IBM® Crypto for C
version 8.2.2.0

FIPS 140-2 Non-Proprietary
Security Policy, version 1.7

July 24, 2013

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
SP800-131A	NIST	Special Publication 800-131A: Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths
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

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-NI	Intel® Advanced Encryption Standard (AES) New Instructions (AES-NI)
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
CPACF	CP (central processor) assist for Cryptographic Functions
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.

2 Introduction

This document is a non-proprietary FIPS 140-2 Security Policy for the IBM Crypto for C (ICC), version 8.2.2.0 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/groups/STM/cmvp/.
- For more information about IBM software please visit http://www.ibm.com

2.1 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.

2.2 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

3. 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 version 8.2.2.0 (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 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.

The cryptographic module (specifically OpenSSL) takes advantage of the AES-NI and CPACF features supported by some of the testing platforms that are part of the operational environment:

- Advanced Encryption Standard (AES) New Instructions (AES-NI) is an extension to the x86 instruction set architecture for microprocessors from Intel and AMD proposed by Intel in March 2008. The purpose of the instruction set is to improve the speed of applications performing encryption and decryption using the Advanced Encryption Standard (AES). The new AES-NI instruction set is comprised of six new instructions that perform several compute intensive parts of the AES algorithm. These instructions can execute using significantly less clock

cycles than a software solution. Four of the new instructions are for accelerating the encryption/decryption of a round and two new instructions are for round key generation.

- Central Processor Assist for Cryptographic Functions (CPACF) is a set of cryptographic instructions on the z196 processors. These instructions provide for high speed cryptography. The CPACF Data Encryption Standard (DES) / Triple Data Encryption Standard (Triple-DES) enablement feature (feature 3863) provides clear keys for DES and Triple-DES instructions. This feature provides:
    - DES 64-bit keys (including 8 parity bits)
    - Triple-DES 192-bit keys (including 24 parity bits)
    - Advance Encryption Standard (AES) for 128, 192 and 256 bit keys.
    - SHA1, SHA-224, SHA-256, SHA-384 and SHA-512.
    - Pseudo Random Number Generation (PRNG).

  OpenSSL 1.0.1, which is part of the cryptographic module, implements hardware acceleration for AES and SHA only. Triple-DES and PRNG are not supported.

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
ALTECH SH67H3 Intel® Core™ i7-2600	Microsoft Windows Server 2008® 64-bit (with and without AES-NI)	✓	✓
IBM 8835 52X AMD Opteron 246	Microsoft Windows Server 2008® 32-bit	✓	
IBM RS6000 7037-A50 PowerPC 5 64	AIX® 6.1 64-bit	✓	✓
Sun Fire T1000 UltraSPARC T1	Solaris® 10 64-bit	✓	✓
ALTECH SH67H3 Intel® Core™ i7-2600	Red Hat Linux Enterprise Server 5 64-bit (with and without AES-NI)	✓	✓
IBM 8835 52X AMD Opteron 246	Red Hat Linux Enterprise Server 5 32-bit	✓	
IBM System p5 185 7037-A50 IBM PowerPC 970	Red Hat Linux Enterprise Server 5 64-bit	✓	✓
IBM z196 type 2817 model M32 zSeries 196	Red Hat Linux Enterprise Server 5 64-bit (with and without CPACF)	✓	✓

Table 1 - Target platforms

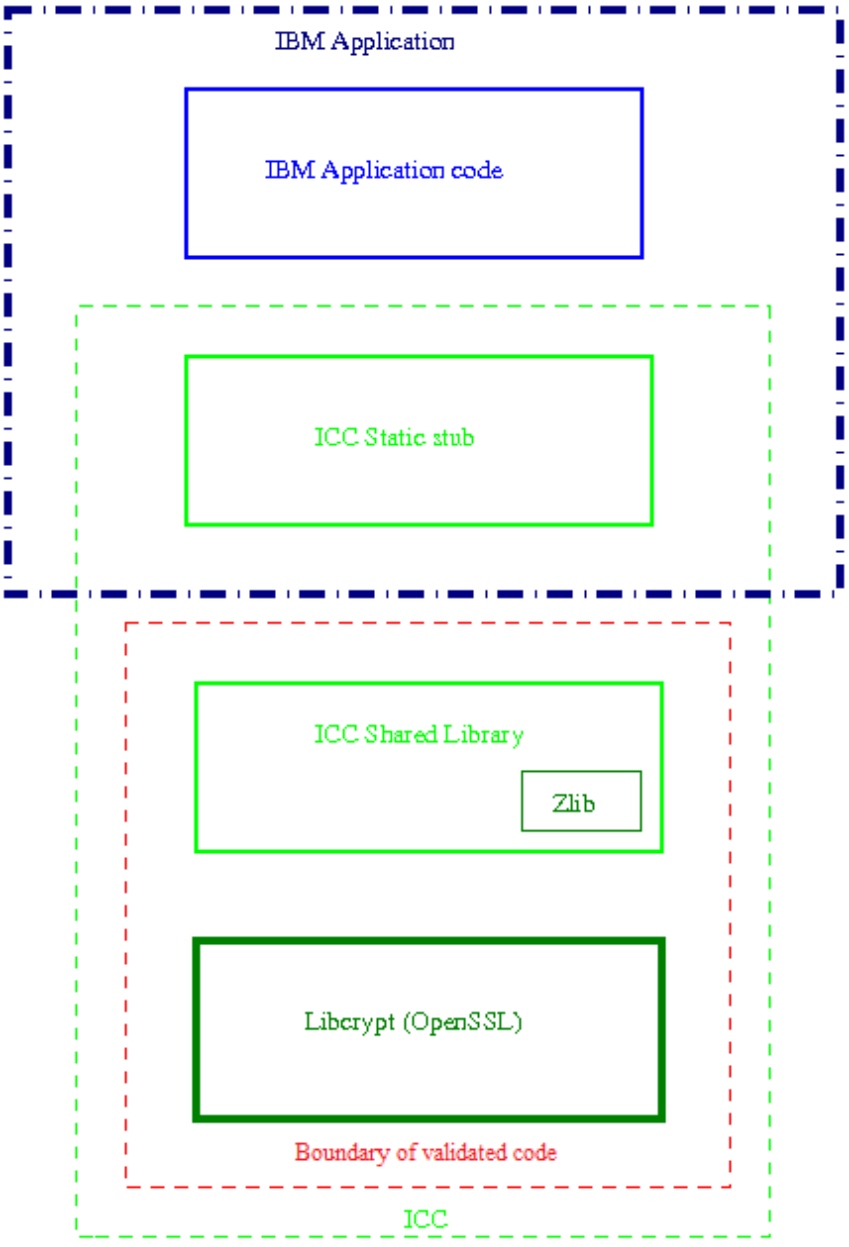
As outlined in G.5 of the Implementation Guidance for FIPS 140-2 (December 21, 2012 Update), the module maintains its compliance on other operating systems (Windows, AIX®, Solaris® and Linux), provided:

- 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.
- The CMVP makes no statement as to the correct operation of the module or the

security strengths of the generated keys when ported.

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 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 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**.**

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 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.

4.2 Roles, Services and Authentication

4.2.1 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.

The table 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 2 - Roles and Services

4.2.2 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 the tables 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.

Table 3 provides a summary of the services and access supported by the ICC. Table 4 provides services supported by the ICC in non-FIPS mode of operation.

Service	Notes	Modes	IS FIPS-Approved? If yes, Cert #	Cryptographic Keys, CSPs and access	
Symmetric Algorithms					
AES encryption & decryption	128, 192,or 256-bit keys (FIPS 197) Encrypt/Decrypt (with and without AES-NI support) (with and without CPACF support)	CBC,ECB, CFB1, CFB8, CFB128, OFB	Yes Cert #2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2169, 2170, 2171, 2172, 2179, 2213, 2214	AES Symmetric key	Read/ Write

Service	Notes	Modes	IS FIPS-Approved? If yes, Cert #	Cryptographic Keys, CSPs and access	
Triple-DES encryption & decryption	192-bit (of which 168 bits are key bits and the rest are parity bits) keys (SP 800-67) Encrypt/Decrypt	CBC, ECB, CFB64, OFB	Yes Cert #1365, 1366, 1367, 1368, 1369, 1370, 1371, 1372, 1373, 1374, 1375, 1376, 1377, 1379	Triple-DES Symmetric key	Read/ Write
Public Key Algorithms					
DSA Key/Parameter Generation	L=1024-bit, N=160 (FIPS 186-2)	N/A	Yes Cert #670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 756, 757	DSA public and private key	Write
DSA Key/Parameter Generation	L=512, N=160 L=2048, N=224 L=2048, N=256 L=3072, N=256 (FIPS 186-3) Available only in non-FIPS mode	N/A	No	DSA public and private key	Write
DSA Signature Generation	L=1024, N=160 (FIPS 186-2)	N/A	Yes Cert #670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 756, 757	DSA private key	Read

Service	Notes	Modes	IS FIPS-Approved? If yes, Cert #	Cryptographic Keys, CSPs and access	
DSA Signature Generation	L=512, N=160 L=2048, N=224 L=2048, N=256 L=3072, N=256 (FIPS 186-3) Available only in non-FIPS mode	N/A	No	DSA private key	Read
DSA Signature Verification	L=1024, N=160 (FIPS 186-2)	N/A	Yes Cert #670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 756, 757	DSA public key	Read
DSA Signature Verification	L=512, N=160 L=2048, N=224 L=2048, N=256 L=3072, N=256 (FIPS 186-3) Available only in non-FIPS mode	N/A	No	DSA public key	Read
ECDSA KeyPair	P: 192, 224, 256, 384, 521 K: 163, 233, 283, 409, 571 B: 163, 233, 283, 409, 571	N/A	Yes Cert #325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 398, 399	ECDSA public and private key	Write

Service	Notes	Modes	IS FIPS-Approved? If yes, Cert #	Cryptographic Keys, CSPs and access	
ECDSA PKV	P: 192, 224, 256, 384, 521 K: 163, 233, 283, 409, 571 B: 163, 233, 283, 409, 571	N/A	Yes Cert #325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 398, 399	ECDSA key material	Write
ECDSA Signature Generation	P: 192, 224, 256, 384, 521 K: 163, 233, 283, 409, 571 B: 163, 233, 283, 409, 571	N/A	Yes Cert #325, 326, 327,328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 398, 399	ECDSA private key	Read
ECDSA Signature Verification	P: 192, 224, 256, 384, 521 K: 163, 233, 283, 409, 571 B: 163, 233, 283, 409, 571	N/A	Yes Cert #325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 398, 399	ECDSA public key	Read
RSA Key Generation	ANSI X9.31 (1024 to 4096 bits)	N/A	Yes Cert #1109, 1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, 1118, 1119, 1120, 1121, 1123, 1253, 1254	RSA public and private key	Write

Service	Notes	Modes	IS FIPS-Approved? If yes, Cert #	Cryptographic Keys, CSPs and access	
RSA Signature Generation	PKCS#1.5 (1024 to 4096 bits) (SHA-1,SHA-224,SHA-256,SHA-384,SHA-512) (with and without CPACF support)	N/A	Yes Cert #1109, 1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, 1118, 1119, 1120, 1121, 1123, 1253, 1254	RSA private key	Read
RSA Signature Verification	PKCS#1.5 (1024 to 4096 bits) (SHA-1,SHA-224,SHA-256,SHA-384,SHA-512) (with and without CPACF support)	N/A	Yes Cert #1109, 1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, 1118, 1119, 1120, 1121, 1123, 1253, 1254	RSA public key	Read
RSA Key Wrapping	Encrypt / Decrypt (1024 to 4096 bits) Allowed to be used in FIPS mode	N/A	No	RSA public and private key	Read
Diffie-Hellman (DH)	1024 to 4096 bits modulus Allowed to be used in FIPS mode	Key agreement and Key Generation	No	DH public and private key	Read/ Write

Service	Notes	Modes	IS FIPS-Approved? If yes, Cert #	Cryptographic Keys, CSPs and access	
EC Diffie-Hellman (ECDH)	P: 192, 224, 256, 384, 521 K: 163, 233, 283, 409, 571 B: 163, 233, 283, 409, 571 (SP 800-56A) Allowed to be used in FIPS mode	Key agreement and Key Generation	No	ECDH public and private key	Read/ Write
Hash Functions					
SHA-1 message digest generation	FIPS 180-4	N/A	Yes Cert #1874, 1875, 1876, 1877, 1878, 1879, 1880, 1881, 1882, 1883, 1884, 1885, 1886, 1889, 1904, 1905	None	N/A
SHA-224, SHA-256, SHA-384, SHA-512 message digest generation	FIPS 180-4, SHA-2 algorithms (with and without CPACF support)	N/A	Yes Cert #1874, 1875, 1876, 1877, 1878, 1879, 1880, 1881, 1882, 1883, 1884, 1885, 1886, 1889, 1904, 1905	None	N/A
Message Authentication Codes (MACs)					

Service	Notes	Modes	IS FIPS-Approved? If yes, Cert #	Cryptographic Keys, CSPs and access	
HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	FIPS 198, 198-1 (with and without CPACF support)	N/A	Yes Cert #1319, 1320, 1321, 1322, 1323, 1324, 1325, 1326, 1327, 1328, 1329, 1330, 1331, 1333, 1506, 1507	HMAC-SHA-1 key, HMAC-SHA-224 key, HMAC-SHA-256 key, HMAC-SHA-384 key, HMAC-SHA-512 key	Write
AES-128-CMAC, AES-192-CMAC, AES-256-CMAC	128, 192,or 256 bit keys (FIPS 197) Encrypt/Decrypt (with and without AES-NI support) (with and without CPACF support)	N/A	Yes Cert #2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2165, 2166, 2167, 2169, 2170, 2179, 2427, 2429, 2431, 2433, 2438, 2443	CMAC-AES-128 key, CMAC-AES-192 key, CMAC-AES-256 key	Write
Triple-DES _CMAC (CMAC with three key Triple-DES)	192-bit keys (FIPS 197)	CBC	Yes Cert #1365, 1366, 1367, 1368, 1369, 1370, 1371, 1372, 1373, 1374, 1375, 1376, 1377, 1379	CMAC-Triple-DES key (192-bit)	Write

Service	Notes	Modes	IS FIPS-Approved? If yes, Cert #	Cryptographic Keys, CSPs and access	
AES_CCM	128, 192,or 256 bit keys (SP800-38C) (with and without AES-NI support) (with and without CPACF support)	N/A	Yes Cert #2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2165, 2166, 2167, 2169, 2170, 2179, 2427, 2429, 2431, 2433, 2438, 2443	AES_CCM key	Write
AES_GCM	128, 192,or 256 bit keys (FIPS 197, SP800-38D) (with and without AES-NI support) (with and without CPACF support)	N/A	Yes Cert #2421, 2422, 2423, 2424, 2425, 2426, 2427, 2428, 2429, 2430, 2432, 2434, 2435, 2436, 2437, 2438, 2439, 2440, 2441, 2443	AES_GCM key	Write
Random Number Generation					
DRBG 800-90A	SP 800-90A (with and without AES-NI support) (with and without CPACF support)	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)	Yes Cert #240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 326, 327, 328, 329, 330, 331	Seed	Write

Table 3 - Services and Access

Service	Notes	Is 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
MD2	Hash function	No
MD4	Hash function	No
MD5	Hash function	No
HMAC-MD5	Keyed Message Authentication function	No
MDC2	Hash function	No
RIPEMD	Hash function	No
Key Derivation Function	SP800-108 KDF	The KBKDF is a FIPS-Approved algorithm, but its implementation in this module is not certified by the CAVP and hence is non-compliant to the FIPS 140-2 standard.

Table 4- Other services only available in non-FIPS mode

CAVEAT#1: [SP800-131A] describes the transition associated with the use of cryptographic algorithms and key lengths; based on the information included in this publication the usage of following algorithms implemented in this cryptographic module is discouraged as they cannot be used in FIPS mode after the transition period:

- DSA Key Generation and Digital Signature Generation with keys of length < 2048 bits, disallowed after 2013.
- RSA Key Generation and Digital Signature Generation with keys of length < 2048

bits, disallowed after 2013.

- Diffie-Hellman's Key Agreement using finite fields with 1024 bit keys, disallowed after 2013.
- Diffie-Hellman's Key Agreement using elliptic curves with keys of length < 224 bits, disallowed after 2013.
- RSA Key Wrapping with keys of length < 2048 bits, disallowed after 2013.
- SHA-1 for digital signature generation, disallowed after 2013.
- HMAC with key lengths < 112 bits, disallowed after 2013.
- Two-key Triple DES Encryption: acceptable through 2010, restricted use from 2011 through 2015, disallowed after 2015.
- Two-key Triple DES Decryption: Acceptable through 2010, legacy-use after 2010.

Users are encouraged to check on a periodic basis for new versions of this publication and verify whether updates in the transitional periods in these or others cryptographic algorithms and key sizes may affect FIPS approved services in this cryptographic module.

CAVEAT #2: In case of power loss in the cryptographic module, the keys used for the AES GCM shall be re-distributed.

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.

Note: MD5 is not a FIPS-approved algorithm, but it is allowed to be used in the FIPS mode in the context of TLS 1.0 and 1.1 as part of pseudorandom function. This module does not implement the TLS protocols. Rather it provides primitives for the calling application who may support TLS. The use of MD5 from this module in the context of TLS in the FIPS mode does not imply the TLS implementation's compliance with any applicable standards.

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 of power loss in the module, the keys used for the AES GCM shall be re-distributed.
5. 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.
6. 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.

4.3 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:

4.3.1 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

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. 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

4.4 Cryptographic Key Management

4.4.1 Implemented Algorithms

The IBM Crypto for C (ICC) version 8.2.2.0 supports the algorithms (and modes, as applicable) listed above in Table 2 in section 5.2.2.

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, 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.

DRBG 800-90A Random Number Generator

The DRBG used by the cryptographic module is implemented following SP800-90A. Several FIPS approved algorithms in SP800-90A can be chosen, but the default algorithm is HMAC-SHA256.

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 extracts entropy from time measurement jitter (minute variations of clock edges). Only one bit (that is, the least significant moving bit) is taken from each timer sample. Bits from multiple samples are concatenated to form a bitstring as entropy input to seed the DRBG. An n-bit string requires the TRNG to obtain n timer samples and then concatenate 1 bit from each timer sample. The internal TRNG engine feeds entropy on demand into the DRBG (seed and reseed). The state of the TRNG entropy estimation test is cleared on exit.

The minimum guaranteed entropy of the raw entropy source (i.e. 1 bit from a timer sample) is 0.5 bits per bit. This is ensured by the following continuous tests performed on the TRNG:

- Distribution of the raw entropy source: this uses an integer version of the min-entropy entropy assessment algorithm to guarantee >= 0.5bits/bit in each batch of data extracted from the raw source.
- Check on the output from the TRNG cores. (After any post-processing of the raw entropic source) This is a simple Chi-Squared statistical test on a relatively small number of 4 bit samples to ensure that the input distribution of incoming bits is reasonable.
- Check on the long term entropy of the generated data. This uses a compression function over a substantial data set (1k bytes) and checks that the data is less than 50% compressible.

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 the following as key establishment methodologies:

- Diffie-Hellman (DH) with 1024-4096 bit keys providing 80-150 bits of security strength.
- Elliptic Curve Diffie-Hellman (ECDH) with curves (P-192, P-224, P-256, P-384, P-521, K-163, K-233, K-283, K-409, K-571, B-163, B-233, B-283, B-409, B-571) providing 80-256 bits of security strength.
- RSA Encrypt/Decrypt for Key Wrapping with 1024-4096 bit keys providing 80-150 bits of security strength.

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.

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()

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. 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.

4.5.1 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 800-90A seed source

Both functions may be called anytime after ICC_Init.

4.5.2 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:**

  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

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 generation with 1024 modulus
- DSA signature verification with 1024 modulus
- ECDSA signature generation with P-384
- ECDSA signature verification with P-384

Other Known Answer Tests:

- DRBG 800-90A
- RSA encryption with 2048 modulus
- RSA decryption with 2048 modulus

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.

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:

- DSA
- 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 relies on the environment (i.e. proper shutdown of the shared libraries) for resistance to retrospective attacks on data.

- 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 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-1 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 5.5.1 above).

4.6 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 to 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.

4.7 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.

Functions marked with (CO) are crypto officer functions. 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

- ICC_EC_GROUP_set_asn1_flag
- ICC_EVP_CIPHER_CTX_flags
- ICC_EVP_CIPHER_CTX_set_flags
- 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 (non-FIPS mode)
- 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 (non-FIPS mode)
- ICC_SP800_108_KDF (non-FIPS mode)
- 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)

.