IBM Crypto for C
versions 8.6.0.0 and 8.6.1.0

FIPS 140-2 Non-Proprietary Security Policy

Version 2.3

July 3, 2018

This document is the property of International Business Machines Corporation. This document may only be reproduced in its entirety without modifications.

© Copyright 2018 IBM Corp. / atsec information security. All Rights Reserved

This document can be reproduced and distributed only whole and intact, including this copyright notice.

Table of Contents

This document can be reproduced and distributed only whole and intact, including this copyright notice.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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, August 2017
FIPS180-4	NIST	SECURE HASH STANDARD (SHS), August 2015
FIPS186-4	NIST	Digital Signature Standard (DSS), July 2013
FIPS197	NIST	Specification for the ADVANCED ENCRYPTION STANDARD (AES), November 2001
FIPS198-1	NIST	The Keyed Hash Message Authentication Code (HMAC), July 2008
FIPS202	NIST	SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, August 2015
SP800-38A	NIST	Recommendation for Block Cipher Modes of Operation Methods and Techniques, December 2001
SP800-38B	NIST	Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, May 2005
SP800-38C	NIST	Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality, May 2004
SP800-38D	NIST	Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, November 2007
SP800-38E	NIST	Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices, January 2010
SP800-38F	NIST	Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping, December 2012
SP800-52 Rev. 1	NIST	Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations, April 2014

This document can be reproduced and distributed only whole and intact, including this copyright notice.

Reference	Author	Title
SP800-56A Rev. 2	NIST	Recommendation for Pair Wise Key Establishment Schemes Using Discrete Logarithm Cryptography, May 2013
SP800-67 Rev. 1	NIST	Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher, January 2012
SP800-90A Rev. 1	NIST	Recommendation for Random Number Generation Using Deterministic Random Bit Generators, June 2015
SP800-131A Rev. 1	NIST	Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths, November 2015
SP800-133	NIST	Recommendation for Cryptographic Key Generation, December 2012

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 with Cipher block chaining-Message authentication code as documented in NIST SP800-38C.
AES_GCM	AES Galois counter mode as documented in NIST SP800-38D.
AES KW	AES Key Wrap mode as documented in NIST SP800-38F.
AES KWP	AES Key Wrap with Padding mode as documented in NIST SP800-38F
AES_XTS	AES XOR Encrypt XOR (XEX) tweakable block cipher 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	Cryptographic Module Validation Program; a joint effort between US NIST and Communications Security Establishment (CSE) of Canada.
CSE	Communications Security Establishment of Canada.
CPACF	CP (central processor) assist for Cryptographic Functions.
Crypto	Cryptographic capability/functionality

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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) and Transport Layer Security (TLS) 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.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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.
SHA-3	A set of hash algorithms, including SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128 and SHAKE256.
Triple-DES	Based on the DES standard; the plaintext is, in effect, encrypted three times. Triple-DES (TDEA), as specified in SP 800-67, 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.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

2 Introduction

This document is a non-proprietary FIPS 140-2 Security Policy for the IBM Crypto for C versions 8.6.0.0 and 8.6.1.0 (ICC) 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

This document can be reproduced and distributed only whole and intact, including this copyright notice.

3 Cryptographic Module Definition

The 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).

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® Xeon®	AES-NI	AES
SPARC T4	SPARC	SHA-1, SHA-2, AES

Table 1 - Processor Algorithm Acceleration (PAA) functions

The following table presents the operational environments on which versions 8.6.0.0 and 8.6.1.0 of the cryptographic module were tested and validated. Each operational environment includes the hardware platform, the processor and the operating system. Each row of the table also includes the corresponding version of the module.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

OE	Hardware platform	Processor	Operating system	ICC version
1	Dell PowerEdgeR630 XL	Intel® Xeon®	Microsoft Windows Server 2012R2® 64-bit, with and without AES-NI (PAA)	8.6.1.0
2	Dell PowerEdgeR630 XL	Intel® Xeon®	Red Hat Linux Enterprise Server 7.3 64-bit, with and without AES-NI (PAA)	8.6.0.0
3	Dell PowerEdgeR630 XL	Intel® Xeon®	IBM Modular Extensible Security Architecture (MESA), with Linux kernel 3.10, 64-bit, under VMware ESXi 6.0, with and without AES-NI (PAA)	8.6.0.0
4	Netra SPARC T4-1 Server	SPARC T4	Solaris® 11 64-bit, with and without SPARC (PAA)	8.6.0.0

Table 2 - Tested platforms

As outlined in G.5 of the Implementation Guidance for FIPS 140-2 [FIPS140-2-IG], the module maintains its compliance on other operating systems, provided that:

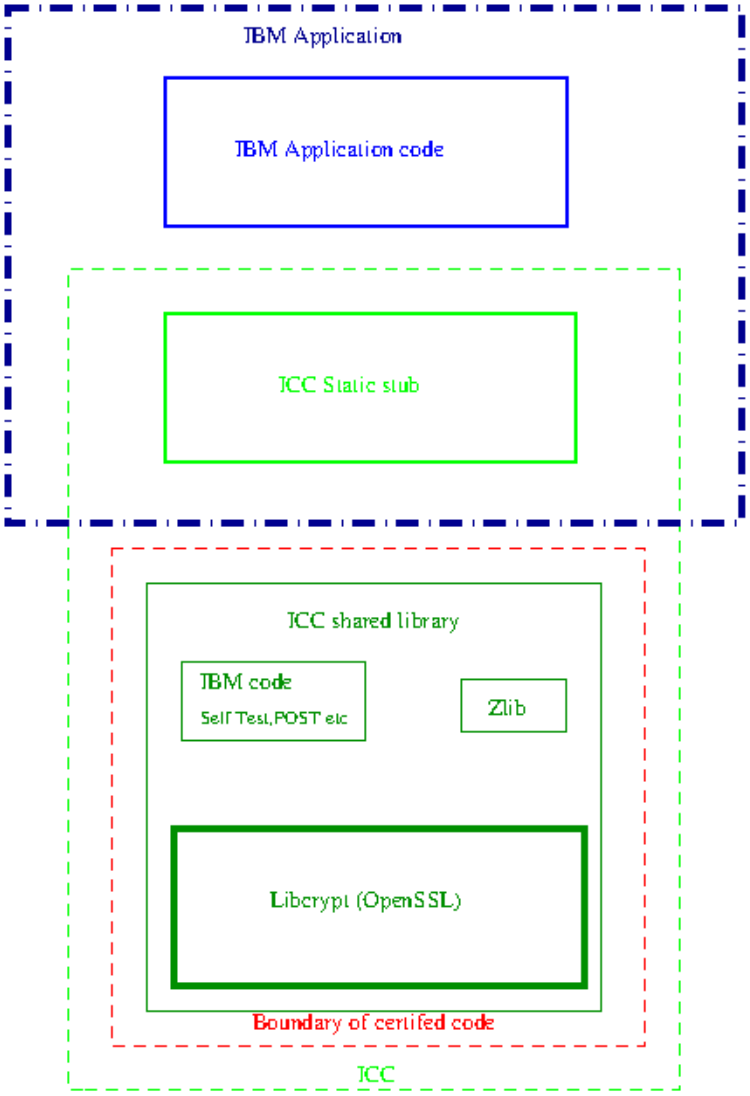
- the operating system meets the operational environment requirements at the module's level of validation, and runs in a single-user mode;
- the module does not require modification to run in the new environment.

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.

This document can be reproduced and distributed only whole and intact, including this copyright notice.



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.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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:

- **Crypto Officer -** any entity that can install and initialize the Module.
- **User** - any entity that can access services implemented in the Module as listed in Table 3 and Table 4.

4.2.2 Services and Cryptographic Algorithms

An operator is implicitly assumed in the User or Crypto-Officer role based upon the

This document can be reproduced and distributed only whole and intact, including this copyright notice.

operations chosen. If an operator installs and/or initializes the Module, then the operator is in the Crypto-Officer role. Otherwise, the operator is in the User role.

The module enters the FIPS mode of operation (the approved mode) implicitly after initialization. Once operational, the module assumes implicitly the mode of operation depending on the services requested.

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

Service	Algorithms	Role	Keys, CSPs and access	
Symmetric encryption and decryption	AES	User	AES key	R
	Triple-DES	User	Triple-DES key	R
DSA signature verification	DSA	User	DSA public key	R
ECDSA Key Pair Generation	ECDSA, DRBG	User	ECDSA public and private keys	W
ECDSA public key validation	ECDSA	User	ECDSA key material	R
ECDSA Signature Generation and Verification	ECDSA	User	ECDSA private and public keys	R
RSA Key Generation	RSA, DRBG	User	RSA public and private keys	W
RSA Signature Generation and Verification	RSA	User	RSA public and private keys	R
Key wrapping	AES	User	AES key	R
Key encapsulation	RSA	User	RSA public and private keys	R
Diffie-Hellman Key Agreement	KAS FFC	User	Diffie-Hellman domain parameters	RW
EC Diffie-Hellman Key Agreement	KAS ECC, ECC CDH primitive	User	EC Diffie-Hellman public and private keys	RW
Message digest	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	User		
	SHA3-224, SHA3-256, SHA3-384, SHA3-512	User		
Message authentication code (MAC)	HMAC with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	User	HMAC key	R
	HMAC with SHA3-224, SHA3-256, SHA3-384, SHA3-512	User	HMAC key	R
	CMAC with AES	User	AES key	R

This document can be reproduced and distributed only whole and intact, including this copyright notice.

Service	Algorithms	Role	Keys, CSPs and access	
	CMAC with Triple-DES	User	Triple-DES key	R
Random number generation	SP800-90A DRBG	User	Entropy input string, Internal state	RW
Get Status	n/a	User		
On demand self-tests	n/a	User		
Zeroization	n/a	User	All CSPs	W
Module installation and configuration	n/a	Crypto Officer		

Table 3 – Services in FIPS mode of operation

The following table shows the services and algorithms not allowed in FIPS mode of operation. Requesting these services will implicitly put the module in non-FIPS mode of operation.

In addition, requesting any of these services using a processor capability mask not allowed by this security policy, as specified in item 4 of section 4.3.2, will also put the module in non-FIPS mode of operation.

Service	Algorithms	Role	Keys, CSPs and access	
Symmetric encryption and decryption	DES, CAST, Camellia, Blowfish, RC4, RC2	User	Symmetric key	R
	2-key Triple-DES	User	Triple-DES key	R
DSA Key/Parameter Generation	DSA	User	DSA public and private keys	RW
DSA Signature Generation	DSA	User	DSA private key	R
DSA Signature Verification	DSA using keys disallowed by [SP800-131A]	User	DSA public key	R
ECDSA Key Pair Generation	ECDSA using keys disallowed by [SP800-131A]	User	ECDSA private and public keys	RW
ECDSA Signature Generation	ECDSA using keys or SHS disallowed by [SP800-131A]	User	ECDSA private key	R
RSA Key Generation	RSA using keys disallowed by [SP800-131A]	User	RSA public and private keys	RW
RSA Signature Generation and Verification	RSA using keys or SHS disallowed by [SP800-131A]	User	RSA public and private keys	R

This document can be reproduced and distributed only whole and intact, including this copyright notice.

Service	Algorithms	Role	Keys, CSPs and access	
Key encapsulation	RSA using keys disallowed by [SP800-131A]	User	RSA public and private keys	R
Diffie-Hellman Key Agreement	KAS FFC using keys disallowed by [SP800-131A]	User	Diffie-Hellman public and private keys	RW
EC Diffie-Hellman Key Agreement	KAS ECC using keys disallowed by [SP800-131A]	User	EC Diffie-Hellman public and private keys	RW
Message digest	MD2, MD4, MD5, MDC2, RIPEMD	User		
Message authentication code (MAC)	HMAC-MD5	User	HMAC key	R
Password Based Encryption		User		
Key Derivation Functions (KDF)	SP800-108	User		

Table 4 - Services in non-FIPS mode of operation

The following table shows the FIPS approved cryptographic algorithms with their associated modes, key sizes and usage in cryptographic services. Each algorithm specifies the CAVS certificate for each of the operational environments (OE) on which it was tested (please refer to Table 2 for a description of the corresponding OE id) and whether the module was set or unset to take advantage of the processor algorithm acceleration (PAA) capabilities.

CAVP Cert#	Algorithm	Standard	Mode / Method	Key size	Use
Tested on OE 1: • PAA: #4732 • no PAA: #4733 Tested on OE 2: • PAA: #4734 • no PAA: #4736 Tested on OE 3: • PAA: #4730 • no PAA: #4731	AES	[FIPS197], [SP800-38A]	ECB, CBC, OFB, CFB1, CFB8, CFB128, CTR	128, 192 and 256 bits	Data Encryption and Decryption
		[SP800-38B]	CMAC	128, 192 and 256 bits	MAC Generation and Verification
		[SP800-38C]	CCM	128, 192 and 256 bits	Data Encryption and Decryption
		[SP800-38E]	XTS	128 and 256 bits	Data Encryption and Decryption for Data Storage
		[SP800-38D]	GCM	128, 192 and 256 bits	Data Encryption and Decryption

This document can be reproduced and distributed only whole and intact, including this copyright notice.

CAVP Cert#	Algorithm	Standard	Mode / Method	Key size	Use
Tested on OE 4: • PAA: #4737 • no PAA: #4738		[SP800-38F]	KW	128, 192 and 256 bits	Key Wrapping and Unwrapping
Tested on OE 1: • PAA: #2514 • no PAA: #2515 Tested on OE 2: • PAA: #2516 • no PAA: #2517 Tested on OE 3: • PAA: #2512 • no PAA: #2513 Tested on OE 4: • PAA: #2518 • no PAA: #2519	Triple-DES	[SP800-67] [SP800-38A]	CBC, ECB, CFB64, OFB	192 bits (168 bits are key bits and the rest are parity bits)	Data Encryption and Decryption
		[SP800-67] [SP800-38B]	CMAC	192 bits (168 bits are key bits and the rest are parity bits)	MAC Generation and Verification
Tested on OE 1: • PAA: #1264 • no PAA: #1265 Tested on OE 2: • PAA: #1266 • no PAA: #1268 Tested on OE 3: • PAA: #1262 • no PAA: #1263 Tested on OE 4: • PAA: #1269 • no PAA: #1270	DSA	[FIPS186-4]	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	L=1024, N=160 L=2048, N=224 L=2048, N=256 L=3072, N=256	Signature Verification
Tested on OE 1:	ECDSA	[FIPS186-4]		P-224, P-256, P-384, P-521	Key Pair Generation

This document can be reproduced and distributed only whole and intact, including this copyright notice.

CAVP Cert#	Algorithm	Standard	Mode / Method	Key size	Use
• PAA: #1175 • no PAA: #1176 Tested on OE 2: • PAA: #1177 • no PAA: #1178 Tested on OE 3: • PAA: #1173 • no PAA: #1174 Tested on OE 4: • PAA: #1179 • no PAA: #1180		[SP800-56A]	SHA-224, SHA-256, SHA-384, SHA-512	K-233, K-283, K-409, K-571 B-233, B-283, B-409, B-571	Signature Generation
				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	Public Key Verification
			SHA-1, SHA-224, SHA-256, SHA-384, SHA-512		Signature Verification
Tested on OE 1: • PAA: #2581 • no PAA: #2582 Tested on OE 2: • PAA: #2583 • no PAA: #2585 Tested on OE 3: • PAA: #2579 • no PAA: #2580 Tested on OE 4: • PAA: #2586 • no PAA: #2587	RSA	[FIPS186-4]	**X9.31**	2048 and 3072 bits	Key Pair Generation
			PKCS#1v1.5 SHA-224, SHA-256, SHA-384, SHA-512	2048 and 3072 bits	Digital Signature Generation
			PKCS#1v1.5 SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	1024, 2048 and 3072 bits	Digital Signature Verification
			PSS SHA-224, SHA-256, SHA-384, SHA-512	2048 and 3072 bits	Digital Signature Generation
			PSS SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	1024, 2048 and 3072 bits	Digital Signature Verification

This document can be reproduced and distributed only whole and intact, including this copyright notice.

CAVP Cert#	Algorithm	Standard	Mode / Method	Key size	Use
Vendor Affirmed	RSA	[FIPS186-4]	**PKCS#1v1.5** SHA3-224, SHA3-256, SHA3-384, SHA3-512	2048 and 3072 bits	Digital Signature Generation
			PKCS#1v1.5 SHA3-224, SHA3-256, SHA3-384, SHA3-512	1024, 2048 and 3072 bits	Digital Signature Verification
			PSS SHA3-224, SHA3-256, SHA3-384, SHA3-512	2048 and 3072 bits	Digital Signature Generation
			PSS SHA3-224, SHA3-256, SHA3-384, SHA3-512	1024, 2048 and 3072 bits	Digital Signature Verification
Tested on OE 1: • PAA: CVL#1371 • no PAA: CVL#1372 Tested on OE 2: • PAA: CVL#1373 • no PAA: CVL#1377 Tested on OE 3: • PAA: CVL#1369 • no PAA: CVL#1370 Tested on OE 4: • PAA: CVL#1375 • no PAA: CVL#1376	ECC CDH primitive	[SP800-56A]		P-224, P-256, P-384, P-521	EC Diffie-Hellman Key Agreement

This document can be reproduced and distributed only whole and intact, including this copyright notice.

CAVP Cert#	Algorithm	Standard	Mode / Method	Key size	Use
Tested on OE 1: • PAA: #3877 • no PAA: #3878 Tested on OE 2: • PAA: #3879 • no PAA: #3881 Tested on OE 3: • PAA: #3875 • no PAA: #3876 Tested on OE 4: • PAA: #3882 • no PAA: #3883	SHS	[FIPS180-4]	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512		Message Digest
Tested on OE 1: • PAA: #32 • no PAA: #33 Tested on OE 2: • PAA: #34 • no PAA: #35 Tested on OE 3: • PAA: #30 • no PAA: #31 Tested on OE 4: • PAA: #36 • no PAA: #37	SHA-3	[FIPS202]	SHA3-224, SHA3-256, SHA3-384, SHA3-512		Message Digest

This document can be reproduced and distributed only whole and intact, including this copyright notice.

CAVP Cert#	Algorithm	Standard	Mode / Method	Key size	Use
Tested on OE 1: • PAA: #3149 • no PAA: #3150 Tested on OE 2: • PAA: #3151 • no PAA: #3153 Tested on OE 3: • PAA: #3147 • no PAA: #3148 Tested on OE 4: • PAA: #3154 • no PAA: #3155	HMAC	[FIPS198-1]	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 SHA3-224, SHA3-256, SHA3-384, SHA3-512	112 or greater	Message authentication code
Tested on OE 1: • PAA: #1618 • no PAA: #1619 Tested on OE 2: • PAA: #1620 • no PAA: #1622 Tested on OE 3: • PAA: #1616 • no PAA: #1617 Tested on OE 4: • PAA: #1623 • no PAA: #1624	DRBG	[SP800-90A]	**Hash_DRBG** SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 with/without PR	n/a	Random Number Generation
			HMAC_DRBG HMAC using SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 with/without PR	n/a	
			CTR_DRBG AES-128, AES-192, AES-256 with/without PR with/without DF	n/a	

Table 5 - Approved Algorithms in FIPS mode of operation

The following algorithms are non-approved but allowed algorithms in FIPS mode of operation.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

Algorithm	Key size	Use
RSA key encapsulation with encryption and decryption primitives	2048 bits or greater	Key transport; allowed in [FIPS140-2-IG] D.9
Diffie-Hellman	2048 bits or greater	Key agreement; allowed in [FIPS140-2-IG] D.8
EC Diffie-Hellman Tested on OE 1: • PAA: CVL#1371 • no PAA: CVL#1372 Tested on OE 2: • PAA: CVL#1373 • no PAA: CVL#1374 Tested on OE 3: • PAA: CVL#1369 • no PAA: CVL#1370 Tested on OE 4: • PAA: CVL#1375 • no PAA: CVL#1376	P-224, P-256, P-384, P-521	Key agreement; allowed in [FIPS140-2-IG] D.8
MD5		According to [SP800-52], MD5 is allowed to be used in TLS versions 1.0 and 1.1 as the hash function used in the PRF, as defined in [RFC2246] and [RFC4346].
SHA-1		According to [SP800-52], SHA-1 is allowed in digital signatures on ephemeral parameters in TLS.
TRNG		The module obtains entropy data from this NDRNG to seed the DRBG.

Table 6 – non-approved but allowed algorithms in FIPS mode of operation

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

Algorithm	Key size	Comments
AES		Use of a processor capability mask not allowed by this security policy, as specified in item 4 of section 4.3.2.
Triple-DES		
DSA		

This document can be reproduced and distributed only whole and intact, including this copyright notice.

Algorithm	Key size	Comments
ECDSA		
RSA		
SHA-1, SHA-2		
SHA-3		
HMAC		
DRBG		
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 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 not tested under the CAVP
DSA Signature Verification	L=512, N=160	Key sizes disallowed by [SP800-131A]
ECDSA Key Pair Generation	P-192, K-163, B-163	Key sizes disallowed by [SP800-131A]
ECDSA Signature Generation	P-192, K-163, B-163	Key sizes disallowed by [SP800-131A]
RSA Key Generation	1024 bits	Key sizes disallowed by [SP800-131A]
RSA Signature Generation	1024 bits	Key sizes disallowed by [SP800-131A]
RSA Key Encapsulation	1024 bits	Key sizes disallowed by [SP800-131A]
SHA-1 in Signature Generation		SHA-1 is disallowed by [SP800-131A], except when signature generation is used for ephemeral parameters in TLS as stated in Table 6.
Diffie-Hellman (DH)	1024 bits	Key sizes do not meet [SP800-131A]
EC Diffie-Hellman (ECDH)	P-192, K-163, B-163	Key sizes disallowed by [SP800-131A]
DES		Non-approved algorithm

This document can be reproduced and distributed only whole and intact, including this copyright notice.

Algorithm	Key size	Comments
CAST		Non-approved algorithm
Camellia		Non-approved algorithm
Blowfish		Non-approved algorithm
RC4		Non-approved algorithm
RC2		Non-approved algorithm
MD2		Non-approved algorithm
MD4		Non-approved algorithm
MD5		Non-approved algorithm
Password Based Encryption		Non-approved algorithm
HMAC-MD5		Non-approved algorithm
MDC2		Non-approved algorithm
RIPEMD		Non-approved algorithm
chacha20		Non-approved algorithm
chacha20-poly1305		Non-approved algorithm
Key Derivation Function (KDF) SP800-108		Algorithm not tested under the CAVP

Table 7 - Algorithms only allowed in non-FIPS mode of operation

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 3 and Table 4. The indicated access rights apply to the role that invokes the service.

4.2.4 Operational Rules and Assumptions

The following operational rules must be followed by any user of the cryptographic module:

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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]. The length of a single data unit encrypted with the XTS-AES shall not exceed 2^{20} AES blocks (16MB of data).
6. To meet the requirement in [FIPS140-2-IG] A.9, the module implements a check to ensure that the two AES keys used in the XTS-AES algorithm are not identical.
7. Data encryption using the same three-key Triple-DES key shall not exceed 2^{28} Triple-DES blocks (2GB of data), in accordance to [SP800-67] and [FIPS140-2-IG] IG A.13.
8. 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.
9. 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

This document can be reproduced and distributed only whole and intact, including this copyright notice.

of appropriate API calls. 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 use of the OpenSSL environment variable to alter the value of the processor capability (OpenSSL capability masks) is strictly prohibited. The only options allowed are either to leave the PAA capability enabled, or disable the PAA using the ICC_CAPABILITY_MASK property filled with all zeroes. Please refer to the user guidance for configuring this property.
5. The module is initialized automatically when the shared library is loaded in the calling application process space. The module executes the power-up self-tests (POST) and, if they are successful, the module enters the FIPS mode of operation. The calling application must include the following calling sequence to have access to the cryptographic services:

This document can be reproduced and distributed only whole and intact, including this copyright notice.

- ***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 ICC cryptographic module supports the algorithms (and modes, as applicable) listed in section 4.2.2.

4.4.2 Random Number Generation

ICC employs a Deterministic Random Bit Generator (DRBG) based on [SP800-90A] for the creation of asymmetric keys. In addition, the module provides a Random Number Generation service to calling applications.

The default algorithm is Hash_DRBG using SHA-256, but another algorithm from the Hash_DRBG, HMAC_DRBG and CTR_DRBG algorithms (see Table 5 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 Generation

Key generation has a dependency on the SP800-90A DRBG. The SP800-90 DRBG is used to generate RSA, DSA, ECDSA DH and ECDH key pairs.

In FIPS mode, RSA key generation is carried out in accordance with the algorithms described in FIPS 186-4. ECDSA key generation is carried out in accordance with the

This document can be reproduced and distributed only whole and intact, including this copyright notice.

algorithms described in FIPS 186-4 and ANSI X9.62. All key generation methods implemented in this module are compliant to SP 800-133.

The ICC module provides PKCS#1v1.5 and PSS 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#1v1.5 compatible.

In addition, there is a set of lower level interfaces for encryption and decryption where the algorithm can be used as PKCS#1v1.5 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.

4.4.4 Key Establishment

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

- Diffie-Hellman (DH) key agreement, providing between 112 and 256 bits of security strength.
- Elliptic Curve Diffie-Hellman (ECDH) key agreement, providing between 112 and 256 bits of security strength.
- RSA key encapsulation, providing between 112 and 256 bits of security strength, respectively.
- AES key wrapping, providing between 128 and 256 bits of security strength.

4.4.5 Key Entry and Output

Keys are entered into and output from the ICC module in electronic form through the data input and output interface (i.e. API function parameters). 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.

4.4.6 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.

4.4.7 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.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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 8 - 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 module 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 point (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 using the Get Status service through the following API function:

- ICC_GetStatus: shows the state of the ICC module

This document can be reproduced and distributed only whole and intact, including this copyright notice.

The function can be called anytime once the the module is initialized with the ICC_Init() and ICC_Attach API functions.

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#1v1.5) 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 – CBC
  - AES_GCM
  - AES_CCM
  - AES_XTS

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

  - SHA-1
  - SHA-256
  - SHA-512
  - SHA3-512
  - HMAC with SHA-1
  - HMAC with SHA-256
  - HMAC with SHA3-512
  - CMAC with AES

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

  - RSA signature generation with 2048-bit modulus
  - RSA signature verification with 2048-bit modulus
  - DSA signature verification with 2048-bit modulus
  - ECDSA signature generation with P-384
  - ECDSA signature verification with P-384
  - ECDSA signature generation with B-233
  - ECDSA signature verification with B-233

  Other Known Answer Tests:

This document can be reproduced and distributed only whole and intact, including this copyright notice.

- SP800-90A Hash_DRBG, HMAC_DRBG, CTR_DRBG
- RSA encryption with 2048-bit modulus
- RSA decryption with 2048-bit modulus
- ECC primitive Z computation KAT
- AES KW and KWP key wrapping / unwrapping

4.5.3 Conditional Tests

Pair-wise 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. Pair-wise consistency tests are performed on the following algorithms:

- ECDSA
- RSA

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

SP800-90A DRBG

- The SP800-90A DRBG 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 SP800-90A DRBG, two internal 8 byte cycles are performed.
- The SP800-90A DRBG 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

This document can be reproduced and distributed only whole and intact, including this copyright notice.

state is to reinitialize the cryptographic module (from an uninitialized state). The error state can be retrieved via the Get Status service (see section 4.5.1).

4.6 Mitigation of Other Attacks

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

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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_38F_KW
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

This document can be reproduced and distributed only whole and intact, including this copyright notice.