SUSE Linux Enterprise Libica Cryptographic Module

Software version 1.1

Hardware version IBM z15

FIPS 140-3 Non-Proprietary Security Policy

Version 1.1

Last update: 2024-10-01

Prepared by:

atsec information security corporation

4516 Seton Center Parkway, Suite 250

Austin, TX 78759

www.atsec.com

1 Table of Contents

1 General

This document is the non-proprietary FIPS 140-3 Security Policy for software version 1.1 and hardware version "IBM z/15" of the SUSE Linux Enterprise Libica Cryptographic Module. It has a one-to-one mapping to the [SP 800-140B] starting with section B.2.1 named “General” that maps to section 1 in this document and ending with section B.2.12 named “Mitigation of other attacks” that maps to section 12 in this document.

ISO/IEC 24759 Section 6. [Number Below]	FIPS 140-3 Section Title	Security Level
1	General	1
2	Cryptographic Module Specification	1
3	Cryptographic Module Interfaces	1
4	Roles, Services, and Authentication	1
5	Software/Firmware Security	1
6	Operational Environment	1
7	Physical Security	1
8	Non-invasive Security	N/A
9	Sensitive Security Parameter Management	1
10	Self-tests	1
11	Life-cycle Assurance	1
12	Mitigation of Other Attacks	N/A

Table 1 – Security Levels

2 Cryptographic Module Specification

2.1 Module Embodiment

The SUSE Linux Enterprise Libica Cryptographic Module (hereafter referred to as “the module”) is a software-hybrid module that provides general purpose cryptographic algorithms to applications running in the user space of the underlying operating system through a C language application program interface (API).

The module is composed by a software library, which provides the API and a subset of the cryptographic algorithms, and the Central Processor Assist for Cryptographic Functions (CPACF), which is part of the z15 processor and provides cryptographic algorithms implemented in hardware. In addition, the module uses the SUSE Linux Enterprise OpenSSL Cryptographic Module as a bound module (also referred to as “the bound OpenSSL module”), which provides additional algorithms not implemented in Libica. The SUSE Linux Enterprise OpenSSL Cryptographic Module has been FIPS 140-3 validated with certificate #4725.

For the purpose of the FIPS 140-3 validation, the module is a software-hybrid, multi-chip standalone cryptographic module validated at overall security level 1.

2.2 Module Design, Components, Versions

The software block diagram below shows the cryptographic boundary of the module, and its interfaces with the operational environment.

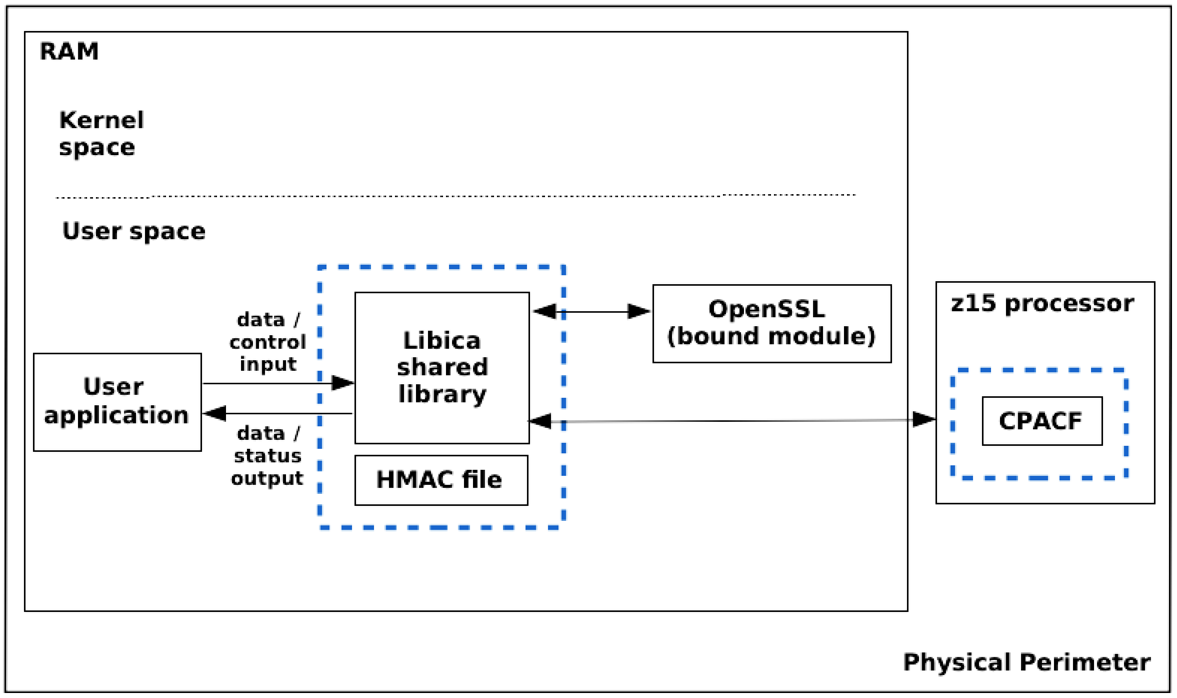

Figure 1 – Cryptographic Boundary

Table 2 lists the components of the cryptographic module that define the cryptographic boundary of the Libica module.

Component Type	Version	Components	Description
Software	1.1	/usr/lib64/libica.so.4.2.1	Shared library for cryptographic algorithms
		/usr/lib64/.libica.so.4.2.1.hmac	Integrity check HMAC value for the Libica shared library
Hardware	IBM z15	Coprocessor that implements the CPACF[1], integrated into the z15 processor.	CPACF is implemented on the z15 processor and provides processor acceleration implementations (PAI) for AES, SHA-1, SHA-2, SHA-3, SHAKE and ECC algorithm implementations.

Table 2 – Cryptographic Module Components

2.3 Modes of Operation

The module supports two modes of operation:

- Approved: only approved or allowed security functions with sufficient security strength can be used.
- Non-Approved: only non-approved security functions can be used.

When the module starts up successfully, after passing all the pre-operational self-tests, the module is operating in the approved mode of operation by default and can only be transitioned into the non-Approved mode by calling one of the non-Approved services listed in Table 11. Please see section 4 for the details on service indicator provided by the module that identifies when an approved service is called.

Critical security parameters (CSPs) used or stored in the approved mode are not used in non-approved mode, and vice versa. The module creates separate contexts for each cryptographic service; therefore, the CSPs used in approved services and non-approved services are separated by design and not shared.

2.4 Tested Operational Environments

The module has been tested on the following platforms with the corresponding module variants and configuration options:

#	Operating System	Hardware Platform	Processor	PAA/Acceleration
1	SUSE Linux Enterprise Server 15 SP4	IBM z/15 with FC3863	z15	With CPACF (PAI)

Table 3 – Tested Operational Environments

[1] The IBM z/15 hardware must have Feature Code 3863 (FC3863) enabled. This feature code enables processor acceleration implementations (PAI) for AES, and ECC algorithms.

2.5 Vendor-Affirmed Operational Environments

In addition to the platforms listed in Table 3, SUSE has also tested the module on the platforms in Table 4, and claims vendor affirmation on them.

Note: the 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.

#	Operating System	Hardware platform	Processor	PAA/Acceleration
1	SUSE Linux Enterprise Server 15SP4	IBM LinuxONE III LT1 with FC3863	z15	With CPACF (PAI)
2	SUSE Linux Enterprise Micro 5.3	IBM z/15 with FC3863	z15	With CPACF (PAI)
3	SUSE Linux Enterprise Micro 5.3	IBM LinuxONE III LT1 with FC3863	z15	With CPACF (PAI)
4	SUSE Linux Enterprise Base Container Image 15SP4	IBM z/15 with FC3863	z15	With CPACF (PAI)
5	SUSE Linux Enterprise Base Container Image 15SP4	IBM LinuxONE III LT1 with FC3863	z15	With CPACF (PAI)

Table 4 - Vendor-Affirmed Operational Environments

2.6 Approved Algorithms

Table 5 below lists all security functions of the Libica module, including specific key strengths employed for approved services, and implemented modes of operation.

CAVP Cert	Algorithm and Standard	Mode / Method	Description / Key Size(s) / Key Strength(s)	Use / Function
A3378	AES FIPS197, SP800-38A	CBC	128, 192, 256-bit keys with 128-256 bits of security strength	Symmetric encryption; Symmetric decryption
	AES FIPS197, SP800-38A-add	CBC_CS1 CBC_CS2 CBC_CS3	128, 192, 256-bit keys with 128-256 bits of security strength	Symmetric encryption; Symmetric decryption
	AES SP800-38C	CCM	128, 192, 256-bit keys with 128-256 bits of security strength	Authenticated encryption; Authenticated decryption
	AES FIPS197, SP800-38A	CFB8, CFB128	128, 192, 256-bit keys with 128-256 bits of security strength	Symmetric encryption; Symmetric decryption

CAVP Cert	Algorithm and Standard	Mode / Method	Description / Key Size(s) / Key Strength(s)	Use / Function
	AES SP800-38B	CMAC	128, 192, 256-bit keys with 128-256 bits of security strength	Message authentication code (MAC)
	AES FIPS197, SP800-38A	CTR	128, 192, 256-bit keys with 128-256 bits of security strength	Symmetric encryption; Symmetric decryption
	AES FIPS197, SP800-38A	ECB	128, 192, 256-bit keys with 128-256 bits of security strength	Symmetric encryption; Symmetric decryption
	AES FIPS197, SP800-38A	OFB	128, 192, 256-bit keys with 128-256 bits of security strength	Symmetric encryption; Symmetric decryption
	AES SP800-38E	XTS	128, 256-bit keys with 128 and 256 bit of security strength	Symmetric encryption and symmetric decryption (for data storage)
A3377, A3378	AES FIPS197, SP800-38D	GCM with internal IV (section 8.2.2)	128, 192, 256-bit keys with 128-256 bits of security strength	Authenticated encryption; Authenticated decryption
A3377, A3378	AES FIPS197, SP800-38D	GMAC	128, 192, 256-bit keys with 128-256 bits of security strength	Message authentication code (MAC)
A3378	ECDSA FIPS186-4	SHA2-224, SHA2-256, SHA2-384, SHA2-512	P-256, P-384, P-521 with 128-256 bits of security strength	Digital signature generation
		SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	P-256, P-384, P-521 with 128-256 bits of security strength	Digital signature verification (usage of SHA-1 is considered Legacy Use)
A3378	KAS-ECC-SSC SP800-56Arev3	ECC Ephemeral Unified Scheme	P-256, P-384, P-521 with 128-256 bits of security strength	Shared secret computation
A3378	SHA-3 FIPS202	SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128, SHAKE-256	N/A	Message digest

CAVP Cert	Algorithm and Standard	Mode / Method	Description / Key Size(s) / Key Strength(s)	Use / Function
A3378	SHS FIPS180-5	SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256	N/A	Message digest

Table 5 – Approved Algorithms provided by Libica module.

Table 6 lists the approved algorithms that are used by the module but provided by the bound OpenSSL module in the approved mode of operation.

CAVP Cert	Algorithm and Standard	Mode / Method	Description / Key Size(s) / Key Strength(s)	Use / Function
Vendor Affirmed	CKG SP800-133rev2	Asymmetric key generation (FIPS-186-4, SP800-56Arev3, SP800-90Arev1)	RSA: 2048, 3072, 4096-bit keys with 112, 128, 149 bits of security strength ECDSA: P-256, P-384, P-521 elliptic curves with 128-256 bits of security strength	Key Generation
A3150	DRBG SP800-90Arev1	CTR_DRBG: AES-256 with DF, without PR	128, 192, 256-bit keys with 128-256 bits of security strength	Random number generation
A3147	ECDSA FIPS186-4	B.4.2 Testing Candidates	P-224, P-256, P-384, P-521 with 112-256 bits of security strength	Key generation
E28, E29	ESV SP 800-90B	CPU Time Jitter RNG (SHA3-256 Conditioning Component)	N/A	Random number generation
A3147	HMAC FIPS198-1	SHA2-256	256-bit key with 256 bits of security strength	Integrity test
A3147	RSA FIPS 186-4	B.3.3 Random Probable Primes	2048, 3072 and 4096-bit keys with 112-149 bits of security strength	Key pair generation

CAVP Cert	Algorithm and Standard	Mode / Method	Description / Key Size(s) / Key Strength(s)	Use / Function
A3147	SHS FIPS180-5	SHA2-256	N/A	Integrity test

Table 6 – Approved Algorithms provided by the bound OpenSSL module.

2.7 Non-Approved Algorithms Allowed in the Approved Mode of Operation

The module does not implement non-approved algorithms that are allowed in the approved mode of operation.

2.8 Non-Approved Algorithms Allowed in the Approved Mode of Operation with No Security Claimed

The module does not implement non-approved algorithms that are allowed in the approved mode of operation.

2.9 Non-Approved Algorithms Not Allowed in the Approved Mode of Operation

Table 7 lists non-approved algorithms implemented in the Libica module that are not allowed in the approved mode of operation. These algorithms are used by the non-approved services listed in Table 11. The Libica mode does not use any non-approved algorithm implemented in the bound OpenSSL module.

Algorithm/Functions	Use/Function
RSA modulus exponentiation primitive.	Sign, verify, encrypt, and decrypt primitives.
DRBG	Random Number Generation service

Table 7 – Non-Approved Not Allowed in the Approved Mode of Operation

3 Cryptographic Module Ports and Interfaces

The logical interfaces are the API through which applications request services. The ports and interfaces are shown in the following table.

All data output via data output interface is inhibited when the module is performing pre-operational test or zeroization or when the module enters error state.

Logical Interface[2]	Physical Port	Data that passes over port/interface
Data Input	None	API input parameters for data.
Data Output	None	API output parameters for data.
Control Input	None	API function calls, API input parameters for control input, /proc/sys/crypto/fips_enabled control file, ICAPATH environment variable.
Status Output	None	API return codes, API output parameters for status output.
Power Input	PC Power Supply Port	N/A

Table 8 – Ports and Interfaces

[2] The control output interface is omitted on purpose because the module does not implement it.

4 Roles, services, and authentication

4.1 Services

The module supports the Crypto Officer role only. This sole role is implicitly assumed by the operator of the module when performing a service. The module does not support authentication.

Role	Service	Input	Output
Crypto Officer (CO)	Symmetric decryption	Ciphertext, key	Plaintext
	Symmetric encryption	Plaintext, key	Ciphertext
	Authenticated encryption	Plaintext, key	Ciphertext, authentication tag
	Authenticated decryption	Ciphertext, authentication tag, key	Plaintext, return code (pass/fail)
	Key pair generation	Key size	Key pair
	Digital signature generation	Message, hash algorithm, private key	Signature
	Digital signature verification	Signature, hash algorithm, public key	Signature verification result
	EC Diffie-Hellman shared secret computation	Private key, public key from peer	Shared secret
	Message authentication code (MAC) generation	Message, key	Message authentication code
	Message authentication code (MAC) verification	Message, key, message authentication code	Return code (pass/fail)
	Message digest	Message	Message digest
	On-demand integrity test and self-tests	None	Return codes/log messages
	Random number generation	Number of bytes	Random numbers
	Show module name and version	None	Module name and version
	Module installation and configuration	None	Log messages
	Zeroization	Context containing SSPs	None

Table 9 – Roles, Service Commands, Input and Output

The module provides services to the users that assume one of the available roles. All services are shown in Table 10 and Table 11.

4.2 Approved Services

Table 10 lists the approved services. For each service, the table lists the associated cryptographic algorithm(s), the role to perform the service, the cryptographic keys or CSPs involved, and their access type(s). The following convention is used to specify access rights to a CSP:

- **G** = **Generate**: The module generates or derives the SSP.
- **R** = **Read**: The SSP is read from the module (e.g., the SSP is output).
- **W** = **Write**: The SSP is updated, imported, or written to the module.
- **E** = **Execute**: The module uses the SSP in performing a cryptographic operation.
- **Z** = **Zeroise**: The module zeroizes the SSP.
- **N/A**: the calling application does not access any CSP or key during its operation.

The details of the approved cryptographic algorithms including the CAVP certificate numbers can be found in Table 5. The module implements the `ica_get_fips_indicator()` API function that outputs the value of `.fips_approved` field, when "1" is output it, indicates that the service is approved.

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and/or SSPs	Indicator
Cryptographic Services						
Symmetric encryption	Perform AES encryption	AES	AES key	CO	W, E	`ica_get_fips_indicator()` returns `.fips_approved = 1`
Symmetric decryption	Perform AES decryption	AES	AES key		W, E	`ica_get_fips_indicator()` returns `.fips_approved = 1`
Authenticated encryption	Perform authenticated AES encryption	AES	AES key	CO	W, E	`ica_get_fips_indicator()` returns `.fips_approved = 1`
Authenticated decryption	Perform authenticated AES decryption	AES	AES key		W, E	`ica_get_fips_indicator()` returns `.fips_approved = 1`
Asymmetric key generation[3]	Generate RSA key pairs	RSA, DRBG	Module-generated RSA public key, Module-generated RSA private key		E, G, R	`ica_get_fips_indicator()` returns `.fips_approved = 1`

[3] This service is providing using algorithms implemented in the bound OpenSSL module.

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and/or SSPs	Indicator
	Generate ECDSA key pairs	ECDSA, DRBG	Module-generated ECDSA public key, Module-generated ECDSA private key		E, G, R	`ica_get_fips_indicator() returns .fips_approved = 1`
Digital signature generation	Generate ECDSA signature	ECDSA, DRBG, SHS	ECDSA private key		W, E	`ica_get_fips_indicator() returns .fips_approved = 1`
Digital signature verification	Verify ECDSA signature	ECDSA, SHS	ECDSA public key		W, E	`ica_get_fips_indicator() returns .fips_approved = 1`
Shared secret computation	EC Diffie-Hellman shared secret computation	KAS-ECC-SSC	EC Diffie-Hellman private key		W, E	`ica_get_fips_indicator() returns .fips_approved = 1`
			EC Diffie-Hellman public key from peer		W, E	
			EC Diffie-Hellman shared secret		E, G, R	
Message digest	Compute SHA hashes	SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	None		N/A	`ica_get_fips_indicator() returns .fips_approved = 1`
		SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128, SHAKE-256	None		N/A	`ica_get_fips_indicator() returns .fips_approved = 1`
Message authentication code (MAC) generation	Compute AES-based CMAC	CMAC with AES	AES key		W, E	`ica_get_fips_indicator() returns .fips_approved = 1`
	Compute AES-based GMAC	GMAC with AES	AES key			`ica_get_fips_indicator() returns .fips_approved = 1`
Message authentication code (MAC) verification	Verify AES-based CMAC	CMAC with AES	AES key		W, E	`ica_get_fips_indicator() returns .fips_approved = 1`
	Verify AES-based GMAC	GMAC with AES	AES key			`ica_get_fips_indicator() returns .fips_approved = 1`

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and/or SSPs	Indicator
Other FIPS-related Services						
Show status	Show module status	N/A	None	CO	N/A	Implicit (always approved)
Zeroization	Zeroize CSPs	N/A	All CSPs		Z	Implicit (always approved)
On-demand Integrity test and self-tests	Perform integrity test and self-tests on-demand	AES, ECDSA, DRBG, HMAC, RSA, SHS	None		N/A	Implicit (always approved)
Self-tests	Perform self-tests	AES, ECDSA, DRBG, HMAC, RSA, SHS	None		N/A	Implicit (always approved)
Show module name and version	Show module name and version	N/A	None		N/A	Implicit (always approved)
Module installation and configuration	Install and configure module	N/A	None		N/A	Implicit (always approved)

Table 10 - Approved Services

Table 11 lists the non-approved services. The details of the non-approved cryptographic algorithms available in the non-approved mode can be found in Table 7.

Service	Description	Algorithms Accessed	Role
Cryptographic Services			
Random Number Generation	Obtain random numbers	DRBG SHA2-512	CO
RSA	RSA sign, verify, encrypt and decrypt primitives	RSA	

Table 11 - Non-Approved Services

5 Software/Firmware security

5.1 Integrity Techniques

The integrity of the module is verified by comparing an HMAC-SHA2-256 value calculated at run time with the HMAC value stored in the .hmac file that was computed at build time for each software component of the module. If the HMAC values do not match, the test fails and the module enters the error state. The module uses the HMAC-SHA2-256 algorithm provided by the OpenSSL bound module.

5.2 On-Demand Integrity Test

Integrity tests are performed as part of the Pre-Operational Self-Tests.

The module provides the Self-Test service to perform self-tests on demand which includes the pre-operational tests (i.e., integrity test) and cryptographic algorithm self-tests (CASTs). This service can be invoked by using the `ica_fips_powerup_tests()` API function call. During the execution of the on-demand self-tests, services are not available, and no data output or input is possible.

In order to verify whether the self-tests have succeeded and the module is in the Operational state, the calling application may invoke the `ica_fips_status()`. See section 10.3 for more information about the possible return values.

5.3 Executable Code

The module consists of executable code in the form of the Libica file stated in Table 2.

6 Operational Environment

6.1 Applicability

This module operates in a modifiable operational environment per the FIPS 140-3 level 1 specifications. The SUSE Linux Enterprise Server operating system is used as the basis of other products. Compliance is maintained for SUSE products whenever the binary is found unchanged per the vendor affirmation from SUSE based on the allowance FIPS 140-3 management manual [FIPS140-3_MM] section 7.9.1 item 1. c) i).

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

6.2 Policy

Instrumentation tools like the ptrace system call, gdb and strace utilities, as well as other tracing mechanisms offered by the Linux environment such as ftrace or systemtap, shall not be used in the operational environment. The use of any of these tools implies that the cryptographic module is running in a non-tested operational environment.

6.3 Requirements

The module shall be installed as stated in section 11. The operating system provides process isolation and memory protection mechanisms that ensure appropriate separation for memory access among the processes on the system. Each process has control over its own data and uncontrolled access to the data of other processes is prevented.

7 Physical Security

The Libica module inherits the physical characteristics of the host running it; the module has no physical security characteristics of its own. Figure 3 illustrates the IBM System z15 mainframe computer that represents the testing platform, that includes the hardware component of the cryptographic module.

Figure 2 – IBM z15 Mainframe Computer

The Central Processor Assist for Cryptographic Functions (CPACF) is part of the CoProcessor Unit (CoP) integrated in the z15 processor, and offers full implementation of several algorithms; this module uses the processor algorithm implementations (PAI) for AES, SHA-1, SHA-2, SHA-3, SHAKE, ECDSA signature generation and verification, and ECDH shared secret computation. The module is a multi-chip standalone with a physical security level of 1. This security level is satisfied by the device (CoP) being included within the cryptographic boundary of the module and the device being made of production grade components that include standard passivation techniques. The module does not implement a maintenance access interface.

With regards to the CPACF physical design, each microprocessor (core) on the 8-core chip (see Figure 4) has its own dedicated CoP, which implements the crypto instructions and provides the hardware compression function. The compression unit is integrated with the CPACF, benefiting from combining (sharing) the use of buffers and interfaces.

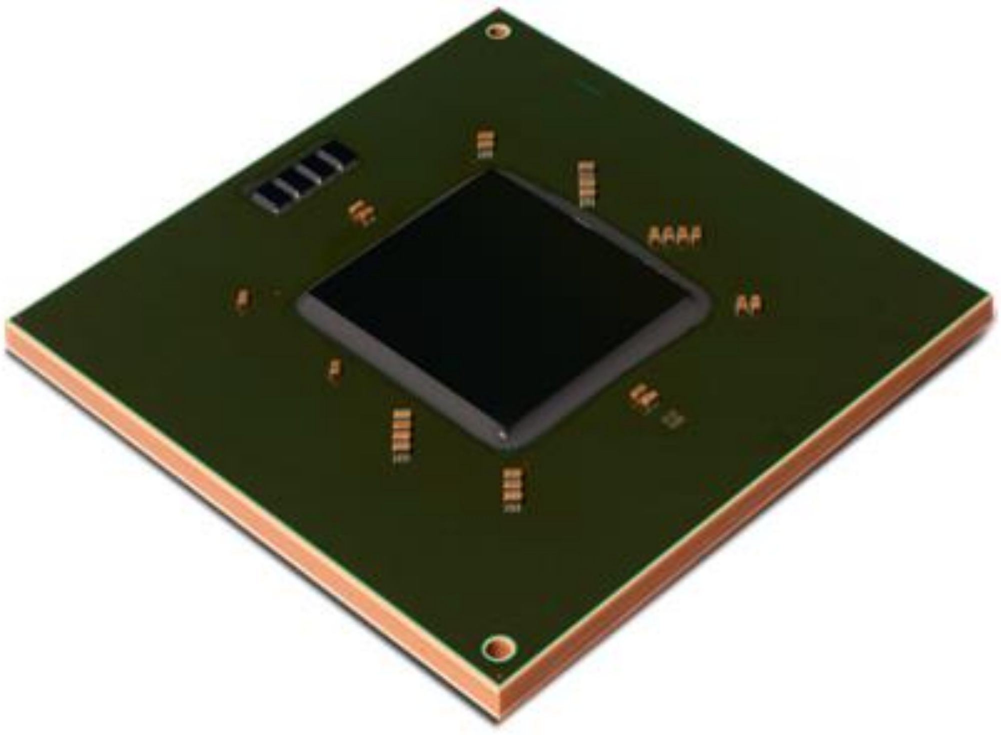

Figure 3 – IBM z15 Processor Unit Chip

8 Non-invasive Security

This module does not implement any non-invasive security mechanism, and therefore this section is not applicable.

9 Sensitive Security Parameter Management

Table 12 summarizes the Sensitive Security Parameters (SSPs) that are used by the cryptographic services implemented in the module.

Key/SSP Name/ Type	Strength	Security Function and Cert. Number	Generation	Import/Export	Establishment	Storage	Zeroization	Use & related keys
AES key	128, 192, 256	AES-CBC, AES-CBC-CS1, AES-CBC-CS2, AES-CBC-CS3, AES-CCM, AES-CFB128, AES-CFB8, AES-CMAC, AES-CTR, AES-ECB, AES-GCM, AES-GMAC, AES-OFB, AES-XTS[4] A3378 AES-GCM, AES-GMAC, A3377	N/A	**Import:** CM from TOEPP Path. Passed into the module via API parameters in plaintext (P) format. **Export:** N/A	N/A	RAM	Zeroized by calling application.	**Use:** Symmetric encryption; Symmetric decryption; Authenticated encryption; Authenticated decryption; Message authentication code (MAC) **Related SSPs:** N/A
ECDSA public key	128, 192, 256	ECDSA A3378	N/A	**Import:** CM from TOEPP Path. Passed into the module via API parameters in plaintext (P) format. **Export:** N/A	N/A	RAM	Zeroized by calling application.	**Use:** Digital signature verification **Related SSPs:** ECDSA private key
ECDSA private key	128, 192, 256	ECDSA A3378	N/A	**Import:** CM from TOEPP Path. Passed into the module via API parameters in plaintext (P) format. **Export:** N/A	N/A	RAM	Zeroized by calling application.	**Use:** Digital signature generation **Related SSPs:** ECDSA public key
EC Diffie-Hellman private key	128, 192, 256	KAS-ECC-SSC A3378	N/A	**Import:** CM from TOEPP Path. Passed into the module via API parameters in plaintext (P) format. **Export:** N/A	N/A	RAM	Zeroized by calling application.	**Use:** EC Diffie-Hellman shared secret computation **Related SSPs:** N/A

[4] AES XTS only supports security strengths of 128 and 256 bits.

Key/SSP Name/ Type	Strength	Security Function and Cert. Number	Generation	Import/Export	Establishment	Storage	Zeroization	Use & related keys
EC Diffie-Hellman public key from peer	128, 192, 256	KAS-ECC-SSC A3378	N/A	**Import:** CM from TOEPP Path. Passed into the module via API parameters in plaintext (P) format. **Export:** N/A	N/A	RAM	Zeroized by calling application.	**Use:** EC Diffie-Hellman shared secret computation **Related SSPs:** N/A
EC Diffie-Hellman shared secret	128, 192, 256	KAS-ECC-SSC A3378	N/A	**Import:** N/A **Export:** CM to TOEPP Path. Passed out from the module via API output parameters in plaintext.	Established during the EC Diffie-Hellman shared secret computation per SP800-56Arev3.	RAM	Zeroized by calling application.	**Use:** EC Diffie-Hellman shared secret computation **Related SSPs:** EC Diffie-Hellman public key from peer; EC Diffie-Hellman private key
SSPs used for cryptographic services provided by the bound OpenSSL module								
Module-generated ECDSA public key	128, 192, 256	ECDSA A3147	B.4.2 Testing Candidates Generated by the bound OpenSSL using the FIPS 186-4 random values are obtained from the OpenSSL SP800-90Arev1 DRBG.	**Import:** N/A **Export:** CM to TOEPP Path. Passed out from the module via API parameters in plaintext (P) format.	N/A	RAM	Zeroized by calling application. Internally zeroized via EVP_PKEY_CTX_free, EVP_PKEY_free, OPENSSL_free	**Use:** Key generation **Related SSPs:** DRBG internal state, Module-generated ECDSA private key
Module-generated ECDSA private key	128, 192, 256	ECDSA A3147	B.4.2 Testing Candidates Generated by the bound OpenSSL using the FIPS 186-4 random values are obtained from the OpenSSL SP800-90Arev1 DRBG.	**Import:** N/A **Export:** CM to TOEPP Path. Passed out from the module via API parameters in plaintext (P) format.	N/A	RAM	Zeroized by calling application. Internally zeroized via EVP_PKEY_CTX_free, EVP_PKEY_free, OPENSSL_free	**Use:** Key generation **Related SSPs:** DRBG internal state, Module-generated ECDSA public key

Key/SSP Name/ Type	Strength	Security Function and Cert. Number	Generation	Import/Export	Establishment	Storage	Zeroization	Use & related keys
Module-generated RSA public key	112, 128, 149	RSA A3147, CTR_DRBG A3150	Generated by the bound OpenSSL using method B.3.3 specified in FIPS 186-4; random values are obtained from the SP800-90Arev1 DRBG.	**Import:** N/A **Export:** CM to TOEPP Path. Passed out from the module via API parameters in plaintext (P) format.	N/A	RAM	Zeroized by calling application. Internally zeroized via RSA_free	**Use:** Key generation **Related SSPs:** DRBG Internal state, Module-generated RSA private key
Entropy input IG D.L compliant	192 to 384 bits	CTR_DRBG A3150	Obtained from the entropy source	**Import/Export:** N/A; it remains within the cryptographic boundary.	N/A	RAM	FIPS_drbg_free	**Use:** Random number generation **Related SSPs:** DRBG seed
DRBG seed IG D.L compliant	192 to 384 bits	CTR_DRBG A3150	Generated from the entropy input as defined in SP800-90Arev1	**Import/Export:** N/A; it remains within the cryptographic boundary.	N/A	RAM	FIPS_drbg_free	**Use:** Random number generation **Related SSPs:** Entropy input, DRBG Internal state
DRBG internal state (V, key) IG D.L compliant	128 to 256 bits	CTR_DRBG A3150	Generated from the DRBG seed as defined in SP800-90Arev1	**Import/Export:** N/A; it remains within the cryptographic boundary.	N/A	RAM	FIPS_drbg_free	**Use:** Random number generation **Related SSPs:** DRBG seed

Table 12 – SSPs

The following sections describe how CSPs, in particular cryptographic keys, are managed during its life cycle.

9.1 Random Number Generation

The Deterministic Random Bit Generator (DRBG) is provided by the bound OpenSSL module. The DRBG is based on [SP800-90Arev1] for the creation of RSA and ECDSA keys in OpenSSL, and for the random generation of the IV used in AES GCM and the k value used in ECDSA signature generation. The use of the DRBG provided by the bound OpenSSL module is only internal. Notice that the Libica module implements a separate DRBG for providing a random generation number service, but this algorithm implementation, as well as the service, are considered non-approved.

The DRBG is initialized during initialization of the bounded OpenSSL module; this module loads the DRBG by default using the CTR_DRBG mechanism with AES-256, with derivation function, and without prediction resistance.

The bound Open SSL module uses an SP800-90B-compliant entropy source specified in Table 13. This entropy source is located within the physical perimeter, but outside of the cryptographic boundary of the module. The module obtains 384 bits to seed the DRBG, and 256 bits to reseed it, sufficient to provide a DRBG with 256 bits of security strength.

Entropy Source	Minimum number of bits of entropy	Details
ESV certs. E28, E29	256 bits of entropy in the 256-bit output	Standalone Userspace CPU Time Jitter RNG version 3.4.0 entropy source (with SHA-3 as the vetted conditioning component) is located within the physical perimeter of the operational environment but outside the cryptographic boundary of the bound OpenSSL module.

Table 13 - Non-Deterministic Random Number Generation Specification

9.2 SSP Generation

The bounded OpenSSL module provides an [SP800-90Arev1]-compliant Deterministic Random Bit Generator (DRBG) for the creation of key components of asymmetric keys, and random number generation.

For generating RSA, ECDSA keys, the bounded module implements asymmetric key generation services compliant with [FIPS186-4]. A seed (i.e., the random value) used in asymmetric key generation is directly obtained from the [SP800-90Arev1] DRBG.

The public and private keys used in EC Diffie-Hellman shared secret computation are generated internally by the bound OpenSSL module using the ECDSA key generation method compliant with [FIPS186-4] and [SP800-56Arev3]. In accordance with FIPS 140-3 IG D.H, the bound cryptographic module performs Cryptographic Key Generation (CKG) for asymmetric keys as per section 5.1 of SP800-133rev2 (vendor affirmed) by obtaining a random bit string directly from an approved DRBG and that can support the required security strength requested by the caller (without any V, as described in Additional Comments 2 of IG D.H).

9.3 SSP Transport

The module does not provide key transport mechanisms.

9.4 SSP Establishment

The module provides EC Diffie-Hellman shared secret computation compliant with SP800-56Arev3, in accordance with scenario 2 (1) of IG D.F.

According to Table 2: Comparable strengths in [SP800-57rev5], the key sizes of EC Diffie-Hellman provides the following security strength in the approved mode of operation:

- EC Diffie-Hellman key agreement provides between 128 and 256 bits of encryption strength.

9.5 SSP Entry and Output

The module does not support direct manual SSP entry or intermediate SSP generation output. The SSPs are provided to the module via API input parameters in plaintext form and output via API output parameters in plaintext form within the physical perimeter of the operational environment. This is allowed by [FIPS140-3_IG] 9.5.A, according to the “CM Software to/from App via TOEPP Path” entry on the Key Establishment Table.

9.6 SSP Storage

The module does not perform persistent storage of SSPs. The SSPs are temporarily stored in the RAM in plaintext form. SSPs are provided to the module by the calling process and are destroyed when released by the appropriate zeroization function calls.

9.7 SSP Zeroization

For those SSPs that are either imported or exported, it is the responsibility of the calling application to zeroize them once they are no longer utilized.

Internal values to store SSPs are zeroized automatically before returning the control to the calling application. Zeroization is performed by overwriting the memory with zeroes.

For services implemented in the bound OpenSSL module, the Libica module calls internally the appropriate zeroization functions provided by the bound module (e.g. OPENSSL_cleanse) before returning to the calling application. The zeroization functions overwrite the memory occupied by SSPs with “zeros” and deallocate the memory with the regular memory deallocation operating system call.

The completion of a zeroization routine will indicate that a zeroization procedure succeeded. Also, module reset can zeroize all SSPs for both the module and the OpenSSL bound module.

10 Self-tests

The module performs pre-operational tests and CASTs automatically when the module is loaded into memory. Pre-operational tests ensure that the module is not corrupted, and the CASTs ensure that the cryptographic algorithms work as expected. While the module is executing the pre-operational tests and CASTs, services are not available, and input and output are inhibited. The module is not available for use by the calling application until the pre-operational tests are completed successfully. After the pre-operational test and the CASTs succeed, the module becomes operational. If any of the pre-operational test or any of the CASTs fail an error message is returned, and the module transitions to the error state.

10.1 Pre-Operational Tests

The module performs the integrity test of the software component using HMAC-SHA2-256. The integrity test uses the HMAC algorithm implemented in the bound OpenSSL module, which tests this algorithm as part of the CASTs when loaded into memory and before Libica performs the pre-operational tests. The details of integrity test are provided in section 5.1.

10.2 Conditional Tests

10.2.1 Cryptographic algorithm tests

Table 14 specifies the CASTs performed by the Libica module. All CASTs performed are in the form of the Known Answer Tests (KATs) and are run prior to performing the integrity test. A KAT includes the comparison of a calculated output with an expected known answer, hard coded as part of the test vectors used in the test. If the values do not match, the KAT fails.

Algorithm	Condition	Test
AES	Power on	KAT AES ECB mode with 128, 192 and 256 bit keys, encryption and decryption (separately tested). KAT AES CBC mode with 128, 192 and 256 bit keys, encryption and decryption (separately tested). KAT AES CBC_CS mode with 128, 192 and 256 bit keys, encryption and decryption (separately tested). KAT AES CFB mode with 128, 192 and 256 bit keys, encryption and decryption (separately tested). KAT AES OFB mode with 128, 192 and 256 bit keys, encryption and decryption (separately tested). KAT AES CTR mode with 128, 192 and 256 bit keys, encryption and decryption (separately tested). KAT AES CCM mode with 128, 192 and 256 bit keys, encryption and decryption (separately tested). KAT AES GCM mode with 128, 192 and 256 bit keys, encryption and decryption (separately tested). KAT AES XTS mode with 128 and 256 bit keys, encryption and decryption (separately tested).
CMAC	Power on	KAT AES CMAC with 128-bit key, MAC generation.
GMAC	Power on	KAT AES GMAC with 128-bit key, MAC generation.

Algorithm	Condition	Test
ECDH	Power on	KAT ECDH shared secret computation with P-256.
ECDSA	Power on	KAT ECDSA with P-256 and SHA2-256, signature generation and verification (separately tested),
SHA-3	Power on	KAT SHA3-224, SHA3-256, SHA3-384 and SHA3-512.
SHS	Power on	KAT SHA-1, SHA2-224, SHA2-256, SHA2-384 and SHA2-512.

Table 14 – Conditional Cryptographic Algorithms Self-Tests performed by the Libica module

Table 15 lists the CASTs performed by the bound OpenSSL module for the services provided in the approved mode of operation.

Algorithm	Condition	Test
DRBG	Power on	KAT CTR_DRBG with AES with 256-bit keys with and without DF, with and without PR Health tests according to section 11.3 of [SP800-90Arev1].
HMAC	Power on	KAT HMAC-SHA2-256.
SHA	Power on	KAT SHA2-256.

Table 15 – Conditional Cryptographic Algorithms Self-Tests performed by the OpenSSL module

10.2.2 Pairwise Consistency Test

The bound OpenSSL module performs the Pair-wise Consistency Tests (PCT) shown in Table 16. If at least one of the tests fails, the module returns an error code and enters the Error state. When the module is in the Error state, no data is output, and cryptographic operations are not allowed.

Algorithm	Test
ECDSA key generation	PCT using SHA2-256, signature generation and verification.
ECDH key generation	Covered by ECDSA PCT as allowed by IG 10.3, additional comment 1.
RSA key generation	PCT using SHA2-256, signature generation and verification.

Table 16 - Pairwise Consistency Test performed by the bound OpenSSL module

10.2.3 Periodic/On-Demand Self-Tests

On-Demand self-tests can be invoked by calling the `fips_powerup_tests()` API function, which causes the module to run the pre-operational tests again. During the execution of the on-demand self-tests, services are not available, and no data output or input is possible.

To verify whether the self-tests have succeeded, and the module is in the Operational state, the calling application may invoke the `ica_fips_status()` API function. See section 10.3 for more information about the possible return values.

10.3 Error States

When the module fails any pre-operational self-test or conditional test, the module will return an error code to indicate the error and will enter the Error state. Any further cryptographic operation is inhibited. The calling application can obtain the module state by calling the `ica_fips_status()` API function. The function can return one following flags indicating the module is in the error state.

Error State	Cause of Error	Status Indicator (bit flag)
Self-test error state	Failure of CAST	ICA_FIPS_CRYPTOALG
	Failure of integrity tests	ICA_FIPS_INTEGRITY

Table 17 - Error States

For errors occurred in the bound OpenSSL, any transition to the error state of the bound OpenSSL module puts the Libica module in the error state.

11 Life-cycle assurance

11.1 Delivery and Operation

11.1.1 Module Installation

The Crypto Officer can install the RPM packages containing the module as listed in Table 19 using the zypper tool as follows.

```
# zypper install libica4
```

The integrity of the RPM package is automatically verified during the installation, and the Crypto Officer shall not install the RPM package if there is any integrity error.

11.1.2 Operating Environment Configuration

The module requires to run on a z15 processor that has Feature Code 3863 enabled. This feature enables processor acceleration implementations (PAI) in CPACF for AES and ECC algorithms (SHA-1, SHA-2, SHA-3 and SHAKE algorithms are enabled by default in z15 processors). Make sure with your system administrator that the hardware platform meets this requirement.

In addition, the operating environment needs to be configured to support the approved mode of operation, so the following steps shall be performed with the root privilege:

1. Install the dracut-fips RPM package:

```
# zypper install dracut-fips
```

2. Recreate the INITRAMFS image:

```
# dracut -f
```

3. After regenerating the initrd, the Crypto Officer has to append the following parameter in the /etc/default/grub configuration file in the GRUB_CMDLINE_LINUX_DEFAULT line:

```
fips=1
```

4. After editing the configuration file, please run the following command to change the setting in the boot loader:

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

If /boot or /boot/efi resides on a separate partition, the kernel parameter boot=<partition of /boot or /boot/efi> must be supplied. The partition can be identified with the command "df /boot" or "df /boot/efi" respectively. For example:

```
# df /boot
Filesystem     1K-blocks     Used     Available     Use%     Mounted on
/dev/sda1      233191        30454    190296        14%      /boot
```

The partition of /boot is located on /dev/sda1 in this example. Therefore, the following string needs to be appended in the aforementioned grub file:

```
"boot=/dev/sda1"
```

5. Reboot to apply these settings.

Now, the operating environment is configured to support the approved mode of operation. The Crypto Officer should check the existence of the file /proc/sys/crypto/fips_enabled, and verify it contains a numeric value “1”. If the file does not exist or does not contain “1”, the operating environment is not configured to support the approved mode of operation and the module will not operate as a FIPS validated module properly.

11.1.3 Module Installation for Vendor Affirmed Platforms

Table 18 includes the information on module installation process for the vendor affirmed platforms that are listed in Table 4.

Product	Link
SUSE Linux Enterprise Micro 5.3	https://documentation.suse.com/sle-micro/5.3/single-html/SLE-Micro-security/#sec-fips-slemicro-install
SUSE Linux Enterprise Base Container Image 15SP4	https://documentation.suse.com/smart/linux/html/concept-bci/index.html

Table 18 - Installation for Vendor Affirmed Platforms

Note: Per section 7.9 in the FIPS 140-3 Management Manual [FIPS140-3_MM], the Cryptographic Module Validation Program (CMVP) makes no statement as to the correct operation of the module or the security strengths of the generated keys when this module is ported and executed in an operational environment not listed on the validation certificate.

11.1.4 End of Life Procedure

For secure sanitization of the cryptographic module, the module must first to be powered off, which will zeroize all keys and CSPs in volatile memory. Then, for actual deprecation, the module shall be upgraded to a newer version that is FIPS 140-3 validated.

The module does not possess persistent storage of SSPs, so further sanitization steps are not required.

11.2 Crypto Officer Guidance

The binaries of the module are contained in the RPM packages for delivery. The Crypto Officer shall follow sections 11.1.1 and 11.1.2 to configure the operational environment and install the module to be operated as a FIPS 140-3 validated module.

Table 19 lists the RPM package that contains the FIPS validated module and the OE directory where the components are installed. The "Show module name and version" service is implemented via the following API functions:

- `ica_get_build_version()` returns the software version, whose value is “FIPS-SUSE-4.2.1-150400.3.8.1”. This identifier matches the version included in the RPM package filenames.
- `ica_get_hw_info()` returns the hardware version, whose value is “IBM/S390, model type 8561”. This identifier corresponds to the IBM z/15 hardware with z15 processor.

Processor Architecture	RPM Packages
z15	libica-4.2.1-150400.3.8.1.s390x.rpm

Table 19 – RPM packages

11.2.1 AES XTS

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, that is 16MB of data.

To meet the requirement stated in IG C.I, the module implements a check that ensures, before performing any cryptographic operation, that the two AES keys used in AES XTS mode are not identical.

11.2.2 AES GCM IV

The AES GCM IV generation is in compliance with section 8.2.2 of [SP800-38D] and IG C.H scenario 2 [FIPS140-3_IG], in which the GCM IV is generated internally at its entirety randomly. The DRBG provided by the OpenSSL bound module, compliant with SP800-90A-rev1, is used for generating the IV. The DRBG is fully seeded with entropy provided by the non-physical entropy source that is not within the cryptographic boundary of the module but within its physical perimeter.

The GCM IV must be at least 96 bits in length, which is enforced by the module.

When a GCM IV is used for decryption, the responsibility for the IV generation lies with the party that performs the AES GCM encryption.

12 Mitigation of other attacks

The module does not implement any mitigation mechanism.

Appendix A. Glossary and Abbreviations

AES	Advanced Encryption Standard
CAVP	Cryptographic Algorithm Validation Program
CBC	Cipher Block Chaining
CCM	Counter with Cipher Block Chaining-Message Authentication Code
CFB	Cipher Feedback
CMAC	Cipher-based Message Authentication Code
CMVP	Cryptographic Module Validation Program
CPACF	Central Processor Assist for Cryptographic Function
CSP	Critical Security Parameter
CTR	Counter Mode
DES	Data Encryption Standard
DF	Derivation Function
DRBG	Deterministic Random Bit Generator
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
FIPS	Federal Information Processing Standards Publication
GCM	Galois Counter Mode
HMAC	Hash Message Authentication Code
KAS	Key Agreement Schema
KAT	Known Answer Test
MAC	Message Authentication Code
NIST	National Institute of Science and Technology
OFB	Output Feedback
O/S	Operating System
PAA	Processor Algorithm Acceleration
PAI	Processor Algorithm Implementation
PR	Prediction Resistance
PSS	Probabilistic Signature Scheme
RNG	Random Number Generator
RSA	Rivest, Shamir, Addleman
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
XTS	XEX-based Tweaked-codebook mode with cipher text Stealing

Appendix B. References

FIPS140-3	**FIPS PUB 140-3 - Security Requirements For Cryptographic Modules** March 2019 https://doi.org/10.6028/NIST.FIPS.140-3
FIPS140-3_IG	**Implementation Guidance for FIPS PUB 140-3 and the Cryptographic Module Validation Program** October 2022 https://csrc.nist.gov/csrc/media/Projects/cryptographic-module-validation-program/documents/fips%20140-3/FIPS%20140-3%20IG.pdf
FIPS140-3_MM	**FIPS 140-3 Cryptographic Module Validation Program - Management Manual** April 2024 https://csrc.nist.gov/csrc/media/Projects/cryptographic-module-validation-program/documents/fips%20140-3/FIPS-140-3-CMVP%20Management%20Manual.pdf
FIPS180-4	**Secure Hash Standard (SHS)** March 2012 https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf
FIPS186-4	**Digital Signature Standard (DSS)** July 2013 https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf
FIPS197	**Advanced Encryption Standard** November 2001 https://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
FIPS198-1	**The Keyed Hash Message Authentication Code (HMAC)** July 2008 https://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf
FIPS202	**SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions** August 2015 https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf
OPENSSL-SP	**SUSE Linux Enterprise Server OpenSSL Cryptographic Module version 4.2 - FIPS 140-3 Non-Proprietary Security Policy** https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp4725.pdf
PKCS#1	**Public Key Cryptography Standards (PKCS) #1: RSA Cryptography** Specifications Version 2.1 February 2003 https://www.ietf.org/rfc/rfc3447.txt

SP800-38A	**NIST Special Publication 800-38A - Recommendation for Block Cipher Modes of Operation Methods and Techniques** December 2001 https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf
SP800-38A-add	**Addendum to NIST Special Publication 800-38A - Recommendation for Block Cipher Modes of Operation: Three Variants of Ciphertext Stealing for CBC Mode** October 2010 https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a-add.pdf
SP800-38B	**NIST Special Publication 800-38B - Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication** May 2005 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38B.pdf
SP800-38C	**NIST Special Publication 800-38C - Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality** May 2004 https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38c.pdf
SP800-38D	**NIST Special Publication 800-38D - Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC** November 2007 https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf
SP800-38E	**NIST Special Publication 800-38E - Recommendation for Block Cipher Modes of Operation: The XTS AES Mode for Confidentiality on Storage Devices** January 2010 https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38e.pdf
SP800-56Arev3	**NIST Special Publication 800-56A Revision 2 - Recommendation for Pair Wise Key Establishment Schemes Using Discrete Logarithm Cryptography** April 2018 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf
SP800-57rev5	**NIST Special Publication 800-57 Part 1 Revision 5 - Recommendation for Key Management Part 1: General** May 2020 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf

SP800-90Arev1	**NIST Special Publication 800-90A - Revision 1 - Recommendation for Random Number Generation Using Deterministic Random Bit Generators** June 2015 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf
SP800-90B	**NIST Special Publication 800-90B - Recommendation for the Entropy Sources Used for Random Bit Generation** January 2018 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90B.pdf
SP800-131Arev2	**NIST Special Publication 800-131A Revision 2 - Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths** March 2019 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar2.pdf
SP800-133rev2	**NIST Special Publication 800-133 Revision 2 - Recommendation for Cryptographic Key Generation** June 2020 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-133r2.pdf
SP800-135rev1	**NIST Special Publication 800-135 Revision 1 - Recommendation for Existing Application-Specific Key Derivation Functions** December 2011 https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-135r1.pdf
SP800-140B	**NIST Special Publication 800-140B - CMVP Security Policy Requirements** March 2020 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-140B.pdf