IBM® Crypto for C (ICC) Version 8.0.0

FIPS 140-2 Non-Proprietary Security Policy, version 1.2

September 30, 2010

Table Of Contents

Contents

2. References and Abbreviations

2.1 References

Author	Title
NIST	FIPS PUB 140-2: Security Requirements For Cryptographic Modules, May 2001
NIST	[Derived Test Requirements for FIPS PUB 140-2, November 2001
NIST	Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program

2.2 Abbreviations

ANS.1	Abstract Syntax Notation One. A notation for describing data structures.
AES	The Advanced Encryption Standard. The AES is intended to be issued as a FIPS standard and will replace DES. In January 1997 the AES initiative was announced and in September 1997 the public was invited to propose suitable block ciphers as candidates for the AES. NIST is looking for a cipher that will remain secure well into the next century. NIST selected Rijndael as the AES algorithm.
AES_CCM	AES counter mode as documented in NIST SP800-38C
AES_GCM	AES Galois counter mode as documented in NIST SP800-38D
Camellia	A 128 bit block cipher developed by NTT
CMAC	Cipher based MAC. As documented in NIST SP800-38B
CMVP	(The NIST) Cryptographic Module Validation Program; an integral part of the Computer Security Division at N IST, the CMVP encompasses validation testing for cryptographic modules and algorithms
Crypto	Cryptographic capability/functionality
CSEC	The Communications Security Establishment Canada; An entity operating under the Canadian Department of National Defense, CSEC provides technical advice, guidance and services to the Government of Canada to maintain the security of its information and information infrastructures. The CMV Program was established by NIST and CSEC in July 1995.
DER	Distinguished Encoding Rules
DES	The Data Encryption Standard, an encryption block cipher defined and endorsed by the U.S. government in 1977 as an official standard; the details can be found in the latest official FIPS (Federal Information Processing Standards) publication concerning DES. It was originally developed at IBM. DES has been extensively studied since its publication and is the most well-known and widely used cryptosystem in the world.
DH	Diffie-Hellman key agreement protocol (also called exponential key agreement) was developed by Diffie and Hellman in 1976 and published in the ground-breaking paper “New Directions in Cryptography”. The protocol allows two users to exchange a secret key over an insecure medium without any prior secrets.
DSA	The Digital Signature Algorithm (DSA) was published by NIST in the Digital Signature Standard (DSS)

ECC	Elliptic curve cryptography. A potentially faster and more secure replacement for prime field based asymmetric algorithms such as RSA and Diffie-Hellman
ECDH	Elliptic curve Diffie-Hellman
ECDSA	Elliptic Curve digital signature algorithm
ICC	IBM Crypto for C-language is a general-purpose cryptographic provider module.
Libcrypt	The cryptography engine of OpenSSL.
MD2 MD4 MD5	MD2, MD4, and MD5 are message-digest algorithms developed by Rivest. They are meant for digital signature applications where a large message has to be "compressed" in a secure manner before being signed with the private key. All three algorithms take a message of arbitrary length and produce a 128-bit message digest. While the structures of these algorithms are somewhat similar, the design of MD2 is quite different from that of MD4 and MD5 and MD2 was optimized for 8-bit machines, whereas MD4 and MD5 were aimed at 32-bit machines. Description and source code for the three algorithms can be found as Internet RFCs 1319 - 1321.
MDC2	A seldom used hash algorithm developed by IBM
NIST	(The) National Institute of Standards and Technology; NIST is a non-regulatory federal agency within the U.S. Commerce Department's Technology Administration. NIST's mission is to develop and promote measurement, standards, and technology to enhance productivity, facilitate trade, and improve the quality of life. NIST oversees the Cryptographic Module Validation Program.
OpenSSL	A collaborative effort to develop a robust, commercial-grade, full-featured and Open Source toolkit implementing the Secure Socket Layer (SSL V1/V3) and Transport Layer Security (TLS V1) protocols.
PKCS#1	A standard that describes a method for encrypting data using the RSA public-key crypto system
PRNG	Pseudo-Random number generator. Essentially a sequence generator which, if the internal state is unknown, is unpredictable and has good distribution characteristics.
RC2	A variable key-size block cipher designed by Rivest for RSA Data Security. "RC" stands for "Ron's Code" or "Rivest's Cipher." It is faster than DES and is designed as a "drop-in" replacement for DES. It can be made more secure or less secure than DES against exhaustive key search by using appropriate key sizes. It has a block size of 64 bits and is about two to three times faster than DES in software. The algorithm is confidential and proprietary to RSA Data Security. RC2 can be used in the same modes as DES.
RC4	A stream cipher designed by Rivest for RSA Data Security. It is a variable key-size stream cipher with byte-oriented operations.
RSA	A public-key cryptosystem for both encryption and authentication; it was invented in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman.
SHA-1	The Secure Hash Algorithm, the algorithm specified in the Secure Hash Standard (SHS), was developed by NIST and published as a federal information processing standard. SHA-1 was a revision to SHA that was published in 1994. The revision corrected an unpublished flaw in SHA.
SHA-2	A set of hash algorithms intended as an upgrade to SHA-1. These support a wider range of hash sizes than SHA-1 and should be more secure
Triple DES	Based on the DES standard; the plaintext is, in effect, encrypted three times. Triple DES (TDEA), as specified in ANSI X9.52, is recognized as a FIPS approved

	algorithm.
TRNG	True Random number generator. A random number generator using an entropy source. May have worse distribution characteristics than a PRNG, but its output cannot be predicted even with knowledge of its previous state.

3 Introduction

This document is a non-proprietary FIPS 140-2 Security Policy for the IBM Crypto for C (ICC), Version 8.0.0 cryptographic module. It contains a specification of the rules under which the module must operate and describes how this module meets the requirements as specified in FIPS PUB 140-2 (Federal Information Processing Standards Publication 140-2) for a Level 1 multi-chip standalone software module. This Policy forms a part of the submission package to the testing lab.

- FIPS 140-2 specifies the security requirements for a cryptographic module protecting sensitive information. Based on four security levels for cryptographic modules this standard identifies requirements in eleven sections. For more information about the standard visit http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf.For more information on the FIPS 140-2 standard and validation program please refer to the NIST website at http://csrc.nist.gov/cryptval/.
- For more information about IBM software please visit http://www.ibm.com

3.1 Purpose of the Security Policy

- There are three major reasons that a security policy is required. It is required for FIPS 140-2 validation. It allows individuals and organizations to determine whether the cryptographic module, as implemented, satisfies the stated security policy describes the capabilities, protection, and access rights provided by the cryptographic module, allowing individuals and organizations to determine whether it will meet their security requirements.

3.2 Target Audience

This document is intended to be part of the package of documents that are submitted for FIPS validation. It is intended for the following people:

- Developers working on the release
- Product Verification
- Documentation
- Product and Development Managers

4. Cryptographic Module Definition

This section defines the software cryptographic module that is being submitted for validation to FIPS PUB 140-2, level 1.

The IBM Crypto for C v8.0.0 (ICC) cryptographic module is implemented in the C programming language. It is packaged as dynamic (shared) libraries usable by

applications written in a language that supports C language linking conventions (e.g. C, C++, Java, Assembler, etc.) for use on commercially available operating systems. The ICC allows these applications to access cryptographic functions using an Application Programming Interface (API) provided through an ICC import library and based on the API defined by the OpenSSL group. The physical boundary of the cryptographic module is defined to be the enclosure of the computer that runs the ICC software.

The cryptographic module provided to the customer consists of:

- **ICC static stub:** static library (object code) that is linked into the customer's application, performs the integrity checks on the Crypto Module and communicates with it. C headers (source code) containing the API prototypes and other definitions needed for linking the static library.
- **ICC shared library:** Shared library (executable code) containing the IBM code needed to meet FIPS and functional requirements not provided within the OpenSSL libraries (e.g. TRNG, PRNG, self-tests, startup/shutdown). Contains also **zlib**, used for TRNG entropy estimation
- **Libcrypt:** Shared library (executable code) containing the OpenSSL cryptographic library.

There is a different set of the cryptographic module (static and shared libraries) for each of the target platforms.

As outlined in G.5 of the Implementation Guidance for FIPS 140-2 (March 10, 2009 Update), the module maintains its compliance on other operating systems, provided:

- The operating system meets the operational environment requirements at the module's level of validation
- The module does not require modification to run in the new environment

ICC **was** tested and validated on a machine running the Microsoft Windows Server 2008® 64-bit and 32-bit operating system (x86-64). The software module maintains compliance when running on other versions of Microsoft Windows.

ICC **was** tested and validated on a machine running the AIX® 6.1 64-bit operating system (PowerPC 5 64). The software module maintains compliance when running on other versions of AIX.®

ICC **was** tested and validated on a machine running the Solaris® 10 64-bit operating system (UltraSparc-64). The software module maintains compliance when running on other versions of Solaris®.

ICC **was** tested and validated on a machine running the Red Hat Linux Enterprise Server 5 64-bit and 32-bit operating system (x86-64). The software module maintains compliance

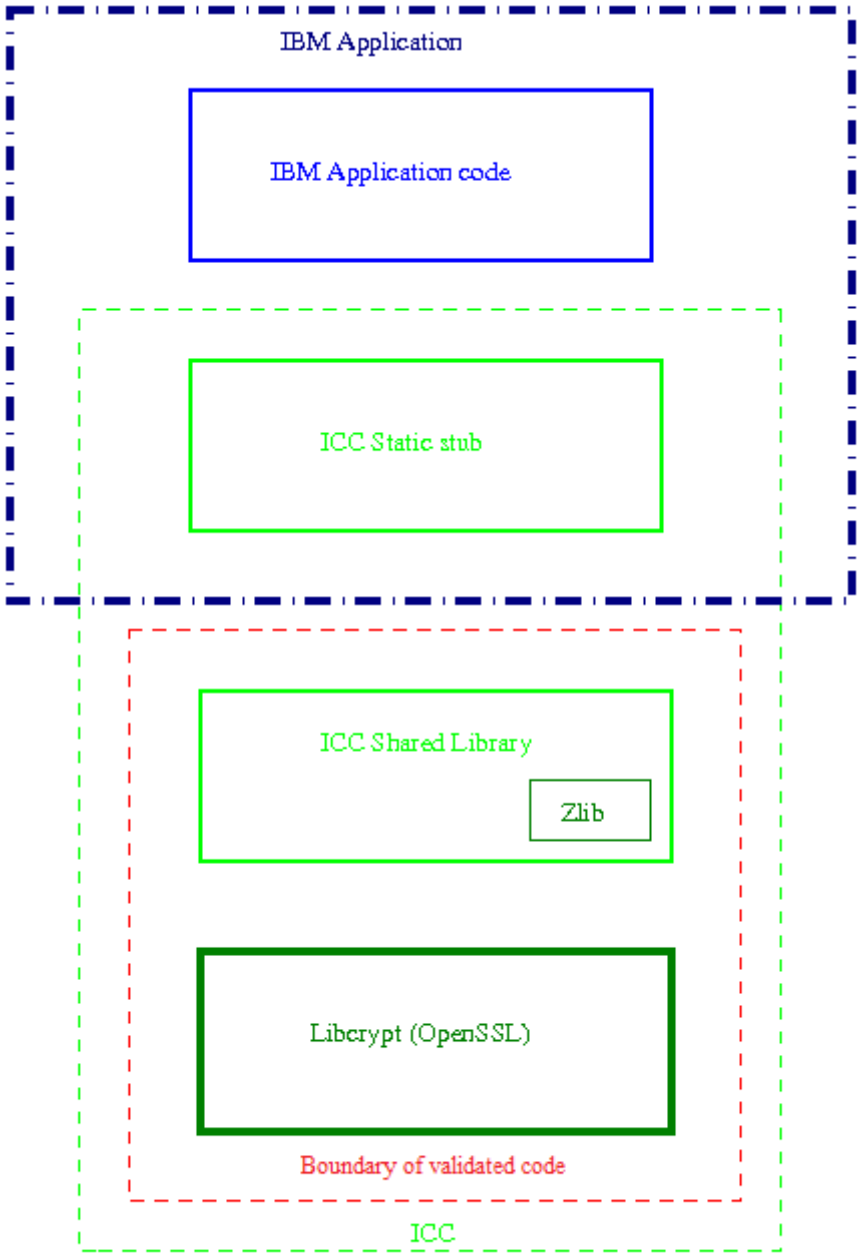
when running on other Linux based operating systems.

ICC **was** tested and validated on a machine running the Red Hat Linux Enterprise Server 5 64-bit operating system (PowerPC-64). The software module maintains compliance when running on other Linux based operating systems.

ICC **was** tested and validated on a machine running the Red Hat Linux Enterprise Server 5 64-bit operating system (zSeries-64). The software module maintains compliance when running on other Linux based operating systems.

4.1 Cryptographic Module Boundary

The relationship between ICC and IBM applications is shown in the following diagram. ICC comprises a static stub linked into the IBM application which bootstraps and validates the two cryptographic shared libraries.



ICC Functional decomposition

- **IBM Application** - The IBM application using ICC. This contains the application code, and the ICC static stub.
- **IBM Application code** - The program using ICC to perform cryptographic functions
- **ICC Static stub -** Linked in to the application, this contains signatures of the ICC and OpenSSL shared libraries, plus code to bootstrap the loading of the shared libraries.
- **ICC shared library** - This contains IBM code needed to meet FIPS and functional requirements not provided within the OpenSSL libraries. TRNG, PRNG, self test,

startup/shutdown.

- **zlib** - A statically linked copy of zlib used for TRNG entropy estimation
- **Libcrypt** - The OpenSSL cryptographic shared library.
- **The logical boundary of Cryptographic Module -** consists of ICC Static stub, ICC shared library, zlib and Libcrypt bounded by the dashed green line in the above figure. While the signatures of the ICC components used for the integrity check of the ICC during its initialization are contained in the ICC static stub**,** all of the validated cryptographic algorithms are implemented in ICC shared library, zlib and Libcrypt whose binary object code is enclosed in the dashed red lines in the above Figure**.**

5. FIPS 140-2 Specifications

5.1 Ports and Interfaces

The ICC meets the requirements of a multi-chip standalone module. Since the ICC is a software module, its interfaces are defined in terms of the API that it provides. Data Input Interface is defined as the input data parameters of those API functions that accept, as their arguments, data to be used or processed by the module. The return value or arguments of appropriate types, data generated or otherwise processed by the API functions to the caller constitute Data Output Interface. Control Input Interface is comprised of the call used to initiate the module and the API functions used to control the operation of the module as well as environment variables.

Status Output Interface is defined as the API functions ICC_GetStatus and ICC_GetValue that provide information about the status of the module. The functions ICC_GetStatus and ICC_GetValue may be called anytime after ICC_Init to indicate the status of the ICC module.

5.2 Roles, Services and Authentication

5.2.1 Roles and Authentication

The ICC implements the following two roles: Crypto-Officer role and User role (there is **no** Maintenance Role). The Operating System (OS) provides functionality to require any user to be successfully authenticated prior to using any system services. However, the Module does not support user identification or authentication that would allow for distinguishing users between the two supported roles. Only a single operator assuming a particular role may operate the Module at any particular moment in time. The OS authentication mechanism must be enabled to ensure that none of the Module's services are available to users who do not assume an authorized role.

The Module does not identify nor authenticate any user (in any role) that is accessing the Module. This is only acceptable for a FIPS 140-2, Security Level 1 validation.

The two roles are defined per the FIPS140-2 standard as follows:

1. **Crypto Officer -** any entity that can access services implemented in the Module and, install and initialize the Module
2. **User** - any entity that can access services implemented in the Module.

Table 1, below, lists the Roles and their associated authentication:

Role	Authentication Type	Authentication Data	Authentication Mechanism	Authentication Strength
Crypto Officer	Not required	Not required	Not required	Not required
User	Not required	Not required	Not required	Not required

Table 1: Roles and Services

5.2.2 Authorized Services

An operator is implicitly assumed in the User or Cryptographic Officer role based upon the operations chosen. Both User and Cryptographic Officer can call all services implemented in the Module as listed in Table 2 below. Only Cryptographic Officer can install and initialize the Module. If the operator installs and/or initializes the Module, then he is in the Cryptographic Officer role. Otherwise, the operator is in the User role.

The following table provides a summary of the services and access supported by the ICC.

Service	Notes	Modes	FIPS-Approved If yes, Cert #	Cryptographic Keys, CSPs and access	
Symmetric Algorithms					
AES	128, 192,or 256 bit keys (FIPS 197) Encrypt/Decrypt	CBC,ECB, CFB1, CFB8, CFB128, OFB	Yes Cert # 1318, 1319, 1320, 1321, 1322, 1323, 1324, 1325, 1326, 1327, 1328, 1329, 1330, 1331	AES Symmetric key	Read/ Write

TDES	168 bit keys Encrypt/Decrypt	CBC, ECB, CFB64, OFB	Yes Cert # 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930	TDES Symmetric key	Read/ Write
Public Key Algorithms					
DSA Key/Parameter Generation	1024 bit modulus (FIPS 186-2 key size)	N/A	Yes Cert # 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435	DSA public and private key	Write
DSA Key/Parameter Generation	512-bit, 1536-bit 2048-bit, 3072-bit modulus	N/A	No	DSA public and private key	Write
DSA Signature Generation	1024 bit modulus	N/A	Yes Cert # 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435	DSA private key	Read
DSA Signature Generation	512-bit, 1536-bit, 2048-bit, 3072-bit modulus	N/A	No	DSA private key	Read
DSA Signature Verification	1024 bit modulus	N/A	Yes Cert # 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435	DSA public key	Read
DSA Signature Verification	512-bit, 1536-bit 2048-bit, 3072-bit modulus	N/A	No	DSA public key	Read
ECDSA KeyPair	P: 192 to 521 K: 163 to 571 B: 163 to 571	N/A	Yes Cert # 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170	ECDSA public and private key	Write
ECDSA PKV	P: 192 to 521 K: 163 to 571 B: 163 to 571	N/A	Yes Cert # 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170	ECDSA key material	Write

ECDSA Signature Generation	P: 192 to 521 K: 163 to 571 B: 163 to 571	N/A	Yes Cert # 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170	ECDSA private key	Read
ECSDA Signature Verification	P: 192 to 521 K: 163 to 571 B: 163 to 571	N/A	Yes Cert # 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170	ECDSA public key	Read
RSA Key Generation	ANSI X9.31 (1024 to 4096 bits)	N/A	Yes Cert # 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643	RSA public and private key	Write
RSA Signature Generation	PKCS#1.5 (1024 to 4096 bits) (SHA-1,SHA-224,SHA-256,SHA-384,SHA-512)	N/A	Yes Cert # 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643	RSA private key	Read
RSA Signature Verification	PKCS#1.5 (1024 to 4096 bits) (SHA-1,SHA-224,SHA-256,SHA-384,SHA-512)	N/A	Yes Cert # 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643	RSA public key	Read
RSA Key Wrapping	Encrypt / Decrypt (1024 to 4096 bits)	N/A	No	RSA public and private key	Read
Diffie-Hellman (DH)	1024 to 4096 bits) modulus	Key agreement and Key Generation	No	DH public and private key	Read/ write
ECDH	163 to 571 bits for curve; (SP 800-56A)	Key agreement and Key Generation	No	ECDH public and private key	Read/ write
Hash Functions					

SHA-1	FIPS 180-1	N/A	Yes Cert # 1204,1205, 1206,1207, 1208,1209, 1210,1211, 1212,1213, 1214,1215, 1216,1217	None	N/A
SHA-224 SHA-256 SHA-384 SHA-512	FIPS 180-2 -SHA-2 algorithms	N/A	Yes Cert # 1204,1205, 1206,1207, 1208,1209, 1210,1211, 1212,1213, 1214,1215, 1216,1217	None	N/A
MD2		N/A	No	None	N/A
MD4		N/A	No	None	N/A
MD5	Only allowed during TLS handshake	N/A	No	None	N/A
MDC2		N/A	No	None	N/A
RIPEMD		N/A	No	None	N/A
Message Authentication Codes (MACs)					
HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	FIPS 198, 198a	N/A	Yes Cert # 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779	HMAC-SHA-1 key, HMAC-SHA-224 key, HMAC-SHA-256 key, HMAC-SHA-384 key, HMAC-SHA-512 key	Write
HMAC-MD5	N/A	N/A	No	HMAC-MD5 key	Write
CMAC-AES-128, CMAC-AES-192, CMAC-AES-256	FIPS 197	N/A	Yes Cert # 1318, 1319, 1320, 1321, 1322, 1323, 1324, 1325, 1326, 1327, 1328, 1329, 1330, 1331	CMAC-AES-128 key, CMAC-AES-192 key, CMAC-AES-256 key	Write
CMAC-TDES3 (168-bit)	FIPS 197	CBC	Yes Cert # 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930	CMAC-TDES3 key (168-bit)	Write

AES_CCM	FIPS 197,	N/A	Yes Cert # 1318, 1319, 1320, 1321, 1322, 1323, 1324, 1325, 1326, 1327, 1328, 1329, 1330, 1331	AES_CCM key	Write
AES_GCM (96-bit IV)	FIPS 197, SP800-38D	N/A	Yes Cert # 1318, 1319, 1320, 1321, 1322, 1323, 1324, 1325, 1326, 1327, 1328, 1329, 1330, 1331	AES_GCM key	Write
Random Number Generation					
DRBG 800-90	SP 800-90	HMAC_DRBG (SHA-1, SHA-224, SHA-256), CTR_DRBG (AES-128-ECB, AES-192-ECB, AES-256-ECB with/without derivation function and prediction resistance supported)	Yes Cert# 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47	Seed	Write
Other functions (Not FIPS Approved – Not allowed in FIPS mode)					

Service	**Notes**	**FIPS-Approved**
DES encryption/decryption	Cipher algorithm	No
CAST encryption/decryption	Cipher algorithm	No
Camellia	Cipher algorithm	No
Blowfish	Cipher algorithm	No
RC4	Cipher algorithm	No
RC2 encryption/decryption	Cipher algorithm	No

Table 2: Services and Access

When operating in FIPS approved mode no unapproved algorithms may be used. There is an allowance for key establishment and exchange to use any algorithm when operating in FIPS approved mode (under the phrase “commercially available methods may be used”). The ICC will not limit the algorithms but in the ICC policy it will list the FIPS approved algorithms, the allowances/exceptions (e.g. SSL key exchange and

establishment) and the algorithms that are not FIPS approved.

5.2.3 Access Rights Within Services

An operator performing a service within any role can read/write cryptographic keys and critical security parameters (CSP) only through the invocation of a service by use of the Cryptographic Module API. Each service within each role can only access the cryptographic keys and CSPs that the service's API defines. The following cases exist:

- A cryptographic key or CSP is provided to an API as an input parameter; this indicates read/write access to that cryptographic key or CSP.
- A cryptographic key or CSP is returned from an API as a return value; this indicates read access to that cryptographic key or CSP.

The details of the access to cryptographic keys and CSPs for each service are indicated in the rightmost column of Table 2. The indicated access rights apply to both the User role and Cryptographic Officer role who invokes services.

5.2.4 Operational Rules and Assumptions

The following operational rules must be followed by **any user** of the Module:

1. The Module is to be used by a single human operator at a time and may not be actively shared among operators at any period of time.
2. The OS authentication mechanism must be enabled in order to prevent unauthorized users from being able to access system services.
3. All keys entered into the module must be verified as being legitimate and belonging to the correct entity by software running on the same machine as the module.
4. Since the ICC runs on a general-purpose processor all main data paths of the computer system will contain cryptographic material. The following items need to apply relative to where the ICC will execute:
   - Virtual (paged) memory must be secure (local disk or a secure network)
   - The system bus must be secure.
   - The disk drive that ICC is installed on must be in a secure environment.
5. The above rules must be upheld at all times in order to ensure continued system security and FIPS 140-2 mode compliance after initial setup of the validated configuration. If the module is removed from the above environment, it is assumed not to be operational in the validated mode until such time as it has been returned to the above environment and re-initialized by the user to the validated condition.

NOTE: It is the responsibility of the Crypto-Officer to configure the operating system to operate securely and ensure that only a single operator may operate the Module at any particular moment in time.

The services provided by the Module to a User are effectively delivered through the use of

appropriate API calls. In this respect, the same set of services is available to both the User and the Crypto-Officer.

When a client process attempts to load an instance of the Module into memory, the Module runs an integrity test and a number of cryptographic functionality self-tests. If all the tests pass successfully, the Module makes a transition to “FIPS Operation” state, where the API calls can be used by the client to obtain desired cryptographic services. Otherwise, the Module enters to “Error” state and returns an error to the calling application. When the Module is in “Error” state, no FIPS-approved services should be available, and all of data input and data output except the status information should be inhibited.

5.3 Operational Environment

Along with the conditions stated above in paragraph 5.2.4 (“Operational Rules and Assumptions”), the criteria below must be followed in order to achieve, and maintain, a FIPS 140-2 mode of operation:

5.3.1 Assumptions

The following assumptions are made about the operating environment of the cryptographic module:

1. The prevention of unauthorized reading, writing, or modification of the module’s memory space (code and data) by an intruder (human or machine) is assured.
2. The prevention of Replacement or modification of the legitimate cryptographic module code by an intruder (human or machine) is assured.
3. The module is initialized to the FIPS 140-2 mode of operation

5.3.2 Installation and Initialization

The following steps must be performed to install and initialize the module for operating in a FIPS 140-2 compliant manner:

1. The operating system must be configured to operate securely and to prevent remote login. This is accomplished by disabling all services (within the Administrative tools) that provide remote access (e.g. – ftp, telnet, ssh, and server) and disallowing multiple operators to log in at once.
2. The operating system must be configured to allow only a single user. This is accomplished by disabling all user accounts except the administrator. This can be done through the Computer Management window of the operating system.
3. The module must be initialized to operate in FIPS 140-2 mode; this is done by the following calling sequence:
   - ICC_Init to create the crypto module context.
   - ICC_SetValue to set the parameter FIPS_APPROVED_MODE to "on"

- ICC_Attach to load the library, perform the self-tests and turn the crypto module in an operational state

5.4 Cryptographic Key Management

5.4.1 Implemented Algorithms

The IBM Crypto for C (ICC) version 8.0.0 supports the algorithms (and modes, as applicable) listed above in Table 2 in section 5.2.2.

5.4.2 Key Generation

Key generation has dependency on random number generator DRBG 800-90, which is detailed below. DRBG 800-90 is used to generate RSA/DSA/ECDSA/DH/ECDH key pairs as well as AES keys and TDES keys. Key sizes for AES keys can be 128-bit, 192-bit or 256-bit. Key size for TDES key is 3-key TDES key.

In FIPS mode, RSA key generation is carried out in accordance with the algorithms described in ANSI X9.31, the code used is the same as that used in the openssl-fips-1.2 sources.

Also in FIPS mode, DSA and ECDSA key generation is carried out in accordance with the algorithms described in FIPS 186-2 and ANSI X9.62, respectively.

In non-FIPS mode, the normal OpenSSL (PKCS style) RSA key generation is used.

The ICC provides X9.31 and PKCS#1 compatible algorithms for processing signatures (creating and verifying) the function of which is available as specified in the API's in this document. These algorithms are also available for encryption and decryption where it is used as PKCS#1 compatible.

In addition, there is a set of lower level interfaces for encryption and decryption where the algorithm can be used as PKCS#1 compatible but it also allows other types of padding operations to be used. See RSA encryption functions for the definition of the functions and for the list of padding modes.

The DRBG 800-90 used by default internally will be of 256 bits effective strength.

DRBG 800-90 Random Number Generators

The following describes the random number generation.

- Seed must have an incremental element/quality (time stamp) to it.

- Seed must have an entropy estimate/shutdown on failure. A continuous entropy estimate of the raw entropy source is obtained by using a compression function. See EntropyEstimator()
- Generation of seed. See ICC_GenerateRandomSeed()
- Seed is fed to DRBG 800-90. See ICC_RAND_seed()
- The minimum guaranteed entropy is 0.5 bits/bit. See EntropyEstimator()
- The current entropy estimate is available via ICC_GetValue()
- The initial state of the TRNG is time/date mixed with TRNG source and is unique for each invocation of ICC.
- Previous state of the TRNG is saved as a hash, therefore the previous state is not obtainable from the residual program image without reversing a (lossy) hash function. See META_GenerateRandomSeed()
- State of the DRBG 800-90 is cleared on exit. See OpenSSL_Cleanup(), ICC_RNG_CTX_free()
- State of the TRNG entropy estimation test is cleared on exit. See ICCUnload(), CleanupEntropyEstimator()

5.4.3 Key Establishment

The ICC uses the following as key establishment methodologies:

- Diffie-Hellman (DH) with 1024-4096 bits key to provide 80-150 bits of security strength
- Elliptic Curve Diffie-Hellman (ECDH) with 163-571 bits curve to provide 80-256 bits of security strength
- RSA Encrypt/Decrypt for Key Wrapping with 1024-4096 bits of key to provide 80-150 bits of security strength.

5.4.4 Key Entry and Output

The ICC module does not support manual key entry or intermediate key generation key output. In addition, the ICC module does not produce key output in plaintext format outside its physical boundary.

5.4.5 Key Storage

The module does not provide any long-term key storage and no keys are ever stored on the hard disk.

5.4.6 Key Zeroization

ICC modifies the default OpenSSL scrubbing code to zero objects instead of filling with pseudo random data and adds explicit testing for zeroization.

Key zeroization is performed via the following API calls:

- ICC_BN_clear_free() , ICC_BN_CTX_free() - Low-level arithmetic functions
- ICC_EVP_CIPHER_CTX_free() - Symmetric ciphers
- ICC_RSA_free() - RSA
- ICC_DSA_free() - DSA
- ICC_DH_free() - DH
- ICC_EVP_PKEY_free – Generic asymmetric key contexts (RSA, DSA and DH keys)
- ICC_HMAC_CTX_free() - HMAC
- ICC_EC_KEY_free() – Low-level EC function, as well as ECDSA/ECDH
- ICC_CMAC_CTX_free() - CMAC
- ICC_AES_GCM_CTX_free() - AES-GCM
- ICC_RNG_CTX_free()

.

5.5 Self-Tests

The ICC implements a number of self-tests to check proper functioning of the module. This includes power-up self-tests (which are also callable on demand) and conditional self-tests. The self-test can be initiated by calling the function ICC_SelfTest, which returns the operational status of the module (after the self-tests are run) and an error code with description of the error (if applicable). Additionally, when the module is performing self-tests, no API functions are available and no data output is possible until the self-tests are successfully completed.

5.5.1 Show Status

Two functions indicate the status of the ICC module:

- ICC_GetStatus
  - Shows the state of the ICC module

- ICC_GetValue
  - Get the ICC version
  - Inform whether the ICC module is in FIPS / non-FIPS mode
  - Current entropy estimate for the DRBG 800-90 seed source

Both functions may be called anytime after ICC_Init.

5.5.2 Startup Tests

The module performs self-tests automatically when the API function ICC_Attach is called or on demand when the API function ICC_SelfTest is called.

Whenever the startup tests are initiated the module performs the following; if **any** of these tests fail, the module enters the error state.

- **Integrity Test of Digital Signature:** The ICC uses an Integrity test which uses a 2048-bit CAVS-validated RSA public key (PKCS#1.5) and SHA-256 hashing. This RSA public key is stored inside the static stub and relies on the operating system for protection.
- **Cryptographic algorithm tests:**

At startup, a Known Answer Test (encryption & decryption) is performed for the following FIPS approved and non-approved algorithms:

- Triple DES – CBC
- AES 256 – CBC
- RSA – PKCS#1.5 signature generation/verification

One way known answer tests are performed for the following FIPS approved algorithms:

- SHA-1
- SHA-224
- SHA-256
- SHA-384
- SHA-512
- SHA-1 HMAC
- SHA-224 HMAC
- SHA-256 HMAC
- SHA-384 HMAC
- SHA-512 HMAC
- CMAC-AES-256-CBC
- RSA signature generation
- ECDSA signature generation
- AES_GCM
- AES_CCM
- DRBG 800-90

Signature generation and verification tests with known keys and data are performed on the following algorithms

- RSA 1024
- DSA 1024
- ECDSA P-384

In FIPS mode a failure occurred during self-tests is considered a fatal error, in non-FIPS modes the failing algorithms become unavailable. Additionally, if any of the self-tests fail, the FIPS mode will not be enabled and none of the FIPS-approved algorithms will be available.

5.5.3 Conditional Tests

Pairwise consistency test for public and private key generation. - The consistency of the keys is tested by the calculation and verification of a digital signature. If the digital signature cannot be verified, the test fails. Pairwise consistency tests are performed on the following algorithms :

- DSA
- ECDSA
- RSA

Continuous RNG test – The module implements a Continuous RNG test as follows:

DRBG 800-90

- The DRBG 800-90 generate a minimum of 8 bytes per request. If less than 8 bytes are requested, excess data is discarded. The first 8 bytes of every request is compared with the last 8 bytes requested, if the bytes match an error is generated. For the first request made to any instantiation of a DRBG 800-90, two internal 8 byte cycles are performed. The DRBG 800-90 relies on the environment (i.e. proper shutdown of the shared libraries) for resistance to retrospective attacks on data. The DRBG 800-90 performs known answer tests when first instantiated and health checks at intervals as specified in the standard.

5.5.4 Severe Errors

When severe errors are detected (e.g. self-test failure or a conditional test failure) then all security related functions shall be disabled and no partial data is exposed through the data output interface. The only way to transition from the error state to an operational state is to reinitialize the cryptographic module (from an uninitialized state). The error state can be retrieved via the status interface (see Section 5.5.1 above).

5.6 Design Assurance

The ICC module design team utilizes IBM's Configuration Management Version Control (CMVC) system.

CMVC integrates four facets of the software development process in a distributed development environment to facilitate project-wide coordination of development activities across all phases of the product development life cycle:

1. Configuration Management – the process of identifying, managing and controlling software modules as they change over time.
2. Version Control – the storage of multiple versions of a single file along with information about each version.
3. Change Control – centralizes the storage of files and controls changes to files through the process of checking files in and out.
4. Problem Tracking – the process of effectively tracking all reported defects and proposed design changes through to their resolution and implementation.

Files are stored in a file system on the server by means of a version control system. All other development data is stored in a relational database on the CMVC server. A CMVC client is a workstation that runs the CMVC client software (or browser for the web interface) to access the information and files stored on a CMVC server.
CMVC is used o perform the following tasks:

1. Organizing Development Data
2. Configuring CMVC Processes
3. Reporting Problems and Design Changes
4. Tracking Features and Defects

All source code is tracked using CMVC; documents are available in Lotus Notes database “Team Rooms” with version numbers assigned by document owner.

CMVC monitors changes with defects, features, and integrated problem tracking. Each of these restricts file changes so that they are made in a systematic manner. CMVC can require users to analyze the time and resources required to make changes, verify changes, and select files to be changed, approve work to be done, and test the changes. The requirements for changes are controlled by processes. Family administrators can create processes for components and releases to use, configuring them from CMVC sub processes.

Finally, the CMVC administrator policy mandates a regular audit of access check of all user accounts.

5.7 Mitigation Of Other Attacks

The cryptographic module is not designed to mitigate any specific attacks.

6. API Functions

The module API functions are fully described in the *IBM Crypto for C (ICC) Design Document.*

The following is the list of the API functions supported.

- ICC_GetStatus
- ICC_Init (CO)
- ICC_SetValue (CO)
- ICC_GetValue
- ICC_Attach (CO)
- ICC_Cleanup
- ICC_SelfTest
- ICC_GenerateRandomSeed
- ICC_OBJ_nid2sn
- ICC_EVP_get_digestbyname
- ICC_EVP_get_cipherbyname
- ICC_EVP_MD_CTX_new
- ICC_EVP_MD_CTX_free
- ICC_EVP_MD_CTX_init
- ICC_EVP_MD_CTX_cleanup
- ICC_EVP_MD_CTX_copy
- ICC_EVP_MD_type
- ICC_EVP_MD_size
- ICC_EVP_MD_block_size
- ICC_EVP_MD_CTX_md
- ICC_EVP_Digestinit
- ICC_EVP_DigestUpdate
- ICC_EVP_DigestFinal
- ICC_EVP_CIPHER_CTX_new
- ICC_EVP_CIPHER_CTX_free
- ICC_EVP_CIPHER_CTX_init
- ICC_EVP_CIPHER_CTX_cleanup
- ICC_EVP_CIPHER_CTX_set_key_length
- ICC_EVP_CIPHER_CTX_set_padding
- ICC_EVP_CIPHER_block_size
- ICC_EVP_CIPHER_key_length
- ICC_EVP_CIPHER_iv_length
- ICC_EVP_CIPHER_type
- ICC_EVP_CIPHER_CTX_cipher
- ICC_DES_random_key
- ICC_DES_set_odd_parity
- ICC_EVP_EncryptInit
- ICC_EVP_EncryptUpdate
- ICC_EVP_EncryptFinal
- ICC_EVP_DecryptInit
- ICC_EVP_DecryptUpdate
- ICC_EVP_DecryptFinal
- ICC_EVP_OpenInit
- ICC_EVP_OpenUpdate
- ICC_EVP_OpenFinal
- ICC_EVP_SealInit
- ICC_EVP_SealUpdate
- ICC_EVP_SealFinal
- ICC_EVP_SignInit
- ICC_EVP_SignUpdate
- ICC_EVP_SignFinal
- ICC_EVP_VerifyInit
- ICC_EVP_VerifyUpdate
- ICC_EVP_VerifyFinal
- ICC_EVP_ENCODE_CTX_new
- ICC_EVP_ENCODE_CTX_free
- ICC_EVP_EncodeInit
- ICC_EVP_EncodeUpdate
- ICC_EVP_EncodeFinal
- ICC_EVP_DecodeInit
- ICC_EVP_DecodeUpdate
- ICC_EVP_DecodeFinal
- ICC_RAND_bytes
- ICC_RAND_seed
- ICC_EVP_PKEY_decrypt
- ICC_EVP_PKEY_encrypt
- ICC_EVP_PKEY_new
- ICC_EVP_PKEY_free

- ICC_EVP_PKEY_size
- ICC_RSA_new
- ICC_RSA_generate_key
- ICC_RSA_check_key
- ICC_EVP_PKEY_set1_RSA
- ICC_EVP_PKEY_get1_RSA
- ICC_RSA_free
- ICC_RSA_private_encrypt
- ICC_RSA_private_decrypt
- ICC_RSA_public_encrypt
- ICC_RSA_public_decrypt
- ICC_i2d_RSAPrivateKey
- ICC_i2d_RSAPublicKey
- ICC_d2i_PrivateKey
- ICC_d2i_PublicKey
- ICC_EVP_PKEY_set1_DH
- ICC_EVP_PKEY_get1_DH
- ICC_DH_new
- ICC_DH_new_generate_key
- ICC_DH_ check
- ICC_DH_free
- ICC_DH_size
- ICC_DH_compute_key
- ICC_DH_generate_parameters
- ICC_DH_get_PublicKey
- ICC_id2_DHparams
- ICC_d2i_DHparams
- ICC_EVP_PKEY_set1_DSA
- ICC_EVP_PKEY_get1_DSA
- ICC_DSA_dup_DH
- ICC_DSA_sign
- ICC_DSA_verify
- ICC_DSA_size
- ICC_DSA_new
- ICC_DSA_free
- ICC_DSA_generate_key
- ICC_DSA_generate_parameters
- ICC_i2d_DSAPrivateKey
- ICC_d2i_DSAPrivateKey
- ICC_i2d_DSAPublicKey
- ICC_d2i_DSAPublicKey
- ICC_i2d_DSAparams
- ICC_d2i_DSAparams
- ICC_ERR_get_error
- ICC_ERR_peek_error
- ICC_ERR_peek_last_error
- ICC_ERR_error_string
- ICC_ERR_error_string_n
- ICC_ERR_lib_error_string
- ICC_ERR_func_error_string
- ICC_ERR_reason_error_string
- ICC_ERR_clear_error
- ICC_ERR_remove_state
- ICC_BN_bn2bin
- ICC_BN_bin2bn
- ICC_BN_num_bits
- ICC_BN_num_bytes
- ICC_BN_new
- ICC_BN_clear_free
- ICC_RSA_blinding_off
- ICC_EVP_CIPHER_CTX_ctrl
- ICC_RSA_size
- ICC_BN_CTX_new
- ICC_BN_CTX_free
- ICC_BN_mod_exp
- ICC_HMAC_CTX_new
- ICC_HMAC_CTX_free
- ICC_HMAC_Init

- ICC_HMAC_Update
- ICC_HMAC_Final
- ICC_BN_div
- ICC_d2i_DSA_PUBKEY
- ICC_i2d_DSA_PUBKEY
- ICC_ECDSA_SIG_new
- ICC_ECDSA_SIG_free
- ICC_i2d_ECDSA_SIG
- ICC_d2i_ECDSA_SIG
- ICC_ECDSA_sign
- ICC_ECDSA_verify
- ICC_ECDSA_size
- ICC_EVP_PKEY_set1_EC_KEY
- ICC_EVP_PKEY_get1_EC_KEY
- ICC_EC_KEY_new_by_curve_name
- ICC_EC_KEY_new
- ICC_EC_KEY_free
- ICC_EC_KEY_generate_key
- ICC_EC_KEY_get0_group
- ICC_EC_METHOD_get_field_type
- ICC_EC_GROUP_method_of
- ICC_EC_POINT_new
- ICC_EC_POINT_free
- ICC_EC_POINT_get_affine_coordinates_GFp
- ICC_EC_POINT_set_affine_coordinates_GFp
- ICC_EC_POINT_get_affine_coordinates_GF2m
- ICC_EC_POINT_set_affine_coordinates_GF2m
- ICC_EC_KEY_get0_public_key
- ICC_EC_KEY_set_public_key
- ICC_EC_KEY_get0_private_key
- ICC_EC_KEY_set_private_key
- ICC_ECDH_compute_key
- ICC_d2i_ECPrivateKey
- ICC_i2d_ECPrivateKey
- ICC_d2i_ECParameters
- ICC_i2d_ECParameters
- ICC_EC_POINT_is_on_curve
- ICC_EC_POINT_is_at_infinity
- ICC_EC_KEY_check_key
- ICC_EC_POINT_mul
- ICC_EC_GROUP_get_order
- ICC_EC_POINT_dup
- ICC_PKCS5_pbe_set
- ICC_PKCS5_pbe2_set
- ICC_PKCS12_pbe_crypt
- ICC_X509_ALGOR_free
- ICC_OBJ_txt2nid
- ICC_EVP_EncodeBlock
- ICC_EVP_DecodeBlock
- ICC_CMAC_CTX_new
- ICC_CMAC_CTX_free
- ICC_CMAC_Init
- ICC_CMAC_Update
- ICC_CMAC_Final
- ICC_AES_GCM_CTX_new
- ICC_AES_GCM_CTX_free
- ICC_AES_GCM_CTX_ctrl
- ICC_AES_GCM_Init
- ICC_AES_GCM_EncryptUpdate
- ICC_AES_GCM_DecryptUpdate
- ICC_AES_GCM_EncryptFinal
- ICC_AES_GCM_DecryptFinal
- ICC_AES_GCM_GenerateIV
- ICC_GHASH
- ICC_AES_CCM_Encrypt

- ICC_AES_CCM_Decrypt
- ICC_get_RNGbyname
- ICC_RNG_CTX_new
- ICC_RNG_CTX_free
- ICC_RNG_CTX_Init
- ICC_RNG_Generate
- ICC_RNG_ReSeed
- ICC_RNG_CTX_ctrl
- ICC_RSA_sign
- ICC_RSA_verify
- ICC_EC_GROUP_get_degree
- ICC_EC_GROUP_get_curve_GFp
- ICC_EC_GROUP_get_curve_GF2m
- ICC_EC_GROUP_get0_generator
- ICC_i2o_ECPublicKey
- ICC_o2i_ECPublicKey
- ICC_BN_cmp
- ICC_BN_add
- ICC_BN_sub
- ICC_BN_mod_mul
- ICC_EVP_PKCS82PKEY
- ICC_EVP_PKEY2PKCS8
- ICC_PKCS8_PRIV_KEY_INFO_free
- ICC_d2i_PKCS8_PRIV_KEY_INFO
- ICC_i2d_PKCS8_PRIV_KEY_INFO
- ICC_d2i_ECPKParameters
- ICC_i2d_ECPKParameters
- ICC_EC_GROUP_free
- ICC_EC_KEY_set_group
- ICC_EC_KEY_dup
- ICC_SP800_108_get_KDFbyname
- ICC_SP800_108_KDF
- ICC_DSA_SIG_new
- ICC_DSA_SIG_free
- ICC_d2i_DSA_SIG
- ICC_i2d_DSA_SIG
- ICC_RSA_X931_derive_ex
- ICC_Init
- ICC_lib_init (non-FIPS mode)
- ICC_lib_cleanup (non-FIPS mode)
- ICC_MemCheck_start (non-FIPS mode)
- ICC_MemCheck_stop (non-FIPS mode)

The functions marked with (CO) are crypto officer functions.
The functions marked with (non-FIPS mode) are not allowed to be called when running in FIPS mode. They may be usable only in development or test conditions.