IBM® Crypto for C
version 8.4.1.0

FIPS 140-2 Non-Proprietary
Security Policy, version 1.9

July 16, 2015

Table of Contents

1. References and Abbreviations

1.1 References

Reference	Author	Title
FIPS140-2	NIST	FIPS PUB 140-2: Security Requirements For Cryptographic Modules, May 2001
FIPS140-2-DTR	NIST	Derived Test Requirements for FIPS PUB 140-2, November 2001
FIPS140-2-IG	NIST	Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program
FIPS180-4	NIST	SECURE HASH STANDARD (SHS)
FIPS186-4	NIST	Digital Signature Standard (DSS)
FIPS197	NIST	Specification for the ADVANCED ENCRYPTION STANDARD (AES)
FIPS198-1	NIST	The Keyed Hash Message Authentication Code (HMAC)
SP800-38B	NIST	Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication
SP800-38C	NIST	Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality
SP800-38D	NIST	Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC
SP800-38E	NIST	Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices
SP800-56A	NIST	Recommendation for Pair Wise Key Establishment Schemes Using Discrete Logarithm Cryptography
SP800-67	NIST	Recommendation for the Triple Data Encryption Algorithm (TDEA) Block
SP800-131A	NIST	Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths

1.2 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
AES_XTS	AES XEX-based Tweaked-codebook mode with ciphertext Stealing mode as documented in NIST SP800-38E
AES-NI	Intel® Advanced Encryption Standard (AES) New Instructions (AES-NI)
Camellia	A 128 bit block cipher developed by NTT
CMAC	Cipher-based Message Authentication Code, 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
CPACF	CP (central processor) assist for Cryptographic Functions
Crypto	Cryptographic capability/functionality
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.
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. It includes SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256.
Triple-DES	Based on the DES standard; the plaintext is, in effect, encrypted three times. Triple-DES (TDEA), as specified in SP 800-67 TDEA, 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.

2. Introduction

This document is a non-proprietary FIPS 140-2 Security Policy for the IBM Crypto for C (ICC), version 8.4.1.0 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 security level 1 multi-chip standalone software module.

The table below shows the security level claimed for each of the eleven sections that comprise the FIPS 140-2 standard:

FIPS 140-2 Section		Security Level
1	Cryptographic Module Specification	1
2	Cryptographic Module Ports and Interfaces	1
3	Roles, Services and Authentication	1
4	Finite State Model	1
5	Physical Security	N/A
6	Operational Environment	1
7	Cryptographic Key Management	1
8	EMI/EMC	1
9	Self Tests	1
10	Design Assurance	1
11	Mitigation of Other Attacks	N/A

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

3. Cryptographic Module Definition

The IBM Crypto for C version 8.4.1.0 (ICC) cryptographic module is implemented in the C programming language. It is packaged as a dynamic (shared) library 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 software provided to the customer consists of:

- **ICC static stub**: static library (object code) that is linked into the customer's application and communicates with the Crypto Module. It includes the C headers (source code) containing the API prototypes and other definitions needed for linking the static library. This static library is not part of the cryptographic module.
- **ICC shared library** (libicclib84.dll for Windows, libicclib084.so for the rest): shared library (executable code) containing proprietary code needed to meet FIPS and functional requirements not provided by OpenSSL (e.g., TRNG, PRNG, self-tests, startup/shutdown), the OpenSSL cryptographic library and the zlib used for NRBG entropy estimation. This shared library constitutes the cryptographic module.
- **ICCSIG.txt file**: contains the signature file used for integrity tests. This file is not part of the cryptographic module (that is, it is not within the logical boundary).

There is a different software package for each of the target platforms and also for 32-bit and 64-bit variants.

The cryptographic module takes advantage of the hardware cryptographic accelerator features supported by the testing platforms that are part of the operational environment, as shown in the following table:

Processor	Processor Algorithm Acceleration (PAA) functions utilized by the module	Algorithms affected in the Cryptographic Module
Intel E5 2697v2	AES-NI	AES
IBM POWER 8	vcipher, vshasigma	AES, SHA-2
SPARC T4	T4 cryptographic instructions	SHA-1, SHA-2, AES, RSA, Prime Curve ECC
zSeries	Central Processor Assist for Cryptographic Functions (CPACF)	AES, SHA-1, SHA-2

The following table presents the variants of the cryptographic module that were tested and validated with their corresponding hardware and software platforms in the operational environment:

Hardware platform	Operating system	ICC variants	
		32-bits	64-bits
S2600CP, Intel E5 2697v2	Microsoft Windows Server 2008® 64-bit (with and without hardware support)	✓	✓
IBM 8286-42A POWER 8	AIX® 7.1 (with and without hardware support)	✓	✓
Netra SPARC T4-1 Server	Solaris 11 64-bit (with and without hardware support)	✓	✓
S2600CP, Intel E5 2697v2	Red Hat Linux Enterprise Server 7.0 64-bit (with and without hardware support)	✓	✓
IBM 8247-22L, POWER 8	Ubuntu 14.04 LE (with and without hardware support)		✓
IBM 8286-42A POWER 8	Red Hat Linux Enterprise Server 7.0 BE 64-bit (with and without hardware support)	✓	✓
IBM zSeries z196 type 2817 model M32	SLES 11 64-bit (with and without hardware support)	✓	✓

Table 1 - Target platforms

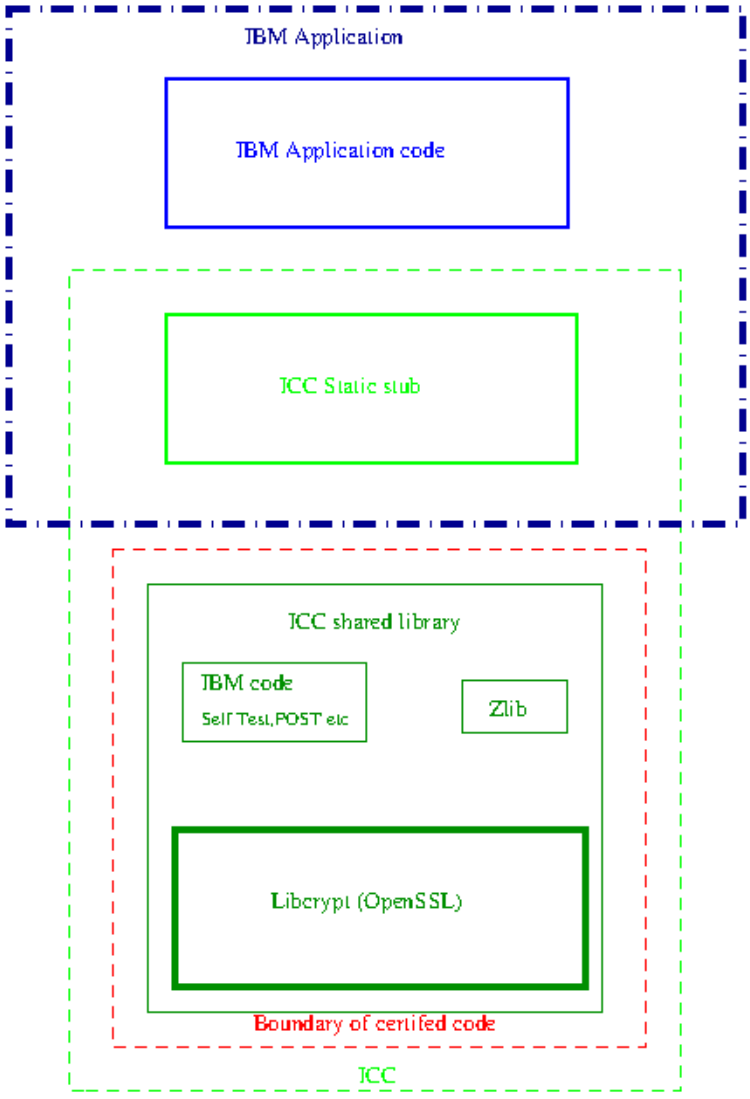
As outlined in G.5 of the Implementation Guidance for FIPS 140-2 (December 21, 2012 Update), the module maintains its compliance on other operating systems (Windows, AIX®, Solaris® and Linux), provided:

- The operating system meets the operational environment requirements at the module's level of validation, and runs in a single-user mode.
- The module does not require modification to run in the new environment.

CMVP makes no statement as to the correct operation of the module or the security strengths of the generated keys when so ported if the specific operational environment is not listed on the validation certificate.

3.1 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 binds the API functions with the shared library containing the cryptographic functionality.



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 into the calling application to bind the API with the implementation of the cryptographic services in the shared library.
- **ICC shared library** - This contains proprietary code needed to meet FIPS requirements and cryptographic services not provided by OpenSSL, a statically linked copy of zlib used for TRNG entropy estimation, and a statically linked copy of the OpenSSL cryptographic library.

The logical boundary of the cryptographic module consists of the ICC shared library bounded by the dashed red line in the figure. The signature used for the integrity check of the ICC during its initialization is contained in the file ICCSIG.txt (not shown in the figure). This file is not considered within the logical boundary.

The physical boundary of the cryptographic module is defined to be the enclosure of the computer that runs the ICC software.

4. FIPS 140-2 Specifications

4.1 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 by the API function ICC_Init (used to initiate the context handle of the module), the API function ICC_Attach (used to bind the entry point of the API functions with their implementation in the shared library), the API functions used to control the operation of the module, and configuration parameters and environment variables used to provide alternative values before the module has been initialized.
- Status Output Interface is defined as the API function ICC_GetStatus that provides information about the status of the module. The function may be called once the context of the module has been obtained.

4.2 Roles, Services and Authentication

4.2.1 Roles and Authentication

The ICC assumes 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. The roles are implicitly assumed by the services that are requested as follows:

1. **Crypto Officer -** any entity that can install and initialize the Module.
2. **User** - any entity that can access services implemented in the Module as listed in Table 2 and 4.

4.2.2 Authorized Services

An operator is implicitly assumed in the User or Cryptographic Officer role based upon the operations chosen. If an operator installs and/or initializes the Module, then the operator is in the Cryptographic Officer role. Otherwise, the operator is in the User role.

The following table shows the services and algorithms allowed in FIPS mode of operation. Requesting these services will implicitly put the module in FIPS mode of operation.

Service	Key size / Modes / Standards	CAVP	Cryptographic Keys, CSPs and access	
Symmetric Algorithms				
AES encryption & decryption	128, 192,or 256-bit keys CBC,ECB, CFB1, CFB8, CFB128, OFB modes [FIPS197]	Yes	AES Symmetric keys	R/W
Triple-DES encryption & decryption	192-bit (of which 168 bits are key bits and the rest are parity bits) keys CBC, ECB, CFB64, OFB modes [SP800-67]	Yes	Triple-DES Symmetric key	R/W
AES_XTS encryption & decryption	128 or 256 bit keys [FIPS197], [SP800-38E]	Yes	AES_XTS key	W
Public Key Algorithms				
DSA Signature Verification	L=1024, N=160 L=2048, N=224 L=2048, N=256 L=3072, N=256 [FIPS186-4]	Yes	DSA public key	R
ECDSA KeyPair Generation	P: 224, 256, 384, 521 K: 233, 283, 409, 571 B: 233, 283, 409, 571 [FIPS186-4]	Yes	ECDSA public and private key	W
ECDSA PKV	P: 192, 224, 256, 384, 521 K: 163, 233, 283, 409, 571 B: 163, 233, 283, 409, 571 [FIPS 186-4], [SP800-56A]	Yes	ECDSA key material	W

Service	Key size / Modes / Standards	CAVP	Cryptographic Keys, CSPs and access	
ECDSA Signature Generation	P: 224, 256, 384, 521 K: 233, 283, 409, 571 B: 233, 283, 409, 571 [FIPS186-4], [SP800-56A]	Yes	ECDSA private key	R
ECDSA Signature Verification	P: 192, 224, 256, 384, 521 K: 163, 233, 283, 409, 571 B: 163, 233, 283, 409, 571 [FIPS186-4], [SP800-56A]	Yes	ECDSA public key	R
RSA Key Generation	ANSI X9.31 2048 and 3072 bits [FIPS186-4]	Yes	RSA public and private key	W
RSA Signature Generation	PKCS#1.5 2048 and 3072 bits SHA-224,SHA-256,SHA-384,SHA-512 [FIPS186-4]	Yes	RSA private key	R
RSA Signature Verification	PKCS#1.5 1024, 2048 and 3072 bits SHA-1,SHA-224,SHA-256,SHA-384,SHA-512 [FIPS186-4]	Yes	RSA public key	R
Key Wrapping, Key Agreement and Key Establishment				
RSA Key encryption/ decryption	2048 and 3072 bits Allowed to be used in FIPS mode [SP800-56B]	No	RSA public and private key	R
Diffie-Hellman (DH) Key agreement and establishment	2048 to 4096 bits modulus Allowed to be used in FIPS mode [SP800-56A]	No	DH public and private key	R/W
EC Diffie-Hellman (ECDH) Key agreement and establishment	P: 224, 256, 384, 521 K: 233, 283, 409, 571 B: 233, 283, 409, 571 [SP800-56A]	Yes	ECDH public and private key	R/W
Hash Functions				
SHA-1 message digest generation	Not valid for signature generation [FIPS180-4]	Yes	None	N/A

Service	Key size / Modes / Standards	CAVP	Cryptographic Keys, CSPs and access	
SHA-224, SHA-256, SHA-384, SHA-512 message digest generation	SHA-2 algorithms [FIPS180-4]	Yes	None	N/A
Message Authentication Codes (MACs)				
HMAC-SHA message authentication code	HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512 [FIPS198-1] at least 112 bits	Yes	HMAC key	W
AES-CMAC message authentication code	128, 192 or 256 bit keys [FIPS197]	Yes	CMAC-AES keys	W
Triple-DES CMAC message authentication code	192-bit keys [FIPS197]	Yes	CMAC-Triple-DES key	W
AES_CCM	128, 192,or 256 bit keys [FIPS197], [SP800-38C]	Yes	AES_CCM key	W
AES_GCM	128, 192,or 256 bit keys [FIPS197], [SP800-38D] (compliant to section 8.2.1 for IV generation)	Yes	AES_GCM key	W
Random Number Generation				
DRBG 800-90A	HMAC_DRBG (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512) HASH_DRBG (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512) CTR_DRBG (AES-128-ECB, AES-192-ECB, AES-256-ECB) [SP800-90A]	Yes	Seed	W
Other services				
Get Status	N/A	N/A	ICC_GetStatus() API function	

Service	Key size / Modes / Standards	CAVP	Cryptographic Keys, CSPs and access	
On demand self-tests	N/A	N/A	ICC_SelfTest() API function	
Zeroization	N/A	N/A	All CSPs	

Table 2 - Services and Access in FIPS mode

Algorithms with a "Yes" in the CAVP column indicate that the algorithm has been validated through the CAVP. The table below shows the corresponding certificate numbers:

Algorithm	CAVP certificate numbers
AES	#3226, #3227, #3228, #3229, #3230, #3231, #3232, #3233, #3235, #3236, #3237, #3238, #3239, #3240, #3241, #3242, #3243, #3244, #3245, #3246, #3247, #3248, #3249, #3250, #3251, #3252
Triple-DES	#1832, #1833, #1834, #1835, #1836, #1837, #1838, #1839, #1840, #1841, #1842, #1843, #1844
DSA	#919, #920, #921, #922, #923, #924, #925, #926, #927, #928, #929, #930, #931
RSA	#1640, #1641, #1642, #1643, #1645, #1646, #1647, #1648, #1649, #1650, #1651, #1652, #1653, #1654, #1655
ECDSA	#596, #597, #598, #599, #600, #601, #602, #603, #604, #605, #606, #607, #608, #609, #610
SHA	#2666, #2667, #2668, #2669, #2670, #2671, #2672, #2673, #2675, #2676, #2677, #2678, #2679, #2680, #2681, #2682, #2683, #2684, #2685, #2686, #2687, #2688
DRBG	#687, #688, #689, #690, #691, #692, #693, #694, #696, #697, #698, #699, #700, #701, #702, #703, #704, #705, #706, #707, #708, #709, #710, #711, #712, #713
HMAC	#2030, #2031, #2032, #2033, #2034, #2035, #2036, #2037, #2038, #2039, #2040, #2041, #2042, #2043, #2044, #2045, #2046, #2047, #2048, #2049, #2050, #2051
EC Diffie-Hellman (ECDH) Primitive CVL	#441, #442, #443, #444, #445, #446, #447, #448, #449, #450, #451, #452, #453

Table 3 - CAVP certificate numbers

The table below shows the non-approved algorithms; requesting these services will implicitly put the module in non-FIPS mode of operation.

Algorithm	Notes
DSA Key/Parameter Generation	L=512, N=160; L=1024, N=160; L=2048, N=224; L=2048, N=256; L=3072, N=256 (Algorithm was not tested under the CAVP)
DSA Signature Generation	L=512, N=160; L=1024, N=160; L=2048, N=224; L=2048, N=256; L=3072, N=256 (Algorithm was not tested under the CAVP)
DSA Signature Verification	L=512, N=160 (Algorithm was not tested under the CAVP)
ECDSA KeyPair Generation	P-192, K-163, B-163 (Key sizes do not meet [SP800-131A])
ECDSA Signature Generation	P-192, K-163, B-163 (Key sizes do not meet [SP800-131A])
RSA Key Generation	Key size with 1024 bits, greater or equal than 4096 bits (Key sizes do not meet [SP800-131A])
RSA Signature Generation	Key size with 1024 bits, greater or equal than 4096 bits (Key sizes do not meet [SP800-131A])
RSA Key Wrapping	Key size with 1024 bits, greater or equal than 4096 bits (Key sizes do not meet [SP800-131A])
Diffie-Hellman (DH)	1024 bit modulus (Key sizes do not meet [SP800-131A])
EC Diffie-Hellman (ECDH)	P-192, K-163, B-163 (Key sizes do not meet [SP800-131A])
DES encryption/decryption	Non-approved algorithm
CAST encryption/decryption	Non-approved algorithm
Camellia encryption/decryption	Non-approved algorithm
Blowfish encryption/decryption	Non-approved algorithm
RC4 encryption/decryption	Non-approved algorithm
RC2 encryption/decryption	Non-approved algorithm
MD2 encryption/decryption	Non-approved algorithm

Algorithm	Notes
MD4 encryption/decryption	Non-approved algorithm
MD5 encryption/decryption	Non-approved algorithm
Password Based Encryption	Non-approved algorithm
HMAC-MD5 message authentication code	Non-approved algorithm
MDC2 message digest	Non-approved algorithm
RIPEMD message digest	Non-approved algorithm
Key Derivation Function SP800-108 KDF	Algorithm was not tested under the CAVP
TRNG	Supplies seed to the DRBG.

Table 4- non-approved algorithms in FIPS mode

4.2.3 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.

4.2.4 Operational Rules and Assumptions

The following operational rules must be followed by any user of the cryptographic 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. In case the module's power is lost and then restored, the keys used for the AES GCM encryption/decryption shall be re-distributed. The GCM is used in the context of TLS version 1.2 or higher. The mechanism for IV generation is compliant with RFC 5288.
5. The AES algorithm in XTS mode can be only used for the cryptographic protection of data on storage devices, as specified in [SP800-38E].
6. 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.
7. 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 the "Operational" 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 services are available, and all of data input and data output except the status information are inhibited.

4.3 Operational Environment

Along with the conditions stated in section 4.2.4 (“Operational Rules and Assumptions”), the criteria below must be followed in order to achieve and maintain a FIPS 140-2 mode of operation:

4.3.1 Assumptions

None.

4.3.2 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. Before the module initialization, the user has a choice to configure the TRNG alternatives and the DRBG algorithm to use. This can be set using global setting ICC_TRNG' and 'ICC_RANDOM_GENERATOR' respectively.
4. The module is initialized automatically and power-up self-tests (POST) are executed by the module when the shared library is loaded in the calling application process space. The calling application must include the following calling sequence to have access to the cryptographic services::
   - ***ICC_Init()*** creates the crypto module context.
   - ***ICC_Attach()*** binds the cryptographic functions with the API entry points.

4.4 Cryptographic Key Management

4.4.1 Implemented Algorithms

The IBM Crypto for C (ICC) version 8.4.1.0 supports the algorithms (and modes, as applicable) listed in section 4.2.2.

4.4.2 Key Generation

Key generation has dependency on random number generator DRBG 800-90A, which is detailed below. DRBG 800-90A is used to generate RSA/DSA/ECDSA/DH/ECDH key pairs as well as AES keys and Triple-DES keys. Key sizes for AES keys can be 128-bit, 192-bit or 256-bit. Key size for Triple-DES key is 192 bits long of which 168 bits are key bits and the rest are the parity bits.

In FIPS mode, RSA key generation is carried out in accordance with the algorithms described in ANSI X9.31.

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

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.

DRBG 800-90A Random Number Generator

The DRBG service is compliant with SP800-90A. The default algorithm is Hash_DRBG using SHA-256, but another algorithm from the Hash_DRBG, HMAC_DRBG and CTR_DRBG algorithms (see Table 2 for the complete list) can be also configured (see section 4.3.2). .

ICC allows for multiple entropy sources to instantiate and reseed the DRBG's, software derived, and where available, hardware RBG's. Entropy processing is as in draft SP800-90B, SP800-90C. The DRBG uses a True Random Number Generator (TRNG) to establish the initial state of the DRBG and to reseed the engine after a certain amount of time.

The TRNG's all extract noise from some source assumed to provide entropy, test to guarantee that entropy level of this noise source is at least 0.5 bits/bit, then HMAC compress and retest to guarantee that the output is better than 0.5 bits/bit. The minimum guaranteed entropy of the raw entropy source (i.e. 1 bit from a timer sample) is guaranteed to be least 0.5 bits per bit before and after HMAC compression.

In addition to the default TRNG, ICC offers multiple TRNG designs all providing the same 0.5 bits/bit entropy guarantee.

The DRBG seed and nonce are of the same length (440 bits each for HMAC-SHA256) and obtained from separate and independent calls to the TRNG. Since the DRBG is internalized by 440 bit of entropy data ((440+440)*0.5 = 440), the DRBG supports 256 bits of effective security strength in its output.

4.4.3 Key Establishment

The ICC uses in FIPS mode of operation the following key establishment methodologies:

- Diffie-Hellman (DH) with 2048-4096 bit keys providing 112-150 bits of security strength.
- Elliptic Curve Diffie-Hellman (ECDH) with 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) providing 112-256

bits of security strength.

- RSA Encrypt/Decrypt for Key Wrapping with 2048 or 3072 bit keys providing 112 or 128 bits of security strength, respectively.

4.4.4 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.

4.4.5 Key Storage

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

The only exception is the RSA public key used for integrity test, which is stored in the crypto module and relies on the operating system for protection.

4.4.6 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 services are performed via the following API functions:

Key Zeroization Services	API functions
Clean up memory locations used by low-level arithmetic functions	ICC_BN_clear_free() ICC_BN_CTX_free()
Clean up symmetric cipher context	ICC_EVP_CIPHER_CTX_free()
Clean up RSA context	ICC_RSA_free()
Clean up DSA context	ICC_DSA_free()
Clean up Diffie-Hellman context	ICC_DH_free()
Clean up asymmetric key contexts	ICC_EVP_PKEY_free()
Clean up HMAC context	ICC_HMAC_CTX_free()
Clean up ECDSA and ECDH contexts	ICC_EC_KEY_free()
Clean up CMAC context	ICC_CMAC_CTX_free()
Clean up AES-GCM context	ICC_AES_GCM_CTX_free()
Clean up RNG context	ICC_RNG_CTX_free()

Table 5 - Key Zeroization

It is the calling application's responsibility to appropriately utilize the provided zeroization methods (i.e. API functions) as listed in the table above to clean up

involved cryptographic contexts before they are released.

4.5 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.

Self-tests are automatically invoked by the module during power-up from the default entry poing (DEP) of the shared library. The self-test can also 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).

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. After the power-up tests are successfully completed, the module turns to FIPS mode of operation. Requesting any services from Table 4 will implicitly put the module in the non-FIPS mode of operation.

4.5.1 Show Status

The status of the ICC module can be obtained with the following API function:

- ICC_GetStatus: shows the state of the ICC module

The function can be called anytime after the context of the module is obtained with the ICC_Init API function.

4.5.2 Startup Tests

The module performs self-tests automatically when it is loaded. Self-tests can also be requested on demand through the API function ICC_SelfTest.

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:** 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 shared library and relies on the operating system for protection.
- **Cryptographic algorithm tests:**

  Known Answer Tests for encryption and decryption are performed for the following FIPS approved and allowed algorithms:

  - Triple-DES – CBC
  - AES 256 – CBC
  - AES_GCM
  - AES_CCM

- AES_XTS

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

Known Answer Tests for signature generation and verification are performed on the following algorithms:

- RSA signature generation with 2048 modulus
- RSA signature verification with 2048 modulus
- DSA signature verification with 2048 modulus
- ECDSA pairwise consistency test with P-384
- ECDSA signature verification with P-384
- ECDSA signature verification with B-233
- ECDSA signature verification with K-233

Other Known Answer Tests:

- DRBG 800-90A with Hash,HMAC,CTR
- RSA encryption with 2048 modulus
- RSA decryption with 2048 modulus
- ECC primitive Z computation KAT

4.5.3 Conditional Tests

Pairwise consistency tests 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:

- ECDSA
- RSA

Continuous RNG tests: the module implements Continuous RNG tests as follows:

DRBG 800-90A

- The DRBG 800-90A generates a minimum of 8 bytes per request.

If less than 8 bytes are requested, the rest of the bytes is discarded and the next request will generate new random data.

- 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 800-90A, two internal 8 byte cycles are performed.
- The DRBG 800-90A performs known answer tests when first instantiated and health checks at intervals as specified in the standard.

True Random Number Generator (TRNG)

- A non-deterministic RNG (NDRNG) is used to seed the RNG. Every time a new seed or n bytes is required (either to initialize the RNG, reseed the RNG periodically or reseed the RNG by user's demand), the cryptographic module performs a comparison between the SHA-256 message digest using the new seed and the previously calculated digest. If the values match, the TRNG generates a new stream of bytes until the continuous RNG test passes.

4.5.4 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 4.5.1).

4.6 Mitigation of Other Attacks

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

5. API Functions

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

- ICC_EC_GROUP_set_asn1_flag
- ICC_EVP_CIPHER_CTX_flags
- ICC_EVP_CIPHER_CTX_set_flags
- ICC_GetStatus
- ICC_Init
- ICC_Attach
- 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_AES_GCM_GenerateIV_NIST
- 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
- ICC_lib_cleanup
- ICC_MemCheck_start
- ICC_MemCheck_stop
- ICC_EC_GROUP_set_asn1_flag
- ICC_OPENSSL_cpuid_override
- ICC_OPENSSL_cpuid
- ICC_EVP_CIPHER_CTX_flags
- ICC_EVP_CIPHER_CTX_set_flags
- ICC_OPENSSL_HW_rand
- ICC_OPENSSL_rdtscX
- ICC_EVP_CIPHER_CTX_copy
- ICC_BN_is_prime_fasttest_ex