDA HKDF, KDA OneStep, KDF ANS 9.63 (CVL), KDF IKEv1 (CVL), KDF IKEv2 (CVL), KDF SNMP (CVL), KDF SSH (CVL), KDF TLS[9] (CVL), KDF TPM (CVL), PBKDF, TLS v1.2 KDF RFC7627, TLS v1.3 KDF #A5798	Shared Secret value, Password	CO	G, R, E	0 upon Success

Table 9 - Approved Services

[9] MD5 (No Security Claimed) is only used for TLS 1.0/1.1 KDF.

Service	Description	Algorithms Accessed	Roles	Indicator
Symmetric Key Generation using DRBG	Generate a random key for usage in a non-Approved algorithm	DRBG	CO	0 upon Success (when in the Non-Approved mode) CDK_OP_UNSUPPORTED in the Approved mode
Random Number Generation	Generate a random number	DRBG (with HMAC-SHA-1, HMAC-SHA2-224, HMAC-SHA2-384, HMAC-SHA2-512/256, HMAC-SHA3-224, HMAC-SHA3-384)	CO	0 upon Success (when in the Non-Approved mode) CDK_OP_UNSUPPORTED in the Approved mode
Asymmetric Key Generation	Generate an asymmetric public and private key for usage in a non-Approved algorithm	DRBG	CO	0 upon Success (when in the Non-Approved mode) CDK_OP_UNSUPPORTED in the Approved mode
Symmetric Encrypt/Decrypt	Encrypt/decrypt data using a symmetric algorithm	AES (CFB64, GCM with external IV, XPN, GCM SIV), ChaCha20, DES, DESX, DES40, Skipjack, Triple-DES (encryption, 2-key encryption / decryption, or CFB32 mode)	CO	0 upon Success (when in the Non-Approved mode) CDK_OP_UNSUPPORTED in the Approved mode
Message Digest	Digest data	SHA-0, or using ISC's incorrect, non-compliant, versions of SHA2-256, SHA2-384, SHA2-512, or SHA2-224 corresponding to API input 12, 13, 15, or 17 to the SHA2 constructor	CO	0 upon Success (when in the Non-Approved mode) CDK_OP_UNSUPPORTED in the Approved mode
Key Agreement	Derive a shared key	Diffie-Hellman, Elliptic Curve Diffie-Hellman (using non-NIST approved curves, or sizes less than 224 bits)	CO	0 upon Success (when in the Non-Approved mode) CDK_OP_UNSUPPORTED in the Approved mode
Key Transport	Encrypt a data encryption key with a key encryption key	RSA key wrapping using PKCS #1 v1.5 padding as shown in section 8.1 of RFC 2313, or a modulus size that is less than 2048-bits	CO	0 upon Success (when in the Non-Approved mode) CDK_OP_UNSUPPORTED in the Approved mode
Digital Signature	Create a digital signature	DSA signature generation, ECDSA (using non-NIST approved curves, or sizes less than 224 bits or SHA-1), EdDSA, RSA using a modulus size less than 2048-bits or SHA-1	CO	0 upon Success (when in the Non-Approved mode) CDK_OP_UNSUPPORTED in the Approved mode
Key Derivation	Derive a key	ANSI x9.63 KDF using SHA-1 or SHA-3, TLS 1.3 KDF using SHA-1 or SHA-3, PBKDF using key lengths less than 112 bits or salt lengths less than 128 bits	CO	0 upon Success (when in the Non-Approved mode) CDK_OP_UNSUPPORTED in the Approved mode

Table 10 - Non-Approved Services

5. Software/Firmware Security

The CDK integrity is checked on startup using HMAC-SHA2-256 as described in section Pre-Operational Tests. The integrity check can be initiated on demand using the mechanisms specified in section On-

Demand Self-Tests. The module executable is provided in the compiled form described in section Platform Availability below.

6. Operational Environment

The CDK is a software module that operates in a modifiable operational environment running on a general-purpose computer. The CDK is a single shared library.

Within the tested environments user processes are segregated into their own process space. Processes are logically separated from all other processes by the operating system and underlying hardware. As the module exists within the process space of the calling application, acting in the Crypto Officer role, and no other process can access the same instance of the module, the module operates in single user mode.

6.1 Platform Availability

The CDK software was designed for use on a variety of operating systems and hardware platforms. For FIPS 140-3 validation purposes, operational testing was performed on the tested platforms listed in section 2.1.

The CDK software is provided as compiled code in the form of shared link libraries that can be run on Microsoft Windows (CDKC8123S.DLL) and Linux (libcdkc.so.81.2.3) operating systems. There are no security rules, settings, or restrictions to the configuration of the operational environment.

The module's application programming interface (API), which provides access to the supported cryptographic primitives, consists of a set of C++ classes as documented in 'cdk_fips.h', the other header files referenced therein, and related documentation. For FIPS 140-3 validation, the CDK was loaded by multiple test applications (one for each algorithm family) and executed on each of the supported platforms.

7. Physical Security

The module is a software-only module and the physical security requirements of FIPS 140-3 level 1 do not apply.

8. Non-Invasive Security

Non-invasive security is Non Applicable as there are currently no requirements in SP 800-140F.

9. Sensitive Security Parameters Management

The CDK uses, creates, and/or manages:

- symmetric keys (for use with a symmetric cipher or keyed hash functions), and
- asymmetric key pairs (for digital signatures and key transport/agreement protocols based on public key schemes)

9.1 Storage Areas

The CDK is a low-level cryptographic toolkit and does not provide any key storage. As detailed in section Pre-Operational Tests, a single, special purpose, integrity key is hard coded in the module in plaintext form and is used to verify the integrity of the module.

9.2 SSP Input-Output Methods

The CDK does not manage any manually distributed cryptographic keys, either entry or output, external to the cryptographic boundary. However, the logical C++ API exposed by the CDK provides methods for loading and unloading symmetric keys and public/private key pairs in electronic form for manual[10] key distribution by the application.

9.3 SSP Zeroization Methods

An instantiated CDK object may contain a cryptographic key during its lifetime. Such keys are available to the user for manipulation, but when the object is released, its memory and all keys in it are cleared. Under normal operations all internal memory allocated by the CDK for temporary key storage is zeroized when the object owning that memory is destroyed. The CO is responsible for ensuring that CDK objects are destroyed properly (i.e., the application must allow the C++ destructors to be called by properly exiting the application or by deleting all heap allocated CDK objects before application termination). To zeroize the special purpose integrity key embedded in the CDK in plaintext form, the CDK shared library must be securely erased from the hard disk.

9.4 SSPs

Listed in Table 11 are the keys and SSPs used by the module in the Approved mode.

Key/SSP Name/ Type	Strength	Security Function and Cert. Number	Gener-ation	Import/ Export	Establish-ment	Storage	Zeroization	Use & related keys
AES Key (CSP)	Between 128 and 256	AES-CBC, AES-CCM, AES-CFB128, AES-CFB8, AES-CTR, AES-ECB,	DRBG	Import/export via API (plaintext)	Agreement, Transport, Generation, Entry, Derivation	Plaintext in RAM	Implicit zeroization when object is deallocated or reboot	AES encrypt/decrypt key

[10] IG 9.5.A MD/EE - CM Software to/from App via TOEPP Path

		AES-OFB, Conditioning Component AES-CBC-MAC #A5798						
AES MAC Key (CSP)	Between 128 and 256	AES-CMAC #A5798	DRBG	Import/export via API (plaintext)	Agreement, Transport, Generation, Entry, Derivation	Plaintext in RAM	Implicit zeroization when object is deallocated or reboot	AES CMAC generate/verify key
AES GCM IV[11] (PSP)	96 (random) or 8-1024 (counter)	AES-GCM #A5798	Internally using a counter or random value	Import/export via API (plaintext)	None	Plaintext in RAM	Implicit zeroization when object is deallocated or reboot	AES GCM initialization vector
AES GCM Key (CSP)	Between 128 and 256	AES-GCM #A5798	DRBG	Import/export via API (plaintext)	Agreement, Transport, Generation, Entry, Derivation	Plaintext in RAM	Implicit zeroization when object is deallocated or reboot	AES GCM encrypt/decrypt /generate/verify key
AES Key Wrap key (CSP)	Between 128 and 256	AES-KW, AES-KWP #A5798	DRBG	Import/export via API (plaintext)	Agreement, Transport, Generation, Entry, Derivation	Plaintext in RAM	Implicit zeroization when object is deallocated or reboot	AES encrypt/decrypt key
AES XTS Key (CSP)	128 or 256	AES-XTS Testing Revision 2.0 #A5798	DRBG	Import/export via API (plaintext)	Agreement, Transport, Generation, Entry, Derivation	Plaintext in RAM	Implicit zeroization when object is deallocated or reboot	AES XTS encrypt/decrypt key
DSA Public Key (PSP)	112 or 128	DSA SigVer [FIPS186-4] #A5798	N/A	Import/export via API (plaintext)	None	Plaintext in RAM	Implicit zeroization when object is deallocated or reboot	DSA signature verification public key
ECC DH Private Key (CSP)	Between 112 - 256	KAS-ECC-SSC #A5798	Internally using the DRBG	Import/export via API (plaintext)	None	Plaintext in RAM	Implicit zeroization when object is deallocated or reboot	ECC DH private key agreement key
ECC DH Public Key (PSP)	Between 112 - 256	KAS-ECC-SSC #A5798	Internally computed based on the private key	Import/export via API (plaintext)	None	Plaintext in RAM	Implicit zeroization when object is deallocated or reboot	ECC DH public key agreement key
ECDSA Private Key (CSP)	Between 112 - 256	ECDSA SigGen, Deterministic ECDSA SigGen [FIPS186-4],	Internally using the DRBG	Import/export via API (plaintext)	None	Plaintext in RAM	Implicit zeroization when object is deallocated or reboot	ECDSA signature generation private key

[11] The AES-GCM IV is generated internally randomly or as a counter per IG C.H. In the former case the IV is exactly 96-bits. In the latter case the IV may be 8- to 1024-bits in length.

		ECDSA KeyGen [FIPS186-4] #A5798						
ECDSA Public Key (PSP)	Between 112 - 256	ECDSA SigVer, ECDSA KeyVer [FIPS186-4], ECDSA SigVer [FIPS186-4] #A5798	Internally computed based on the private key	Import/export via API (plaintext)	None	Plaintext in RAM	Implicit zeroization when object is deallocated or reboot	ECDSA signature generation public key
MAC Key (CSP)	112 minimum	HMAC-SHA-1, HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512, HMAC-SHA2-512/224, HMAC-SHA2-512/256, HMAC-SHA3-224, HMAC-SHA3-256, HMAC-SHA3-384, HMAC-SHA3-512, KMAC-128, KMAC-256 #A5798	DRBG	Import/export via API (plaintext)	Agreement, Transport, Generation, Entry, Derivation	Plaintext in RAM	Implicit zeroization when object is deallocated or reboot	Keyed hash key
HMAC Integrity Key (Non-SSP)	192	HMAC-SHA2-256 #A5798	N/A	None	None	Plaintext on Disk (embedded in the shared library)	Explicit zeroization when securely erasing the CDK library from disk	Keyed hash key to verify the integrity of the module at startup and on demand
RSA Key Wrap Private Key (CSP)	Between 112 and 192	KAS-IFC-SSC, KTS-IFC, RSA KeyGen [FIPS186-4] #A5798	Internally using the DRBG	Import/export via API (plaintext)	None	Plaintext in RAM	Implicit zeroization when object is deallocated or reboot	Private component of an RSA key pair
RSA Key Wrap Public Key (PSP)	Between 112 and 192	KAS-IFC-SSC, KTS-IFC #A5798	Internally computed based on the private key	Import/export via API (plaintext)	None	Plaintext in RAM	Implicit zeroization when object is deallocated or reboot	Public component of an RSA key pair
RSA Signature Private Key (CSP)	Between 112 and 192	RSA SigGen [FIPS186-4], RSA Signature Primitive #A5798	Internally using the DRBG	Import/export via API (plaintext)	None	Plaintext in RAM	Implicit zeroization when object is deallocated or reboot	Private component of an RSA key pair
RSA Signature	Between 112 and 192	RSA SigVer [FIPS186-4],	Internally computed based on	Import/export via API (plaintext)	None	Plaintext in RAM	Implicit zeroization when object is	Public component of an RSA key pair

Public Key (PSP)		RSA Signature Primitive #A5798	the private key				deallocated or reboot	
Triple-DES Key (CSP)	112	TDES-CBC, TDES-CFB64, TDES-CFB8, TDES-CTR, TDES-ECB, TDES-OFB #A5798	DRBG	Import/export via API (plaintext)	Agreement, Transport, Generation, Entry, Derivation	Plaintext in RAM	Implicit zeroization when object is deallocated or reboot	Triple-DES (3-Key) decrypt key for legacy use only Triple-DES (2-Key) decrypt key for legacy use only
DRBG 'V' Value (CSP)	128	HMAC DRBG, ESV #A5798	Internally using entropy input	None	None	Plaintext in RAM	Implicit zeroization when object is deallocated or reboot	DRBG internal state values
DRBG Key Value (CSP)	256	HMAC DRBG, ESV #A5798	Internally using entropy input	None	None	Plaintext in RAM	Implicit zeroization when object is deallocated or reboot	DRBG internal state values
DRBG Seed (CSP)	384	HMAC DRBG, ESV #A5798	Internally using entropy input	None	None	Plaintext in RAM	Implicit zeroization when object is deallocated or reboot	Entropy input (length is platform dependent but always greater than 256-bits)
Entropy Input String (CSP)	384	HMAC DRBG, ESV #A5798	Entropy as per SP 800-90B	None	None	Plaintext in RAM	Implicit zeroization when object is deallocated or reboot	Entropy input string coming from the entropy source. Input length = 384 bits
Shared Secret Value (CSP)	Between 112 - 256	KDA HKDF, KDA OneStep, KDF ANS 9.63, KDF IKEv1, KDF IKEv2, KDF SNMP, KDF SSH, KDF TLS, TLS v1.2 KDF RFC7627, TLS v1.3 KDF, KDF TPM #A5798	N/A	Import/export via API (plaintext)	Agreement	Plaintext in RAM	Implicit zeroization when object is deallocated or reboot	Shared secret values from key agreement
Password (CSP)	8 - 128	PBKDF #A5798	N/A	Import/export via API (plaintext)	None	Plaintext in RAM	Implicit zeroization when object is deallocated or reboot	Passwords used in PBKDF

Table 11 – SSPs

9.5 Entropy Sources

The CDK includes a non-physical entropy source within the module boundary which complies with SP 800-90B and has been validated using the guidance set out in the FIPS 140-3 Implementation Guidance.

Entropy Source	Minimum number of bits of entropy	Details
ISC CDK Jitter Entropy Component	384-bits obtained per request	The ISC CDK incorporates the Jitter Entropy source for seeding for the DRBG. The underlying noise source is expected to be able to provide at least 1 bit of entropy per 64-bit sample. Due to the sampling rate and conditioning applied, the entropy source provides 1 bit of entropy per each bit of conditioned output.

Table 12 - Non-Deterministic Random Number Generation Specification

9.6 RNGs and Output

The CDK generates keys for the approved and vendor affirmed algorithms listed in section Algorithms and Parameters Allowed in the Approved mode. The CDK also generates Non-Approved keys for algorithms listed in section Non-Approved Algorithms Allowed in the Approved Mode of Operation

The CDK does not support any non-approved algorithms that are allowed in the Approved mode.

Non-Approved-mode Algorithms The CDK can generate symmetric keys (for a symmetric cipher or keyed hash function) using its DRBG. To generate key pairs, the public key generation methods use the CDK's random number generator. The calling application is responsible for maintaining the SSPs that it establishes using the module, inclusive of assuring that SSPs established while operating in the Approved mode are not shared or used in the Non-Approved mode (and vice versa).

9.7 Key Distribution

The CDK doesn't perform key distribution. The CDK has basic cryptographic functions which can be used by developers to build key distribution capabilities into their applications. The key distribution techniques available for use include RSA Key Establishment, ECC Diffie-Hellman Key Agreement, and AES key wrapping.

10. Self-Tests

The CDK performs self-tests to ensure that it is functioning properly. If the message digest value computed over the CDK does not match the embedded expected value, or if an algorithm KAT fails, then the module enters the hard error state and no further cryptographic operations are possible. To recover from the hard error state, the application using the CDK's services must be restarted.

The CDK returns non-zero error codes from its API to indicate failure. Most error codes are output when the CDK transitions through the soft error state (which is immediately and automatically cleared), but there are two special error codes that the CDK returns to indicate it has entered the hard error state. The CDK returns `CDK_ERROR_STATE` with value `1470` from its interfaces when it is in the hard error state for any reason other than a pairwise key test failure. The CDK returns `CDK_KEYPAIR_INCONSISTENT` with value `1234` when a pairwise key test fails during an on-demand self-test and the CDK enters the hard error state. Additionally, a static function, `Algorithm::isErrorState()`, may be called to determine if the CDK is in the hard error state. Table 13 contains details on the error states.

Name	Description	Conditions	Recovery Method	Indicator
Soft Error	This transitional state represents non-critical errors, such as invalid input to a function in the library.	Function returns one of: CDK_FAILED CDK_INTERNAL_ERR CDK_CALLBACK_FAILED CDK_INVALID_PTR CDK_INVALID_CTX CDK_INVALID_DATA CDK_INVALID_DATA_LENGTH CDK_INVALID_KEY CDK_INVALID_KEY_PTR CDK_INVALID_KEY_LENGTH CDK_INVALID_SIGNATURE CDK_INVALID_DIGEST CDK_INVALID_DIGEST_ALG CDK_INVALID_ALG CDK_INVALID_MODE CDK_INVALID_PADDING CDK_INVALID_IV_SIZE CDK_INVALID_IV CDK_INVALID_KEY_SIZE CDK_INVALID_ROUNDS CDK_INVALID_PARAM_LENGTH CDK_INVALID_KEYTYPE CDK_INVALID_KEYUSAGE CDK_INVALID_ITERATION_COUNT CDK_INVALID_SALT CDK_INVALID_RANDOM CDK_INVALID_SEED CDK_INVALID_ALG_PARAMS CDK_INVALID_PUB_EXPONENT CDK_INVALID_TAG CDK_INVALID_TYPE CDK_INPUT_LENGTH_ERR CDK_INPUT_DATA_ERR CDK_OP_UNSUPPORTED	Automatic as the CDK returns to the operational state immediately after returning the error code	Function call returns a non-zero value as listed in Conditions column

		CDK_OP_FAILED CDK_PRVKEY_CANNOT_FIND CDK_KEYGEN_FAILED CDK_PUBKEY_CANNOT_FIND CDK_KEY_INVALID CDK_KEY_INVALID_USAGE CDK_KEY_INVALID_KDF CDK_KEY_INVALID_PARTYID CDK_MODE_UNSUPPORTED CDK_KEY_LENGTH_UNSUPPORTED CDK_NO_KEY CDK_OPERATION_NOT_INITALIZED CDK_RESEED_REQUIRED CDK_NO_ENTROPY CDK_INVALID_BLOCK_SIZE CDK_PARSE_ERROR CDK_INVALID_KEY_TOO_MANY_PRIMES CDK_UNKNOWN_OID		
Hard Error	This state represents critical errors such as failure of the CDK's pseudo-random number generator or failure of an on-demand self-test.	Pre-operational test failure Conditional self-test failure NIST SP 800-90A Health Tests failure Pair-wise self-test failure SP 800-90B Health Tests failure IG C.I XTS-AES Test failure On-demand self-test failure	Restart the application using the CDK's services	Algorithm::isErrorState() returns true, function call returns CDK_ERROR_STATE, or function call returns CDK_KEYPAIR_INCONSISTENT

Table 13 – Error States

10.1 Pre-Operational Tests

When the CDK module is loaded from disk by the operating system, it executes a pre-operational *software integrity test* as well as conditional *cryptographic algorithm self-tests*. Basic self-test and library verification is performed at library load by using a C++ static constructor to call the self-test and integrity test methods in Table 14 – Self-Tests. The conditional tests include known answer tests (KATs) or pair-wise consistency tests (PCTs) for each of the Approved algorithms in the CDK (see Section 2.2.1). Table 14 – Pre-Operational Self-Tests also lists those algorithms whose cryptographic algorithm self-test is run on first use as opposed to at module start.

The integrity test operates by calculating a 256-bit HMAC (HMAC-SHA2-256) over the module and comparing it to an expected value embedded (along with the key) in the module itself at the factory. These tests are performed at startup regardless of whether the module is put into the Approved mode. If the software integrity test fails or if any self-test fails, the module displays a message on the output interface, enters the error state, and inhibits all cryptographic services. The integrity test executes after the HMAC KATs on which it relies. On all platforms but Microsoft Windows, the integrity test is performed over the entire module binary except for the seventy-two (72) byte MAC field. On Microsoft Windows, the integrity test is performed over the entire module binary except for the seventy-two (72) byte MAC field and the Windows PE header IMAGE_DIRECTORY_ENTRY_SECURITY field that contains (or

will contain) a digital signature and certificate applied by Microsoft tooling post the population of the MAC integrity field.

Algorithm	Type	Description
HMAC-SHA2-256	CAST	Known Answer Test done prior to the Software Integrity Test as per IG 10.2.A
HMAC-SHA2-256	SW Integrity	Software integrity test using HMAC-SHA2-256 for CDK

Table 14 – Pre-Operational Self-Tests

10.2 Conditional Self-Tests

Conditional self-tests are performed when certain specific conditions arise within the CDK. The conditional self-tests are described in the following paragraphs.

If any conditional self-test fails, the module displays a message on the output interface, enters the error state, and inhibits all cryptographic services.

Interruption of the module's operation results in the module terminating and causes the module to be unloaded from memory. This most likely occurs when the application using the module terminates. When the application using the module is started again, the module is loaded in the initial state, the start-up self-tests will run, the integrity test will run, and any conditional test states (*i.e.,* counters) will be reset.

Algorithm	Type	Description
AES-CBC	CAST	Encrypt and Decrypt; Key Size: 128
AES CCM	CAST	Encrypt and Decrypt; Key Size: 128
AES CMAC	CAST	Generate; Key Size: 128
AES GCM	CAST	Encrypt and Decrypt; Key Size: 128
AES XTS	CAST	Encrypt and Decrypt AES-256
AES-ECB	CAST	Encrypt and Decrypt; Key Size: 128
Deterministic ECDSA	CAST	Sign and Verify using P-256, SHA2-256
HMAC DRBG	CAST	SHA2-256/512; (with and without PR)
DRBG Health Tests	CAST	Instantiate/generate/reseed health checks
DSA	CAST	Verify using 2048-bit key, SHA2-256
ECC CDH	CAST	Primitive "Z" computation using two P-256 keys
ECDSA	CAST	Sign and verify using P-256, SHA2-256, Sign and verify using B-233, SHA2-256
HKDF	CAST	SHA2-256
HMAC	CAST	One KAT each: SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512 (also covers SHA-1, SHA-2, and SHA-3 algorithms)
IKEv1 KDF	CAST	SHA2-256
IKEv2 KDF	CAST	SHA2-256
KAS-IFC-SSC	CAST	RSASVE.GENERATE and RSASVE.RECOVER using 2048-bit key

KDA OneStep SP800-56Cr2	CAST	SHA2-256
KDF ANSI X9.63	CAST	SHA2-256
PBKDF	CAST	SHA2-256
RSA	CAST	Sign and verify using 2048-bit key, SHA2-256, PKCS#1v1.5
SHA-3	CAST	One KAT each: SHAKE-128/256, cSHAKE-128/256, KMAC-256, TupleHash-128, and ParallelHash-128
SNMP KDF	CAST	Password Length: 64 and 8192
SSH KDF	CAST	SHA2-256
TLS 1.1 KDF	CAST	SHA2-256, SHA2-384, and SHA2-512
TLS 1.2 KDF	CAST	SHA2-256, SHA2-384, and SHA2-512
TLS 1.3 KDF	CAST	SHA2-256
TPM KDFs	CAST	SHA2-256
Triple-DES	CAST	2-key Triple-DES decrypt-only and 3-key Triple-DES decrypt-only

Table 15 – Conditional Self-Tests

10.2.1 Random Number Tests

10.2.1.1 NIST SP 800-90A Health Tests

The DRBG KAT detailed in Table 14 – Pre-Operational Self-Tests exercises the DRBG's instantiate, reseed, and generate functions and covers the required health tests specified in SP 800-90A, Section 11.3.

SP 800-90A, Section 11.3 also requires that the generate function be tested at reasonable intervals. In the CDK, the self-test interval for calls to the generate function is 32,768 and was chosen arbitrarily: every 32,768th call to the generate function causes the DRBG to run its self-tests again.

10.2.2 Pair-Wise Self-Tests

All ECDSA public/private key pairs are automatically tested for pair-wise consistency upon generation by generating a signature and verifying the signature for an embedded message.

All Elliptic Curve Diffie-Hellman public/private key pairs are automatically tested for pair-wise consistency upon generation by computing a shared secret, deriving the key, and encrypting a message and then decrypting the message.

All RSA key pairs are automatically tested for pair-wise consistency upon generation by generating a signature and verifying the signature over, and by encrypting and decrypting, an embedded message.

Algorithm	Type	Description
ECDSA	PCT	Key-Pair Generation and Key Import
RSA	PCT	Key-Pair Generation and Key Import
KAS-ECC	PCT	Key-Pair Generation and Key Import

Table 16 – Pair-Wise Consistency Tests

10.2.3 SP 800-90B Health Tests

As required by SP800-90B the CDK's jitter entropy component implements a continuous Repetition Count Test and a continuous Adaptive Proportion Test. If either test fails, entropy cannot be obtained, the operation is aborted, and the CDK enters the error state.

10.2.4 IG C.I XTS-AES Test

As required per IG C.I, when the XTS-AES object is initialized by an operator the CDK ensures that the key and tweak values are not identical. If they are identical, the CDK returns error code 1038, CDK_INVALID_KEY from its API.

10.2.5 On-Demand Self-Tests

As documented in the Crypto Officer's Guide and the User's Guide, the CO may, on-demand, invoke any of the self-tests listed in Table 14 – Pre-Operational Self-Tests to ensure the integrity of specific algorithms by configuring their application to call the desired self-test function in the API (*e.g.*, `ISC_CDK::Test_SHA256()`). There is also a master test function (`ISC_CDK::SelfTest()`) that the CO may call to run all self and integrity tests.

11. Life-Cycle Assurance

11.1 Finite State Model

The CDK was designed around a Finite State Model (FSM) that is detailed in a proprietary document submitted with this security policy.

11.2 Delivery and Operation and Guidance Documents

The ISC CDK is delivered to the CO as a compressed file archive (zip or tar) electronically or on physical media for each target platform with the version and platform indicated in the package name.

- cdk8.1.Y.X.win.x86_64.zip
- cdk8.1.Y.Z.lin.x64.tgz

The ISC CDK is delivered in binary form as a pre-compiled shared library. The deliverables also include C-header files, crypto officer's guide, and FIPS 140-3 security policy.

When a CO receives the module, in .zip or .tgz form, they should extract the contents on to the computer in a folder of their choice and then follow the user's guide to integrate it into applications. The CO is responsible for installing the ISC CDK on systems as part of their own application's installation.

12. Mitigation of Other Attacks

The CDK has not been designed to mitigate any specific attacks. The CDK does not employ any mitigation techniques against non-invasive attacks.

13. Acronyms

Acronym	Meaning
AES	Advanced Encryption Standard
ANSI	American National Standards Institute
API	Application Programming Interface
CBC	Cipher Block Chaining
CCM	Counter with CBC-Message Authentication Code
CAST	Cryptographic Algorithm Self-Test
PCT	Pair-Wise Consistency Test
CMAC	Cipher-based Message Authentication Code
CO	Crypto Officer
CDK	Cryptographic Development Kit
CSP	Critical Security Parameter
DES	Data Encryption Standard
DH	Diffie-Hellman
DHE	Diffie Hellman Key Exchange
DRGB	Deterministic Random Bit Generator
DSA	Digital Signature Algorithm
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
EES	Escrowed Encryption Standard (also known as Skipjack)
FSM	Finite State Machine
FIPS	Federal Information Processing Standard
GCM	Galois/Counter Mode
HMAC	Keyed Hash Message Authentication Code
ISC	Information Security Corporation
IV	Initialization Vector
KAT	Known Answer Test
KDF	Key Derivation Function
MAC	Message Authentication Code
NIST	National Institute of Standards and Technology
OS	Operating System
PC	Personal Computer
PCT	Pair-wise Consistency Test
PKV	Public Key Verification
RAM	Random Access Memory
RBG	Random Bit Generator
rDSA	RSA Digital Signature Algorithm
RSA	Rivest Shamir Adleman

SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
SIV	Synthetic Initialization Vector
SP	Special Publication
SSP	Sensitive Security Parameter
TOEPP	Tested Operational Environment's Physical Perimeter
XEX	Xor-encrypt-xor
XTS	XEX-based tweaked-codebook mode with ciphertext stealing