ISC Cryptographic Development Kit (CDK)

FIPS 140-2 Non-Proprietary Security Policy

Software Version: 8.0

Document Version: 4.0.15

Issue Date: June 1, 2018

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

Abstract: This document is a non-proprietary FIPS 140-2 Security Policy for ISC's Cryptographic Development Kit (CDK). It applies to CDK Version 8.0 and to all subsequent versions until otherwise indicated in new editions. It describes how the CDK meets the security requirements of FIPS 140-2 and how to run the CDK in a secure FIPS 140-2 mode. This policy was prepared as part of the FIPS 140-2 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

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)
[FIPS 81]	DES Modes of Operation
[FIPS 140-2]	Security Requirements for Cryptographic modules, May 25, 2001
[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
[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
[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 R2]	Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography
[SP 800-67 R1]	Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher
[SP 800-89]	Recommendation for Obtaining Assurances for Digital Signature Applications
[SP 800-90]	Recommendation for Random Number Generation Using Deterministic Random Bit Generators
[SP 800-107 R1]	Recommendation for Applications Using Approved Hash Algorithms
[SP 800-131A R1]	Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths

Table of Contents

0. Introduction

Information Security Corporation's Cryptographic Development Kit (CDK) Version 8.0 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 140-2 level 1 security requirements.

0.1 Document Organization

ISC's submission for FIPS 140-2 validation includes this security policy document and:

- Vender evidence (Entropy 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.

0.2 Platform Availability

The CDK software was designed for use on a variety of operating systems and hardware platforms. For FIPS 140-2 validation purposes, operational testing was performed on the following operating systems running on general purpose computers:

Operating System	Processors	Module Filename
Windows 10 64-bit (single-user mode)	Intel Core i7 with AES-NI	CDKC8008S.DLL
Windows 10 64-bit (single-user mode)	Intel Core i7 w/o AES-NI	CDKC8008S.DLL
Windows 10 64-bit (single-user mode)	AMD A8-3850 w/o AES-NI	CDKC8008S.DLL
CentOS 6.7 64-bit (single-user mode)	Intel Core i7 with AES-NI	libcdkc.so.8.0.0
CentOS 6.7 64-bit (single-user mode)	Intel Core i7 w/o AES-NI	libcdkc.so.8.0.0

Table 1 — Tested Platforms

The CDK software is provided as compiled code in the form of shared link libraries that can be run on Microsoft Windows and Linux operating systems.

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 the purpose of FIPS 140-2 validation, the CDK was loaded by multiple test applications (one for each algorithm family) and executed on each of the supported platform listed in the above table.

0.3 Security Level Summary

The following table lists the validation level met by the CDK for each area in FIPS 140-2. The CDK meets the requirements for an overall FIPS 140-2 level 1 validation.

Security Component	Security Level
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles, Services, and Authentication	1
Finite State Model	1
Physical Security	N/A
Operational Environment	1
Cryptographic Key Management	1
EMI/EMC	1
Self-Test	1
Design Assurance	1
Mitigation of Other Attacks	N/A
Overall Security Level	**1**

Table 2 – Security Component and Level

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

0.4 References

Federal Information Processing Standards Publication (FIPS PUB) 140-2, *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 140-2, and the CDK.

NIST Cryptographic Module Validation Program (CMVP)	http://csrc.nist.gov/groups/STM/cmvp/
FIPS 140-2 Security Requirements	http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf
ISC CDK	http://www.infoseccorp.com/products/cdks.htm
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	http://csrc.nist.gov/groups/STM/cavp/validation.html

1. Cryptographic Module Specification

1.1 Module Description and Overview

The CDK is a software module. The physical embodiment of the computer hardware on which it runs is a "multi-chip standalone module" in FIPS 140-2 terminology. The "physical cryptographic boundary" is defined to be the entire computer on which the CDK software runs. As a software module, the "logical boundary" contains the software modules that comprise the CDK shared link library.

The following diagram (Figure 1) illustrates the relationship between a typical software application (such as the supplied CDK test program) and the CDK, the computer's operating system, and system BIOS.

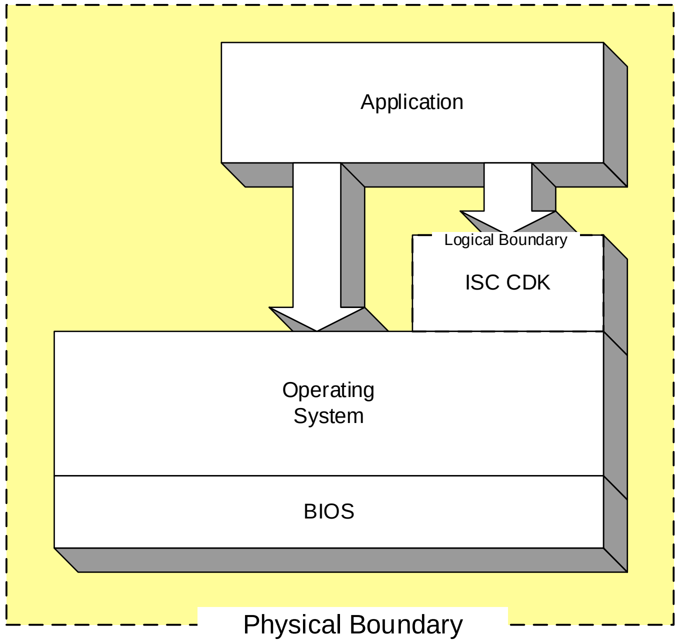

Figure 1 — Relationship between an Application and the Module

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.)

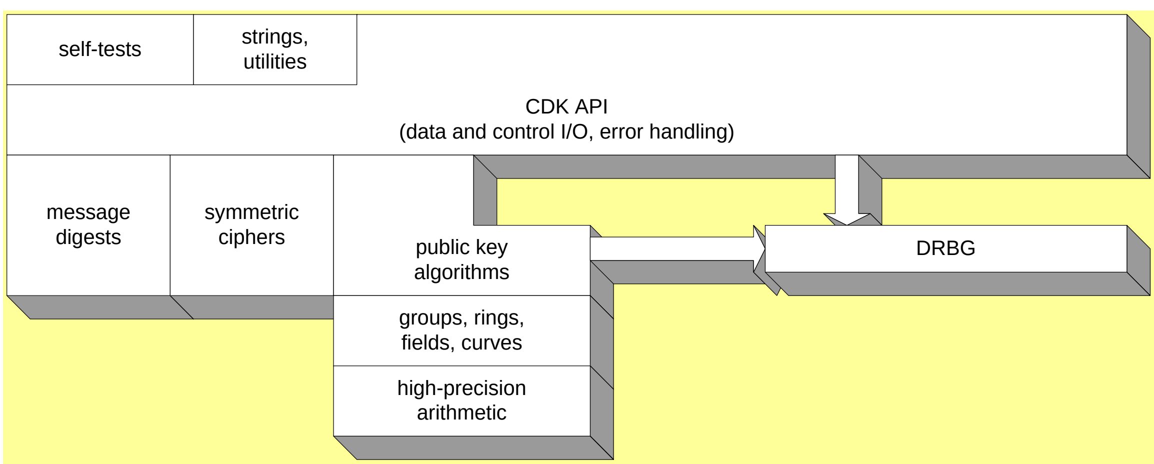

Figure 2 — Important Components of the CDK

1.2 Cryptographic Algorithms

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

1.2.1 Algorithms and Parameters Allowed in FIPS-mode

1.2.1.1 FIPS-Approved Algorithms

The FIPS-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 FIPS 140-2 Approved mode, only algorithms in the following three tables can be used.

CAVP Cert	Algorithm	Standard	Mode/Method	Key Lengths, Curves or Moduli	Use
4002	AES	FIPS 197, SP 800-38A, SP 800-38B, SP 800-38C, SP 800-38D, SP 800-38E	ECB, CBC, CFB8, CFB128, GCM[1] OFB, CTR, CCM, XTS[2], CMAC	128, 192, 256	Data Encryption/ Decryption
4002	AES	SP 800-38F	KW, KWP	128, 192, 256	Key Wrapping/ Unwrapping
854 (CVL)	ANSI x9.63 KDF	SP 800-135, ANSI x9.63 2001	SHA-224, SHA-256, SHA-384, SHA-512		Key Derivation
1090	DSA	FIPS 186-4		**PQG Gen:** (2048, 224) w/SHA(224, 256, 384, 512) (2048, 256) w/SHA(256, 384, 512) (3072, 256) w/SHA(256, 384, 512) **PQG Ver:** (1024, 160) w/SHA(1, 224, 256, 384, 512) (2048, 224) w/SHA(224, 256, 384, 512) (2048, 256) w/SHA(256, 384, 512) (3072, 256) w/SHA(256, 384, 512) **Key Pair Gen:** (2048, 224) (2048, 256) (3072, 256) **Sig Gen:** (2048, 224) w/SHA(1, 224, 256, 384, 512) (2048, 256) w/SHA(1, 224, 256, 384, 512)	Digital Signature Generation and Verification

---

[1] Internal IV generation only. The AES-GCM IV is generated internally randomly (scenario 2) or as a counter (scenario 1) per IG A.5.

[2] For storage applications only

				(3072, 256) w/SHA(1, 224, 256, 384, 512) **Sig Ver:** (1024, 160) w/SHA(1, 224, 256, 384, 512) (2048, 224) w/SHA(1, 224, 256, 384, 512) (2048, 256) w/SHA(1, 224, 256, 384, 512) (3072, 256) w/SHA(1, 224, 256, 384, 512)	
892	ECDSA	FIPS 186-4		**PKG:** P-224, P-256, P-384, P-521, K-233, K-283, K-409, K-571, B-233, B-283, B-409, B-571 **PKV:** ALL-P, ALL-K, ALL-B **SigVer:** Curves: P-192, P-224, P-256, P-384, P-521, K-163, K-233, K-283, K-409, K-571, B-163, B-233, B-283, B-409, B-571 Hash Algorithms: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	Digital Signature Verification
832 (CVL)	ECDSA	FIPS 186-4		**SigGen:** Curves: P-224, P-256, P-384, P-521, K-233, K-283, K-409, K-571, B-233, B-283, B-409, B-571 Hash Algorithms: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	Digital Signature Generation Primitive
853 (CVL)	KAS EC-DH	SP 800-56A	ECC	P-224, P-256, P-384, P-521, K-233, K-283, K-409, K-571, B-233, B-283, B-409, B-571	Key Derivation
2615	HMAC	FIPS 198-1	HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512, HMAC-SHA-512/224, HMAC-SHA-512/256, HMAC-SHA-3-224, HMAC-SHA-3-256, HMAC-SHA-3-384, HMAC-SHA-3-512	112, 112, 128, 192, 256, 112, 128	Message Authentication
1192	DRBG	SP 800-90A	HMAC-SHA-256		Deterministic Random Bit Generation
85	KAS	SP800-56A	FFC, ECC	**(FB:2048/224)** *dhEphem(init/resp)* FB: SHA-224, SHA-256, SHA-384, SHA-512 *dhOneFlow(init/resp)* FB: SHA-224, SHA-256, SHA-384, SHA-512 *dhStatic(init/resp)* FB: SHA-224, SHA-256, SHA-384, SHA-512 **(EB:P-224, EC: P-256, ED P-384, EE: P-521)** *ephemeralUnified(init/resp)* EB: SHA-224, SHA-256, SHA-384, SHA-512 HMAC EC: SHA-256, SHA-384, SHA-512 HMAC ED: SHA-384, SHA-512 HMAC EE: SHA-512 HMAC *onePassDH(init/resp)* EB: SHA-224, SHA-256, SHA-384, SHA-512 HMAC EC: SHA-256, SHA-384, SHA-512 HMAC	Key Agreement

				ED: SHA-384, SHA-512 HMAC EE: SHA-512 HMAC *staticUnified(init/resp)* EB: SHA-224, SHA-256, SHA-384, SHA-512 HMAC EC: SHA-256, SHA-384, SHA-512 HMAC ED: SHA-384, SHA-512 HMAC EE: SHA-512 HMAC	
2065	RSA	FIPS 186-2, FIPS 186-4		**FIPS 186-2** **SigVer9.31** Key Sizes: 1024, 1536, 2048, 3072 Hash Algorithms: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256 **SigVerPKCS1.5** Key Sizes: 1024, 1536, 2048, 3072 Hash Algorithms: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256 **FIPS 186-4** **KeyGenProbRandomRandomE** Key Sizes: 2048, 3072 **SigGen9.31** Key Sizes: 2048, 3072 Hash Algorithms: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256 **SigVer9.31** Key Sizes: 1024, 2048, 3072 Hash Algorithms: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256 **SigGenPKCS1.5** Key Sizes: 2048, 3072 Hash Algorithms: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256 **SigVerPKCS1.5** Key Sizes: 1024, 2048, 3072 Hash Algorithms: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256	Digital Signature Generation, Digital Signature Verification
831 (CVL)	RSA	FIPS 186-4		**RSADP:** (Mod2048)	RSADP Primitive
3307	SHS		SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256		Message Digest
4	SHA-3	FIPS 202	SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128, SHAKE256		Message Digest
2197	Triple-DES	SP 800-38A, SP 800-67	TECB, TCBC, TCFB8, TCFB64, TOFB, TCTR	192	Data Encryption/ Decryption

Table 3 – FIPS-Approved Cryptographic Algorithms

The CDK provides several FIPS approved methods for which there are no algorithm tests, but whose use is nevertheless allowed in FIPS-mode. The proper implementation and functionality of these mechanisms is "Vendor Affirmed." These are algorithms are listed in Table 4.

CAVP Cert	Algorithm	Standard	Mode/ Method	Key Lengths, Curves or Moduli	Use
Vendor Affirmation	AES	Addendum to SP 800-38A	CBC-CS3	128, 192, 256	Data Encryption/ Decryption

Table 4 — Vendor-Affirmed Cryptographic Algorithms

1.2.1.2 Non-Approved but Allowed in FIPS-mode Algorithms

The CDK implements several non-approved but allowed algorithms whose use is permitted in FIPS-mode. Table 5 lists these algorithms, the CDK classes in which they are implemented, their relevant key sizes and/or modes of operation, the referenced standards on which they are based.

Algorithm	Caveat	Use
Diffie-Hellman Supported sizes (2048, 224), (2048,256), (3072, 256)	Provides 112 or 128 bits of encryption strength	Key establishment
Elliptic Curve Diffie-Hellman Supported curves: P-224, K-233, B-233, P-256, K-283, B-283, P-384, K-409, B-409, P-521, K-571, B-571	Provides between 112 and 256 bits of encryption strength	Key establishment
NDRNG – entropy token external to the module's cryptographic boundary	Minimum bits requested per call: **Windows 10:** 1024-bits **CentOS 6.7:** 384-bits The module generates cryptographic keys whose strengths are modified by available entropy	Seeding for the DRBG
RSA Key Wrapping	Provides 112 or 128 bits of encryption strength	Key establishment

Table 5 — Non-FIPS Approved But Allowed Algorithms

1.2.1.3 FIPS-mode Keys and CSPs

Listed in Table 6 are the keys and CSPs used by the module in FIPS-mode.

Keys/CSPs	Generation	Storage	Input/Output	Description or Use	Accessible Role	Size (in bits)	Zeroization
AES key	DRBG	RAM	Input/output via API	AES encrypt/decrypt key	User, CO	128, 192, 256	Overwritten by 0's when freed
AES CMAC key	DRBG	RAM	Input/output via API	AES CMAC generate/verify key	User, CO	128, 192, 256	Overwritten by 0's when freed

AES GCM IV[3]	Internally using a counter or random value	RAM	Input/output via API	AES GCM initialization vector	User, CO	8-1024	Overwritten by 0's when freed
AES GCM key	DRBG	RAM	Input/output via API	AES GCM encrypt/decrypt/generate/verify key	User, CO	128, 192, 256	Overwritten by 0's when freed
AES key wrap key	DRBG	RAM	Input/output via API	AES encrypt/decrypt key	User, CO	128, 192, 256	Overwritten by 0's when freed
AES XTS key	DRBG	RAM	Input/output via API	AES XTS encrypt/decrypt key	User, CO	128, 192, 256	Overwritten by 0's when freed
DH private key	Internally using the DRBG	RAM	Input/output via API	DH private key agreement key	User, CO	2048-4096	Overwritten by 0's when freed
DH public key	Internally computed based on the private key	RAM	Input/output via API	DH public key agreement key	User, CO	2048-4096	Overwritten by 0's when freed
DSA private key	Internally using the DRBG	RAM	Input/output via API	DSA signature generation private key	User, CO	2048-4096	Overwritten by 0's when freed
DSA public key	Internally computed based on the private key	RAM	Input/output via API	DSA signature generation public key	User, CO	2048-4096	Overwritten by 0's when freed
ECC DH private key	Internally using the DRBG	RAM	Input/output via API	ECC DH private key agreement key	User, CO	224-571	Overwritten by 0's when freed
ECC DH public key	Internally computed based on the private key	RAM	Input/output via API	ECC DH public key agreement key	User, CO	224-571	Overwritten by 0's when freed
ECDSA private key	Internally using the DRBG	RAM	Input/output via API	ECDSA signature generation private key	User, CO	224-571	Overwritten by 0's when freed
ECDSA public key	Internally computed based on	RAM	Input/output via API	ECDSA signature generation public key	User, CO	224-571	Overwritten by 0's when freed

[3] The AES-GCM IV is generated internally randomly or as a counter per IG A.5. In the former case the IV is exactly 96-bits. In the latter case the IV may be 8- to 1024-bits in length.

	the private key						
HMAC key	DRBG	RAM	Input/output via API	Keyed hash key	User, CO	160-512	Overwritten by 0's when freed
HMAC integrity key	N/A	Disk	N/A	Keyed hash key to verify the integrity of the module at startup and on demand	None	256	Securely erasing the CDK library from disk
RSA key wrap private key	Internally using the DRBG	RAM	Input/output via API	Private component of an RSA key pair	User, CO	2048, 3072	Overwritten by 0's when freed
RSA key wrap public key	Internally computed based on the private key	RAM	Input/output via API	Public component of an RSA key pair	User, CO	2048, 3072	Overwritten by 0's when freed
RSA signature private key	Internally using the DRBG	RAM	Input/output via API	Private component of an RSA key pair	User, CO	2048, 3072	Overwritten by 0's when freed
RSA signature public key	Internally computed based on the private key	RAM	Input/output via API	Public component of an RSA key pair	User, CO	2048, 3072	Overwritten by 0's when freed
Triple-DES key	DRBG	RAM	Input/output via API	Triple-DES (3-Key) encrypt/decrypt key	User, CO	192	Overwritten by 0's when freed
				Triple-DES (2-Key) decrypt key for legacy use only	User, CO	112	
DRBG V and key	Internally using entropy input	RAM	N/A	DRBG internal state values	None	256	Overwritten by 0's when freed
DRBG seed key	Internally using entropy input	RAM	N/A	Entropy input (length is platform dependent but always greater than 256-bits)	None	>256	Overwritten by 0's when freed

Table 6 – Keys and CSPs, Key Sizes, and Security Strengths

1.2.1.4 AES-GCM Notes

1.2.1.4.1 IV Construction

In FIPS 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 A.5.

1.2.1.4.1.1 TLS – Section 8.1 SP800-38D, Scenario 1 IG A.5

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 and in support of the GCM ciphersuites listed in SP800-52. The AES GCM IV is generated internally using a deterministic counter as the nonce_explicit value and takes as input a 16-bit salt. The resulting IV is exactly 96-bits in length.

1.2.1.4.1.2 Random – Section 8.2 SP800-38D, Scenario 2 IG A.5

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 operating system APIs. The IV is exactly 96-bits in length.

1.2.1.4.2 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.

1.2.1.4.3 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`.

1.2.2 Non-FIPS-mode Algorithms

When run in non-FIPS-mode all of 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 FIPS mode**.

Algorithm	Non-Compliant Use
AES	Encryption/decryption using the CFB64 mode
AES GCM	Encryption using external IV generation by calling init() or initExt()
ANSI x9.63 KDF	Key derivation – using SHA-1 or SHA-3
DES	Encryption/decryption using the DES, DESX, or DES40 variants
Diffie-Hellman	Key establishment using keys whose size is less than (2048, 224) Key generation of keys with size less than (2048,224) (Non-compliant less than 112 bits of encryption strength)
DSA	Digital signature generation with keys whose size is less than (2048, 224) Key generation of keys with size less than (2048,224)
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 (Non-compliant less than 112 bits of encryption strength)
ECDSA	Digital signature generation 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
RSA	Digital signature generation using keys whose size is less than 2048-bits Key generation of keys with size less than 2048-bits 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 SHA-256, SHA-384, SHA-512, or SHA-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

Table 7 – Non-Approved Algorithms

1.3 CDK Modes

1.3.1 Running the CDK in FIPS-mode

When the CDK is run in FIPS-mode, only FIPS 140-2 approved algorithms are allowed to be used. It is the responsibility of the CO to properly configure the computer system, operating system, applications, and the CDK to operate in a secure FIPS-mode, if that is desired. (This may include configuring the system on which the application is installed as part of the installation process.) In configuring a system for use, the CO has access to the complete set of services declared in `cdk_fips.h`. The CO can allow or disallow User access to the CDK's built-in integrity tests, control the loading of applications by the User, and restrict User access to only FIPS 140-2 approved algorithms. A User has access to only those services provided by the CDK that are exposed for his use by the CO.

In order to operate in FIPS-mode, the CO must ensure that applications loaded by the operating system call the `enableFIPS()` method at startup and are limited in their use of the CDK to:

- only algorithms (and modes) noted above as FIPS-approved or those noted as exceptions
- only methods that are commented as being usable by a FIPS 140-2 compliant application
- only classes that are commented as being usable by a FIPS 140-2 compliant application
- only modes that are commented as being usable by a FIPS 140-2 compliant application
- only variants that are commented as being usable by a FIPS 140-2 compliant application

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

1.3.2 Running the CDK in non-FIPS Mode

If the `enableFIPS()` is not called, the CDK will operate in non-FIPS mode. The `isFIPS()`function will return false to indicate that the CDK is not operating in FIPS mode. The output of `StrVersion()`will not include the statement that the module is operating in FIPS mode.

2. Cryptographic Module Ports and Interfaces

As a FIPS 140-2 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 physical boundary of the CDK.

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()` method which may be called at any time to determine if the CDK is in the hard error state. 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.

Interface	Logical Interface	Physical Port
Data Input	Data passed to the API calls to be used by the Module	Standard Input Port (e.g. Keyboard)
Data Output	Data returned from API calls, generated by the Module	Standard Output Port (e.g. Monitor)
Control Input	API calls	N/A
Status Output	C++ exceptions, the Algorithm::isErrorState() function, and the ISC_CDK::isFIPS() functions	Standard Output Port (e.g. Monitor)
Power	N/A	Supplied by PC

Table 8 – Module Interface Mapping

3. Roles, Services, and Authentication

The CDK module supports two roles: "Crypto-Officer" (*CO*) and "User." The CO is the human being who initializes the module using services provided by the CDK. The User is the resulting application.

The CDK provides no maintenance access interface and therefore does not support a *Maintenance* role. FIPS 140-2 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 and User roles. No other roles are supported.

The CO configures the computer system, operating system, and the CDK to operate in a secure FIPS 140-2 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 140-2 requirements are provided in a separate document: Crypto Officer's Guide. The User has access to only those services provided by the CDK that are exposed by the CO.

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

Bypass services are not provided.

Tables 9 and 10 provide details on the services available to each role, and each role's access rights with respect to those services. If the CO does not place any restrictions on users during installation, users have the right to perform any of the following basic encryption or decryption methods:

- AES::crypt – perform AES encryption or decryption.
- DES::crypt – perform Triple-DES encryption or decryption.
- EES::crypt – perform Skipjack encryption or decryption.
- Key::Encrypt – perform RSA, Diffie-Hellman, or Elliptic Curve Diffie-Hellman, key wrapping.
- Key::Decrypt – perform RSA, Diffie-Hellman, or Elliptic Curve Diffie-Hellman, key unwrapping.

Security Service	Description	Input/ Output	Role	Cryptographic Keys and Critical Security Parameters	Type of Access
Configure	initialize and configure module	configuration parameters/ status	CO	All algorithms and modes	Read, Write
Integrity self-test	check module integrity	none/ status	User, CO	Integrity test HMAC-SHA-256 key	Read
Perform self-tests	check module algorithm correctness	none/ status	User, CO	Known Answers for SHA-1, SHA-2, SHA-3, HMAC, Triple-DES, AES, DRBG Pairwise Consistency for DSA, ECDSA, RSA	Read
Show status	output hard error state	none/ status	User, CO	None	Read
Zeroize	erase key or critical security parameter	key/ status	User, CO	Any key or security parameter listed further down in this table	Write
Symmetric key generation using DRBG	generate a random key	key size/ key, status	User, CO	Symmetric Key (AES, Triple-DES, CMAC or HMAC)	Create, Read
Random number generation	generate a random number	size/ value, status	User, CO	DRBG Key	Create, Read
Asymmetric key generation	generate an asymmetric public and private key	key type, size/ key, status	User, CO	DH, DSA, ECDH, ECDSA, RSA key	Create, Read
Symmetric encrypt/decrypt	encrypt/decrypt data using a symmetric algorithm	key, data/ ciphertext, status	User, CO	Symmetric Key	Use, Read
Symmetric digest	digest data	key, data/ digest value, status	User, CO	AES CMAC Key	Use, Read
Message digest	digest data	data/ digest value, status	User, CO	None	None
Keyed hash	digest data	key, data/ digest value, status	User, CO	HMAC Key	Use, Read
Key agreement	derive a shared key	keys/ shared secret, status	User, CO	DH, ECDH private key; DH, EC DH public key (provided as input by the caller); ANSI x9.63 KDF	Create, Use, Read
Key transport	encrypt a data encryption key with a key encryption key	keys/ ciphertext, status	User, CO	DH, ECDH, RSA public key, symmetric key (provided as input by the caller); ANSI x9.63 KDF	Use, Read
Digital signature	create a digital signature	key, digest value, digest type/ signature, status	User, CO	DSA, ECDSA, RSA private key	Use, Read

Table 9 – Services Available to Cryptographic Officer and User Roles in FIPS mode

Security Service	Description	Input/ Output	Role	Cryptographic Keys and Critical Security Parameters	Type of Access
Configure	initialize and configure module	configuration parameters/ status	CO	All algorithms and modes	Read, Write
Integrity self-test	check module integrity	none/ status	User, CO	Integrity test HMAC-SHA-256 key	Read
Perform self-tests	check module algorithm correctness	none/ status	User, CO	KAT for SHA-1, SHA-2, SHA-3, HMAC, Triple-DES, AES, Skipjack, DRBG, RSA PCT for DSA, ECDSA	Read
Show status	output hard error state	none/ status	User, CO	None	Read
Zeroize	erase key or critical security parameter	key/ status	User, CO	Any key or security parameter listed further down in this table	Write
Symmetric key generation using DRBG	generate a random key	key size/ key, status	User, CO	Symmetric Key (AES, DES, Skipjack, Triple-DES, CMAC or HMAC)	Create, Read
Random number generation	generate a random number	size/ value, status	User, CO	Random numbers	Create, Read
Asymmetric key generation	generate an asymmetric public and private key	key type, size/ key, status	User, CO	DH, DSA, ECDH, ECDSA, RSA key	Create, Read
Symmetric encrypt/decrypt	encrypt/decrypt data using a symmetric algorithm	key, data/ ciphertext, status	User, CO	Symmetric Key	Use, Read
Symmetric digest	digest data	key, data/ digest value, status	User, CO	AES CMAC Key	Use, Read
Message digest	digest data	data/ digest value, status	User, CO	None	None
Keyed hash	digest data	key, data/ digest value, status	User, CO	HMAC Key	Use, Read
Key agreement	derive a shared key	keys/ shared secret, status	User, CO	DH, ECDH private key; DH, EC DH public key (provided as input by the caller); ANSI x9.63 KDF	Create, Use, Read
Key transport	encrypt a data encryption key with a key encryption key	keys/ ciphertext, status	User, CO	DH, ECDH, RSA public key, symmetric key (provided as input by the caller); ANSI x9.63 KDF	Use, Read
Digital signature	create a digital signature	key, digest value, digest type/ signature, status	User, CO	DSA, ECDSA, RSA private key	Use, Read

Table 10 – Services Available to Cryptographic Officer and User Roles in non-FIPS mode

4. 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.

5. Physical Security

The module is a software-only module and the physical security requirements of FIPS 140-2 level 1 do not apply.

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 as described in section 1 of this document.

Within the tested environments user processes are segregated in to 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 user role, and no other process can access the same instance of the module, the module operates in single user mode.

7. Cryptographic Key Management

The CDK uses, creates, and/or manages:

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

7.1 Key Generation

The CDK generates keys for the FIPS-approved and vendor affirmed algorithms listed in Table 3 and Table 4. The CDK also generates non-FIPS-Approved keys for algorithms listed in Table 5. The CDK can generate symmetric keys (for a symmetric cipher or keyed hash function) using its DRBG. In order to generate key pairs, the public key generation methods use the CDK's random number generator.

7.2 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 Exchange, ECC Diffie-Hellman Key Agreement, Diffie-Hellman Key Exchange, and AES key wrapping.

7.3 Key Entry and Output

The CDK does not manage any manually distributed cryptographic keys, either entry or output, external to the physical 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 key distribution by the application.

7.4 Key Storage/Archiving

The CDK is a low-level cryptographic toolkit and does not provide any key storage. As detailed in 9.1 Power-Up 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.

7.5 Key Destruction

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 calling the clear method in all existing CDK objects before application termination). In order 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.

8. EMI/EMC

The CDK should only be run on commercial computer systems that, at a minimum, conform to the EMI/EMC requirements specified by 47 CFR FCC Part 15, Subpart B, Class A.

9. Self-Tests

The CDK performs self-tests in order 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 a hard error state and no further cryptographic operations are possible. To recovery from this error reimage the module.

The CDK returns error codes from its API to enable the calling application to detect an issue and allow the user to resolve it. There are two special error codes that the CDK returns that 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. The CDK returns `CDK_KEYPAIR_INCONSISTENT` with value `1234` when a pairwise key test fails during an on-demand self-test and the CDK transitions into the hard error state.

9.1 Power-Up Tests

When the CDK module is loaded from disk by the operating system, it executes a *software/firmware integrity test* as well as a *critical functions test*. 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. The critical function test includes known answer tests (KATs) or pair-wise consistency tests (PCTs) for each of the FIPS-approved algorithms in the CDK (see Section 1.2.1 and 1.2.1.2 above). The integrity test operates by calculating a 256-bit HMAC (HMAC-SHA-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 or not the module is put into FIPS-mode. If the software integrity test fails or if any self-test fails, the module displays a message on the output interface, enters the hard error state, and inhibits all cryptographic services.

Algorithm	Type	Description
Software Integrity	KAT	HMAC-SHA-256
DRBG	KAT	SP 800-90A compliant health test (with and without PR)
HMAC	KAT	One KAT each: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512
AES	KAT	Encrypt and decrypt AES-128 ECB, AES-128 CBC
AES CCM	KAT	Generate and verify AES-128
AES GCM	KAT	Encrypt and decrypt AES-128
AES XTS	KAT	Encrypt and decrypt AES-256
AES CMAC	KAT	Generate AES-128
Triple-DES	KAT	2-key Triple-DES decrypt-only and 3-key Triple-DES encrypt/decrypt
Skipjack	KAT	Decrypt
RSA	KAT	Sign and verify using 2048-bit key, SHA-256, PKCS#1
DSA	PCT	Sign and verify using 2048-bit key, SHA-256
ECDSA	PCT	Sign and verify using P-256, SHA-256 Sign and verify using B-233, SHA-256
ECC CDH	KAT	Primitive “Z” computation using two P-256 keys
SHA-1	KAT	FIPS 180-4 KATs
SHA-2	KAT	One KAT each: SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256
SHA-3	KAT	One KAT each: SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128, SHAKE256

Table 11 – Power-Up Self-Tests

9.2 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 hard error state, and inhibits all cryptographic services.

9.2.1 Random Number Tests

9.2.1.1 NIST SP 800-90A Generate Periodic Test

SP 800-90A, Section 11.3 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.

9.2.1.2 FIPS 140-2 Reseed Test/NDRNG Test

During a reseed operation the random number generator compares the new seed key, from the NDRNG, with the current key and if the values are identical the DRBG enters the uninitialized state and returns an error.

9.2.1.3 FIPS 140-2 Continuous Random Number Generation Test

The CDK performs a continuous test of the DRBG implementation. This health test is compliant with NIST SP 800-90A Section 11.3. If two successive DRBG output blocks are ever equal, the CDK aborts the current operation and enters its hard error state.

9.2.2 Pair-Wise Self-Tests

All DSA and 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 Diffie-Hellman, or 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.

9.2.3 SP 800-56A Assurances

As required per IG 9.6, the CDK conditionally performs the necessary checks when generating, importing, or using domain parameters and keys according to SP 800-56A-rev2 sections 5.5.2, 5.6.2, and/or 5.6.3.

9.2.4 IG A.9 XTS-AES Test

As required per IG A.9, when the XTS-AES object is initialized by a user 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.

9.2.5 On-Demand Self-Tests

As documented in the Crypto Officer's Guide and the User's Guide, the CO or User may, on-demand, invoke any of the self-tests listed in Table 11 to ensure the integrity of specific algorithms. There is also a master test function that a User can call to run all self and integrity tests.

10. Mitigation of Other Attacks

The CDK has not been designed to mitigate any specific attacks.

11. 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
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
FFC	Finite-Field Cryptography
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
SP	Special Publication
XEX	Xor-encrypt-xor
XTS	XEX-based tweaked-codebook mode with ciphertext stealing