ISC Cryptographic Development Kit (CDK) Version 7.0

FIPS 140-1 Security Policy

Document Version: 3.10

Issue Date: August 25, 2003

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

Abstract: This document is a non-proprietary Cryptographic Module Security Policy for Information Security Corporation's CDK Version 7.0. It describes how the ISC CDK meets the security requirements of FIPS 140-1, and how to use the ISC CDK in a secure FIPS 140-1 mode. This policy was prepared as part of the FIPS 140-1 Level 1 validation of the ISC CDK.

Table of Contents

0 Introduction

Information Security Corporation's Cryptographic Development Kit (ISC CDK) Version 7.0 is a software module. The software module is a dynamic link library (DLL) 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 ISC 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 140-1 level 1 security requirements.

0.1 Document Organization

ISC's submission for FIPS 140-1 validation includes this security policy document and:

- Vender evidence (Self-test document, Key Management document, Evaluator's guide, and CDK Methods document),
- Finite state machines model diagram and explanation,
- Proprietary source code and build configuration files.

0.2 Platform Availability

The ISC CDK software was designed for use on a variety of operating systems and hardware platforms. For FIPS 140-1 validation purposes, the software was tested on a Dell Optiplex GX1 Personal Computer running the Microsoft Windows 2000 operating system configured as single user system. The ISC CDK software is provided in the form of compiled code as dynamic link libraries (DLL), which can be run on Microsoft Windows 95, 98, ME, NT, 2000, and XP operating systems.

The DLL's application programming interface (API), which provides access to the supported cryptographic primitives, consists of a set of C++ classes as documented in 'cdk.h', the other header files referenced therein, and related documentation. For FIPS 140-1 testing purposes, a testing application was loaded in order to exercise the DLL.

0.3 References

Federal Information Processing Standards Publication (FIPS PUB) 140-1, *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 140-1, and the ISC CDK.

NIST Cryptographic Module Validation Program (CMV)	http://csrc.nist.gov/cryptval
FIPS 140-1 Security Requirements	http://csrc.nist.gov/cryptval/140-1.htm
The 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://sbc.nist.gov/cryptval/vallists.htm

1 Cryptographic Module Architecture

1.1 Module Description and Overview

The ISC CDK is a software module. The physical embodiment of the computer hardware on which it runs is a "multi-chip standalone module" in FIPS 140-1 terminology. The "physical cryptographic boundary" is defined to be the entire computer on which the ISC CDK software runs. As a software module, the "logical boundary" contains the software modules that comprise the ISC CDK dynamic link library.

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

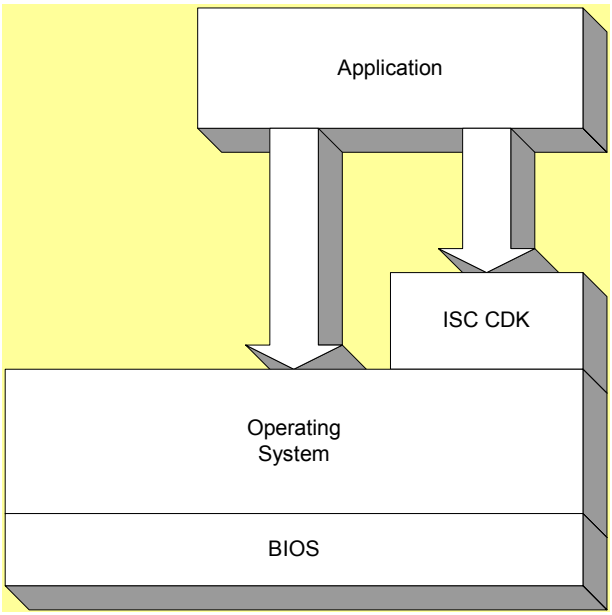

Figure 1: Relationship between an Application and the Cryptographic Module

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

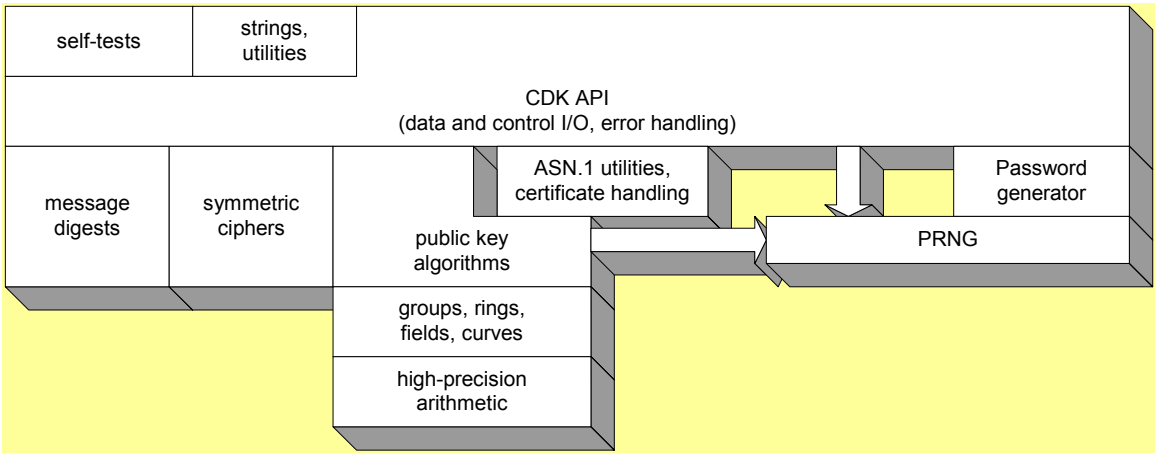

Figure 2: Important Components of the ISC CDK

2 Module Interfaces

As a FIPS 140-1 multi-chip standalone module, the ISC 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 ISC CDK.

The ISC 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 ISC 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 anytime 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)
Date 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 and the Algorithm::isErrorState() function	Standard Output Port (e.g. Monitor)
Power	N/A	Supplied by PC

Table 1: Module Interface Mapping

3 Roles and Services

The ISC CDK module supports two roles: "Crypto-Officer" (*CO*) and "User." The ISC CDK provides no maintenance access interface and therefore does not support a *Maintenance* role. FIPS 140-1 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 ISC CDK for the CO and User roles. No other roles are supported.

The CO configures the computer system, operating system, and the ISC CDK to operate in a secure FIPS 140-1 mode, if that is desired (this may include configuring the system on which the application is installed as part of the installation process). Conditions for meeting FIPS 140-1 requirements are provided in Section 12 of this document. The User has access to only those services provided by the ISC CDK that are exposed for his use by the CO.

A "show status" service is provided by the static Algorithm::isErrorState() function. The API will return TRUE if the CDK is in the hard error state and FALSE otherwise.

Self-test services are described in Section 11 of this document.

Bypass services are not provided.

Tables 2 and 3 provide details on the services available to each role, and each role's access rights within those services. If the CO does not place any restrictions on the user during installation, the user has the right to perform any of the following basic encryption or decryption methods:

- AES::crypt – perform AES encryption or decryption.
- DES::crypt – perform DES[1], DESX, or TDES encryption or decryption.
- EES::crypt – perform EES encryption or decryption.
- Key::Encrypt – perform RSA, ElGamal, or Elliptic Curve ElGamal, key wrapping.
- Key::Decrypt – perform RSA, ElGamal, or Elliptic Curve ElGamal, key unwrapping.

[1] DES is for legacy system interoperability use only.

Role	Security Service	Cryptographic Keys and Critical Security Parameters	Type of Access
Crypto Officer	Test ISC CDK software integrity	TDES MAC key	Read, write to disk
Crypto Officer	Configure ISC CDK to perform selective cryptographic functions	Start-up type for each service	Read, write to disk
Crypto Officer or User	Perform self-tests	Known Answers for CRC, MD2, MD5, SHA-1, SHA256, SHA384, SHA512, HMAC-SHA-1, DES, TDES, AES, RC2, RC4, EES, DSA, ECDSA, RSASign, PRNG, and PWD.	Read from memory
Crypto Officer or User	Show status	Error indicators	Read from memory
Crypto Officer or User	Generate session key using PRNG for use in DES, TDES, AES, EES encryption	Session Key	Read, write to memory
Crypto Officer or User	Generate Random numbers	Random numbers	Read, write to memory
Crypto Officer or User	Shown in Table 3	Shown in Table 3	Read, write to memory

Table 2: Services Available to Cryptographic Officer and User Roles

Table 3 shows the cryptographic methods available to the Cryptographic Officer and User Roles.

Method	Critical Security Parameters	Type of Access
AES::init	symmetric key, IV, mode	write memory
AES::crypt	IV and CTR updated	read/write memory
AES::clear	symmetric key, IV, CTR	write memory
AES::~AES	symmetric key, IV, CTR	write memory
AES::setcounter	CTR	write memory
DES::init	symmetric key, IV, mode, variant[2]	write memory
DES::crypt	IV and CTR updated	read/write memory
DES::clear	symmetric key, IV, CTR	write memory
DES::~ DES	symmetric key, IV, CTR	write memory
DES::setcounter	CTR	write memory
DES::reset	symmetric key	read/write memory
EES::init	symmetric key, IV, mode	write memory
EES::crypt	IV and CTR updated	read/write memory
EES::clear	symmetric key, IV, CTR	write memory
EES::~ EES	symmetric key, IV, CTR	write memory
EES::setcounter	CTR	write memory
HMAC	symmetric key	read/write memory
RSA::clear	public key, private key	read/write memory
RSA::RSA	public key, private key	read/write memory
RSA::~RSA	public key, private key	read/write memory
RSA::setup	public key, private key	read/write memory
RSA::genpub	public key	read/write memory
RSA::loadpub	public key	read/write memory
RSA::loadpriv	private key	read/write memory
RSA::encrypt	public key	read memory
RSA::decrypt	private key	read memory
RSA::Sign	private key	read memory
RSA::asn1private	private key	write memory
RSA::loadasn1private	private key	read/write memory
RSA::loadpub	public key	read/write memory
signature::tostr	signature	read memory
signature::toasn1	signature	read memory
signature::toraw	signature	read memory
signature::load	signature	read/write memory
signature::clear	signature	write memory
signature::init	private key, public key	read memory
signature::~signature	signature	write memory

[2] Variant is DES, DESX, or TDES. Note: CTR mode for DES or TDES **cannot** be used in FIPS mode.

Method	Critical Security Parameters	Type of Access
key::clear	private key, public key	write memory
key::loadprivate	private key	read/write memory
key::DLLoadpublic	public key	read/write memory
key::RSALoadpublic	public key	read/write memory
key::DLGetRawpublic	public key	read memory
key::RSAGetRawpublic	public key	read memory
key::GetRawprivate	private key	read memory
key::genpub	private key, public key	read/write memory
key::RSAkeygen	private key, public key, random	read/write memory
key::DLkeygen	private key, public key, random	read/write memory
key::Sign	signature, private key	read/write memory
key::SignCheck	signature, public key	read memory
key::Encrypt	public key	read memory
key::Decrypt	private key	read memory
key::~key	private key, public key, parameters	write memory
key::A	private key	read memory
key::B	private key	read memory
key::asn1sign	private key, random	read memory
key::loadseed	parameters, private key	read/write memory
key::loadprv	private key	read/write memory
key::loadpub	public key	read/write memory
key::asn1parameters	parameters	read memory
key::asn1private	private key	read memory
key::asn1public	public key	read memory
key::getprivate	private key	read memory
key::setprivate	private key	read/write memory
DSA_Generateparameters	parameters, seed	read/write memory
PRNG::addseed	seed	read/write memory
PRNG::add	seed	read/write memory
PRNG::gens	seed, random	read/write memory
PRNG::gen	seed, random	read/write memory
PRNG::PRNG	seed	read/write memory
PRNG::~PRNG	seed	write memory
PRNG::churn	seed	write memory
PRNG::genbasic	seed, random	read/write memory
PRNG::setseed	seed	write memory

Method	Critical Security Parameters	Type of Access
SHA::reset	hash value	write memory
SHA::resetk	hash value	write memory
SHA::clear	hash value	write memory
SHA::SHA	hash value	write memory
SHA::~SHA	hash value	write memory
SHA::add	hash value	write memory
SHA::final	hash value	write memory
SHA::finalstore	hash value	write memory
SHA::result	hash value	read memory
SHA::tostr	hash value	read memory
SHA2::clear	hash value	write memory
SHA2::SHA2	hash value	write memory
SHA2::~SHA2	hash value	write memory
SHA2::add	hash value	write memory
SHA2::final	hash value	write memory
SHA2::result	hash value	read memory
SHA2::tostr	hash value	read memory

Table 3: Cryptographic Methods Available to Cryptographic Officer and User Roles

4 Finite State Model

The ISC CDK was designed around a Finite State Model (FSM) that is detailed in a proprietary document submitted with this security policy. The ISC CDK has a Crypto Officer role and a User role assumed by the user of the application installed by the Crypto Officer. The ISC CDK has the following states:

Crypto Officer States:

ISC CDK Module installation and configuration (initialization). In addition, the Crypto Officer may perform any user functions.

User States:

Power off, Power on, System load, CDK power on self-test, Operational state (key-less cryptographic functions), Operational state (keyed cryptographic functions), Soft error, Hard error, Key generation, On-demand self-test.

5 Physical Security

The platform used for testing the software module is a Dell Optiplex GX1 Personal Computer, which is a standard PC that includes the use of standard production quality boards, ICs, power supplies, and passivation on chips and boards. The computer is a “multi-chip standalone module” in FIPS 140-1 terminology.

The ISC CDK meets FIPS 140-1 Level 1 physical security requirements.

6 Software Security

The ISC CDK was implemented following the practices listed in FIPS 140-1 Appendix B. The ISC CDK meets FIPS 140-1 Level 3 software security requirements.

- The ISC CDK is provided only as compiled "executable" code in order to discourage scrutiny and modification by users.
- The source code for the ISC CDK is in C++, with comments.
- The ISC CDK uses a keyed TDES MAC on the software module that is tested as part of its power-up self-test in order to ensure that it has not been tampered with.

7 Operating System Security

Operating system security is assured as follows:

- The CO must configure Windows 2000 as a single user operating system environment (*i.e.*, only one user can be logged on at a time). To ensure that the system runs in single user mode, the CO must disable the **Server** and **RunAsService** services by using the **Computer Management** tool as an administrator: select **Programs->Administrative Tools>Computer Management** from the **Start** menu**;** select **Services** under **Services and Applications** and locate the **Server** service in the list; highlight this service, choose **Properties** from the **Action** menu, and change the **Startup type** to **Disabled**. Repeat the last step for the **RunAsService** service. If desired, the critical **Server** and **RunAsService** services can also be disabled in an automated fashion by directly modifying the Windows registry.

- The Windows 2000 pagefile (swapfile) must be located on a local disk, not on a network drive.

- When Windows 2000 loads the CDK into memory, the CDK runs a TDES MAC over the disk image of the file CDK.DLL, skipping an embedded TDES MAC field installed by the CO. This computed MAC is then compared to the embedded value. Only if these two values are equal will the CDK let Windows load it. Each process in Windows 2000 has its own, unique instance of the CDK that is wholly dedicated to that process. The CDK is not shared between processes.

8 Cryptographic Key Management

The ISC 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 agreement protocols based on public key schemes,

The ISC CDK provides cryptographic algorithm support that can be used to meet the FIPS 140-1 Level 1 requirements for security.

8.1 Key Generation

The ISC CDK generates FIPS-Approved keys listed in Table 5. The ISC CDK also generates non-FIPS-Approved keys that are listed in Table 6. The ISC CDK also provides a means for generating random session keys (for a symmetric cipher or key hash function) using its PRNG. When a key is generated, it is stored internally until the user deletes the containing object, the application exits, or the clear() method is called. When such events occur, the memory containing the key is erased. An instantiated 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. In order to generate random key pairs the public key generation methods require a random seed, supplied either from the ISC CDK's random number generator or from other source.

8.2 Key Distribution

The module doesn't perform key distribution. The module has the basic cryptographic functions which could be used by applications to build key distribution capabilities. The key distribution techniques available for use include RSA Key Exchange, Diffie-Hellman Key Agreement, and ElGamal Key Exchange.

8.3 Key Entry and Output

The ISC 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 ISC CDK does provide methods for entering and retrieving symmetric keys and public/private key pairs in electronic form for manual key distribution by the application. While the ISC CDK does provide mechanisms for encrypting/decrypting public/private keys in both PKCS#8 and PKCS#12 format these formats are not approved for use in a 140-1 compliant application so there is no approved method for entering or outputting encrypted public or private keys. Table 4, below, lists the preferred methods for key entry and output. In general, the most preferred method for entering or getting public/private keys from the CDK is in ASN.1 encoded form using loadpub, loadprv, asn1public, and asn1private.

Method	Class Name	Service Provided
AES::init	AES	Enter a plaintext AES key into the module.
DES::init	DES	Enter a plaintext DES key into the module.
EES::init	EES	Enter a plaintext EES/Skipjack key into the module.
PRNG::gens	PRNG	Generate a plaintext random symmetric key and output it from the module.
Key::Encrypt	Key	Encrypt a plaintext symmetric key and output it from the module wrapped with a public key.
Key::Decrypt	Key	Decrypt an encrypted symmetric key and output it from the module in plaintext (suitable for entry using the init methods above).
Key::loadprivate	Key	Enter a raw encoded plaintext DSA or ECDSA private key into the module.
Key::DLLoadPublic	Key	Enter a raw encoded plaintext DSA public key into the module.
Key::GetRawPrivate	Key	Output a generated or loaded plaintext DSA or ECDSA private key in raw format from the module.
Key::DLGetRawPublic	Key	Output a generated or loaded plaintext raw DSA or ECDSA public key from the module.
Key::rsai.loadpriv	Key/RSA	Enter a raw plaintext RSA private key into the module.
Key::RSALoadPublic	Key	Enter a raw plaintext RSA public key into the module.
Key::RSAGetRawPublic	Key	Output a generated or loaded raw plaintext RSA public key.
Key::loadpub	Key	Enter a plaintext ASN.1 encoded RSA, DSA, or ECDSA public key into the module.
Key::loadprv	Key	Enter a plaintext ASN.1 encoded RSA, DSA, or ECDSA private key into the module.
Key::asn1public	Key	Output a plaintext ASN.1 encoded RSA, DSA, or ECDSA public key from the module.
Key::asn1private	Key	Output a plaintext ASN.1 encoded RSA, DSA, or ECDSA private key from the module.

Table 4: Key entry and output methods

8.4 Key Storage/Archiving

The module doesn't store any of the generated keys. A single, special purpose TDES key is hard coded in the module in plaintext form and is used to compute the TDES MAC to verify the integrity of the module. No other keys are stored or archived.

8.5 Key Destruction

Under normal operations all internal memory allocated by the ISC 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 TDES key embedded in the ISC CDK in plaintext form, the hard disk must be reformatted.

9 Cryptographic Algorithms

The ISC CDK supports a wide variety of cryptographic algorithms. Whenever possible, all NIST FIPS-approved algorithms designed for a particular cryptographic function are provided, as are non FIPS-approved, but commercially popular algorithms.

The NIST FIPS-approved cryptographic algorithms implemented in the ISC CDK are listed in Table 5. Also indicated in this table are the corresponding classes and the name of the (principal) source file in which they are implemented, as well as the corresponding NIST standards (or alternate standards referenced by NIST). To use the CDK in FIPS 140-1 mode, one must use these algorithms and no others. Table 6 lists those algorithms offered by the CDK that are not FIPS-approved.

The ISC CDK provides cryptographic and mathematical primitives for performing both Diffie-Hellman (DH) and Elliptic Curve Diffie-Hellman key agreement but does not provide methods for communicating the public keys required to perform the operation. It is left to the application to exchange the public keys used to perform DH and ECDH. The implementation of DH and ECDH is located in key.c and cert.c and instructions for performing either are found in pk.h. The implementations are based on RFC 2631, ANSI X9.42-1998, IEEE 1363-2000, and ANSI X9.63.

Algorithms	Class Name	Definition File	Implementation File(s)	Modes	Standards
AES ("Rijndael")	AES	aes.h	aes.c	ECB CBC CFB OFB CTR	FIPS 197 NIST SP 800-38A
DES (for legacy system interoperability use only)	DES	des.h	des.c	ECB CBC CFB OFB	FIPS 46-2/46-3 FIPS 74 FIPS 81 FIPS 113 ANSI X9.17 1985
TDES	DES	des.h	des.c	ECB CBC CFB OFB CTR[3]	FIPS 46-2/46-3 FIPS 74 FIPS 81 FIPS 113 ANSI X9.52 1998 ANSI X9.17 1985
EES ("Skipjack")	EES	ees.h	ees.c	ECB CBC CFB OFB CTR	FIPS 185

[3] CTR mode for TDES **cannot** be used in FIPS mode. It has not undergone NIST validation.

Algorithms	Class Name	Definition File	Implementation File(s)	Modes	Standards
SHA-1	SHA	sha.h	sha.c shat.a		FIPS 180-1
DSA	Key	pk.h	key.c cert.c		FIPS 186-2
ECDSA	Key	pk.h	key.c cert.c		FIPS 186-2
HMAC-SHA-1	HMAC	hmac.h	none		FIPS 198;
RSA Signature generation and verification	RSA	pk.h	key.c cert.c		FIPS 186-2; RFC2313 (PKCS#1v1.5); block padding as in ANSI X9.31-1998

Table 5: FIPS-Approved Cryptographic Algorithms (Services)

The ISC CDK uses FIPS approved methods to generate passwords and random numbers. These are listed in Table 5A.

Algorithms	Class Name	Definition File	Implementation File(s)	Standards
Password Generator	Password	pass.h	pass.c	FIPS 181
Pseudo-Random Number Generator	PRNG	rand.h	rand.c	Combines a platform-specific seeding mechanism with the pseudo-random number generator described in FIPS 186-2 Appendix 3.1 and the Aug 2001 NIST recommendations. Entropy is derived from a variety of system calls.

Table 5A: FIPS-Approved Algorithms for password and random number generation.

Listed in Table 6 are those algorithms implemented by the ISC CDK which are non FIPS-approved.

Algorithms	Class Name	Definition File	Implementation File(s)	Modes	Standards or References
DES	DES	des.h	des.c	CTR	FIPS 46-2/46-3 FIPS 74 FIPS 81 FIPS 113 ANSI X9.17 1985
SHA-256, SHA-384, SHA-512	SHA2	sha.h	sha2.c		FIPS 180-2
HMAC-MD5	HMAC	hmac.h	none		RFC2104 ANSI X9.71 ISO/IECc FDIS 9797-2
RSA Encryption, Decryption	RSA	pk.h	key.c cert.c		RFC2313 (PKCS#1v1.5)
ElGamal	Key	pk.h	key.c		Menezes, van Oorschot and Vanstone, "Handbook of Applied Cryptography", CRC Press (1997), 294-296.
EC ElGamal	Key	pk.h	key.c		Menezes, van Oorschot and Vanstone, "Handbook of Applied Cryptography", CRC Press (1997), 297-298.
RC2	RC2	rc2.h	rc2.c		RFC2268
RC4	RC4	rc4.h	rc4.c		RFC2246 (SSL/TLS)
MD2	MD2	md2.h	md2.c		RFC1319
MD5	MD5	md5.h	md5.c		RFC1321

Table 6: Non FIPS-Approved Algorithms (Services)

10 EMI/EMC

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

11 Self-tests

The ISC CDK performs self-tests in order to ensure that it is functioning properly.

11.1 Power-Up Tests

When the ISC CDK module is loaded from disk by the operating system, it executes a *software/firmware integrity test* as well as a *critical functions test*. The critical function test includes known answer tests (KATs) for each of the FIPS-approved algorithms in the CDK (See Table 7). The integrity test operates by calculating a 64-bit TDES MAC over the module and comparing it to an expected value embedded (along with the TDES key) in the module itself at the factory. If the computed MAC does not match the embedded expected value, or if one of the algorithm KATs fails, then the module enters an error state: it refuses to load (returning a failure code to the operating system) and no user operations are possible.

Power-Up Self-Test[4]	Self_Test
Hash functions	CRC
	MD2
	MD5
	SHA1
	SHA256
	SHA384
	SHA512
	HMAC-SHA-1
Symmetric ciphers	DES
	TDES
	AES
	RC2
	RC4
	EES
Public key schemes	DSA
	ECDSA
	RSA
Random numbers	PRNG
	Password Generator

Table 7: Power-up Self-Test

[4] The Power-Up Self-Test consists of a software/firmware integrity test plus a known-answer test for all supported FIPS-approved algorithms and some non-FIPS-approved algorithms

11.2 Conditional Self-Tests

Conditional self-tests are performed when certain specific conditions exist. The following self-tests are available.

11.2.1 Continuous Random Number Generation Test

Each call to the random number generator uses the continuous RNG test. If two successive PRNG output blocks are ever equal, the CDK aborts the current operation and enters a failure state.

11.2.2 Pair-wise Self-Tests

All RSA, DSA and ECDSA public/private key pairs are automatically tested for pair-wise consistency upon generation by calling the check() method of the Key class.

11.2.3 On-Demand Self-Tests

To ensure the integrity of specific algorithms, the user can run on-demand any of the known-answer self-tests listed in Table 7. There is also a master test function that the user can call to run all known-answer tests. These tests perform consistency checks and/or run known answer tests against the ISC CDK's implementation of a particular algorithm. These functions take a single parameter indicating whether to run a fast check on only one known value or to run all known answer tests available for that algorithm, return 0 on success and a non-zero value on failure.

12 FIPS 140-1 Mode

When the ISC CDK is in FIPS 140-1 Mode, only FIPS 140-1 approved algorithms are used and no other. It is the responsibility of the CO to properly configure the computer system, operating system, applications, and the ISC CDK to operate in a secure FIPS 140-1 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.h. The CO can allow or disallow User access to the ISC CDK's built-in integrity tests, control the loading of applications by the User, and restrict User access to only FIPS 140-1 approved algorithms. A User has access to only those services provided by the ISC CDK that are exposed for his use by the CO.

In order to operate in a FIPS 140-1 compliant manner, the CO must ensure that applications loaded by the operating system are limited in their use of the ISC 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 140-1 compliant application
- Only classes that are commented as being usable by a FIPS 140-1 compliant application
- Only modes that are commented as being usable by a FIPS 140-1 compliant application
- Only variants that are commented as being usable by a FIPS 140-1 compliant application