Rambus

FIPS 140-3 Non-Proprietary Security Policy

SafeZone FIPS SW Cryptographic Module

Software Version v2.0

Rambus Global Inc., Finnish branch

Sokerilinnantie 11 C

FI-02600 Espoo

Phone: +358 50 3560966

Rambus Inc.

North First Street, Suite 100

San Jose, CA 95134

United States of America

https://www.rambus.com/

2024-11-8

1 General

This document is the non-proprietary FIPS 140-3 Security Policy for the SafeZone FIPS SW Cryptographic Module version 2.0, hereafter referred to as the 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 140-3 [FIPS 140-3] for a Security Level 1 module.

It has a one-to-one mapping to the NIST Special Publication 800-140B [NIST 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.	FIPS 140-3 Section Title	Security Level
1	General	Level 1
2	Cryptographic module specification	Level 1
3	Cryptographic module interfaces	Level 1
4	Roles, services, and authentication	Level 1
5	Software/Firmware security	Level 1
6	Operational environment	Level 1
7	Physical security	N/A
8	Non-invasive security	N/A
9	Sensitive security parameter management	Level 1
10	Self-tests	Level 1
11	Life-cycle assurance	Level 1
12	Mitigation of other attacks	N/A

Table 1 - Security Levels

2 Cryptographic module specification

SafeZone FIPS SW Cryptographic Module is a FIPS 140-3 Level 1 validated software cryptographic module from Rambus. The module provides a set of commonly used cryptographic primitives by exposing a custom API for a wide range of applications, typically running on a general-purpose operating system. There are 4 different binary versions of the module to suit the target environments.

The identification string of the SafeZone FIPS SW Cryptographic Module can be acquired with the `FLS_LibDescription` function. The returned identification string is:

- `"SafeZone FL 2.0 NOHASH"`

The cryptographic module has been tested for validation on operational environments listed in the Table 2 below.

#	Operating System	Hardware Platform	Processor	PAA / Acceleration
1	Linux Ubuntu 20.04 LTS (X86 32-bit)	AAEON UP Core UPC-CHT01-A20-0464-A11	Intel® Atom™ x5-Z8350	AES-NI, Intel SHA extensions
2	Linux Ubuntu 20.04 LTS (X86 32-bit)	AAEON UP Core UPC-CHT01-A20-0464-A11	Intel® Atom™ x5-Z8350	-

3	Linux Ubuntu 20.04 LTS (X86 64-bit)	AAEON UP Core UPC-CHT01-A20-0464-A11	Intel® Atom™ x5-Z8350	AES-NI, Intel SHA extensions
4	Linux Ubuntu 20.04 LTS (X86 64-bit)	AAEON UP Core UPC-CHT01-A20-0464-A11	Intel® Atom™ x5-Z8350	-
5	Linux Ubuntu 20.04 LTS (ARMv8-a 64-bit)	Raspberry Pi 4	Broadcom BCM2711	NEON, Cryptography Extensions
6	Linux Ubuntu 20.04 LTS (ARMv8-a 64-bit)	Raspberry Pi 4	Broadcom BCM2711	-
7	Linux Ubuntu 20.04 LTS (ARMv7-a 32-bit)	Raspberry Pi 2	Broadcom BCM2836	NEON

Table 2 - Tested Operational Environments

In addition to the tested operational environments the module has been confirmed by the vendor to be operational on the following platforms.

#	Operating System	Hardware Platform
1	GNU/Linux Debian 11 (x86-64)	Gigabyte H97M-D3H with an Intel I7-4790K
2	GNU/Linux Debian 10 (aarch64)	Kirin 960, 4 Cortex A73 + 4 Cortex A53 Big.Little CPU
3	GNU/Linux Debian 9.13 (aarch64)	Rockship ROCK64 with a Rockchip RK3328

Table 3 - Vendor Affirmed Operational Environments

The Table 4 below lists the Approved Algorithms implemented by the module. The CAVP certs may list more algorithms than are actually utilized by the module. Only those listed below are used by the module.

CAVP Cert	Algorithm and Standard	Mode / Method	Description / Key Size(s) / Key Strength(s)	Use / Function
A2836	AES-CBC [NIST SP 800-38A]	AES-CBC	128, 192, 256 bits	Encryption and Decryption
A2836	AES-CCM [NIST SP 800-38C]	AES-CCM	128, 192, 256 bits	Authenticated Encryption and Decryption
A2836	AES-CMAC [NIST SP 800-38B]	AES-CMAC	128, 192, 256 bits	Message Authentication Code
A2836	AES-CTR [NIST SP 800-38A]	AES-CTR	128, 192, 256 bits	Encryption and Decryption
A2836	AES-ECB [NIST SP 800-38A]	AES-ECB	128, 192, 256 bits	Encryption and Decryption
A2836	AES-GCM [NIST SP 800-38D]	AES-GCM	128, 192, 256 bits	Authenticated Encryption and Decryption
A2836	AES-GMAC [NIST SP 800-38D]	AES-GMAC	128, 192, 256 bits	Message Authentication Code
A2836	AES-KW [NIST SP 800-38F]	AES-KW	128, 192, 256 bits	Key Wrapping and Unwrapping
A2836	AES-KWP [NIST SP 800-38F]	AES-KWP	128, 192, 256 bits	Key Wrapping and Unwrapping
A2836	AES-OFB [NIST SP 800-38A]	AES-OFB	128, 192, 256 bits	Encryption and Decryption

A2836	AES-XTS Testing Revision 2.0 [NIST SP 800-38E]	AES-XTS	2x128 bits (AES-XTS-128) 2x256 bits (AES-XTS-256)	Storage Encryption and Decryption
A2836	Counter DRBG [NIST SP 800-90A-r1]	Counter DRBG	AES-256 with df and no pr	Random Number Generation
A2836	DSA KeyGen [FIPS 186-4]	DSA KeyGen	(L,N) = (2048, 224), (2048, 256), (3072, 256)	Key Generation
A2836	DSA PQGGen [FIPS 186-4]	DSA PQGGen	(L,N) = (2048, 224), (2048, 256), (3072, 256) Hash algorithms: SHA2-224[1], SHA2-256, SHA2-384, SHA2-512	Domain Parameter Generation
A2836	DSA PQGVer [FIPS 186-4]	DSA PQGVer	(L,N) = (2048, 224), (2048, 256), (3072, 256) Hash algorithms: SHA-1[2], SHA2-224[3], SHA2-256, SHA2-383, SHA2-512	Domain Parameter Verification
A2836	DSA SigGen [FIPS 186-4]	DSA SigGen	(L,N) = (2048, 224), (2048, 256), (3072, 256) Hash algorithms: SHA2-224, SHA2-256, SHA2-384, SHA2-512	Digital Signatures
A2836	DSA SigVer [FIPS 186-4]	DSA SigVer	(L,N) = (1024, 160), (2048, 224), (2048, 256), (3072, 256) Hash algorithms: SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512.	Digital Signatures
A2836	ECDSA KeyGen [FIPS 186-4]	ECDSA KeyGen	NIST P-224, P-256, P-384, P-521 curves	Key Generation
A2836	ECDSA SigGen [FIPS 186-4]	ECDSA SigGen	NIST P-224, P-256, P-384, P-521 curves Hash algorithms: SHA2-224, SHA2-256, SHA2-384, SHA2-512	Digital Signatures
A2836	ECDSA SigVer [FIPS 186-4]	ECDSA SigVer	NIST P-224, P-256, P-384, P-521	Digital Signatures

[1] (L,N) = (2048, 224) only.

[2] (L,N) = (1024, 160) only.

[3] (L, N) = (1024, 160) and (2048, 224) only.

			curves Hash algorithms: SHA2-224, SHA2-256, SHA2-384, SHA2-512	
A2836	HMAC-SHA-1 [FIPS 198-1]	HMAC-SHA-1	Key Length: 112-512 bits MAC Length: 80, 96, 160 bits	Message Authentication Code
A2836	HMAC-SHA2-224 [FIPS 198-1]	HMAC-SHA2-224	Key Length: 112-512 bits MAC Length: 224 bits	Message Authentication Code
A2836	HMAC-SHA2-256 [FIPS 198-1]	HMAC-SHA2-256	Key Length: 112-512 bits MAC Length: 128, 256 bits	Message Authentication Code
A2836	HMAC-SHA2-384 [FIPS 198-1]	HMAC-SHA2-384	Key Length: 112-512 bits MAC Length: 192, 384 bits	Message Authentication Code
A2836	HMAC-SHA2-512 [FIPS 198-1]	HMAC-SHA2-512	Key Length: 112-512 bits MAC Length: 256, 512 bits	Message Authentication Code
A2836	KAS-ECC CDH-Component (CVL) [NIST SP 800-56A-r3]	KAS-ECC CDH-Component	ephemeralUnified NIST P-224, P-256, P-384 and P- 521 curves	Key Agreement Primitives
A2836	KAS-ECC-SSC [NIST SP 800-56A-r3]	KAS-ECC-SSC	ephemeralUnified NIST P-224, P-256, P-384 and P- 521 curves	Key Agreement Primitives
A2836	KAS-FFC-SSC [NIST SP 800-56A-r3]	KAS-FFC-SSC	dhEphem 2048-8192 bit modular Diffie-Hellman groups; including FFDHE2048, FFDHE3072, FFDHE4096, FFDHE6144, FFDHE8192, MODP2048, MODP3072, MODP4096, MODP6144, MODP8192, FIPS 186-type FFC parameter-size sets FB and FC	Key Agreement Primitives
A2836	KDA HKDF [NIST SP 800-56C-r2]	KDA HKDF	HMAC Algorithm: SHA2-224, SHA2-256, SHA2-384, SHA2-512	Key Derivation - Generic
A2836	KDA TwoStep [NIST SP 800-56C-r2]	KDA TwoStep	Two-Step Key Derivation with SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512 or	Key Derivation – Generic

			AES- CMAC; counter and feedback modes	
A2836	KDF IKEv1 (CVL) [NIST SP 800-135-r1]	KDF IKEv1	Hash Algorithm: SHA-1, SHA2-224, SHA2-384	Key Derivation – Application Specific
A2836	KDF IKEv2 (CVL) [NIST SP 800-135-r1]	KDF IKEv2	Hash Algorithm: SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	Key Derivation – Application Specific
A2836	KDF SP800-108 [NIST SP 800-108-r1]	KDF SP800-108	112-512 bits SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, AES-CMAC; counter, feedback and double pipeline modes	Key Derivation – Generic
A2836	KDF SRTP (CVL) [NIST SP 800-135-r1]	KDF SRTP	AES Key Length: 128, 192, 256	Key Derivation – Application Specific
A2836	KTS-IFC [NIST SP 800-56B-r2]	KTS-IFC	KTS-OAEP-basic Modulo: 2048, 3072, 4096 bits. Hashes: SHA2-224, SHA2-256, SHA2-384, SHA2-512	Key Encapsulation and Un-encapsulation
A2836	PBKDF [NIST SP 800-132]	PBKDF	SHA-1, SHA2-256	Key Derivation – Application Specific
A2836	RSA KeyGen [FIPS 186-4]	RSA KeyGen	2048, 3072, 4096 bits	Key Generation
A2836	RSA SigGen [FIPS 186-4]	RSA SigGen	RSASSA-PKCS1-v1.5 and RSASSA-PSS with SHA2-224, SHA2-256, SHA2-384, SHA2-512 and 2048, 3072, 4096 bit modulus	Digital Signatures
A2836	RSA SigVer [FIPS 186-4]	RSA SigVer	RSASSA-PKCS1-v1.5 and RSASSA-PSS with SHA-1[4], SHA2-224, SHA2-256, SHA2-384, SHA2-512 and 1024[4], 2048, 3072, 4096 bit modulus	Digital Signatures
A2836	SHA-1 [FIPS 180-4]	SHA-1	SHA-1	Hash Function Not allowed for

				signature generation. Allowed for legacy use[4] for digital signature verification. Allowed for non-digital-signature applications.
A2836	SHA2-224 [FIPS 180-4]	SHA2-224	SHA2-224	Hash Function
A2836	SHA2-256 [FIPS 180-4]	SHA2-256	SHA2-256	Hash Function
A2836	SHA2-384 [FIPS 180-4]	SHA2-384	SHA2-384	Hash Function
A2836	SHA2-512 [FIPS 180-4]	SHA2-512	SHA2-512	Hash Function
A2836	SHA3-224 [FIPS 202]	SHA3-224	SHA3-224	Hash Function
A2836	SHA3-256 [FIPS 202]	SHA3-256	SHA3-256	Hash Function
A2836	SHA3-384 [FIPS 202]	SHA3-384	SHA3-384	Hash Function
A2836	SHA3-512 [FIPS 202]	SHA3-512	SHA3-512	Hash Function
A2836	SHAKE-128 [FIPS 202]	SHAKE-128	128 bits	Extensible Output Function
A2836	SHAKE-256 [FIPS 202]	SHAKE-256	256 bits	Extensible Output Function
A2836	TDES-CBC [NIST SP 800-67-r2]	TDES-CBC	192 bits	Decryption[4]
A2836	TDES-ECB [NIST SP 800-67-r2]	TDES-ECB	192 bits	Decryption[4]
A2836	TLS v1.2 KDF RFC7627 (CVL) [NIST SP 800-135-r1]	TLS v1.2 KDF	Hash Algorithm: SHA2-256, SHA2-384, SHA2-512	Key Derivation – Application Specific
A2890	SHA3-256 [FIPS 202]	SHA3-256	N/A	Vetted conditioner
AES-KW A2836	AES-KW (KTS) [NIST SP 800-38F]	SP 800-38F. KTS (key wrapping and unwrapping) per IG D.G.	128, 192, and 256-bit keys providing 128, 192, or 256 bits of encryption strength	Key Transport
AES-KWP A2836	AES-KWP (KTS) [NIST SP 800-38F]	SP 800-38F. KTS (key wrapping and unwrapping) per IG D.G.	128, 192, and 256-bit keys providing 128, 192, or 256 bits of encryption strength	Key Transport
KTS-IFC A2836	KTS-IFC (KTS) [NIST SP 800-56B-r2]	SP 800-56Brev2. KTS-IFC (key encapsulation and un-encapsulation) per IG D.G.	2048, 3072, and 4096-bit modulus providing 112, 128, or 152 bits of encryption strength	Key Transport

[4] Legacy usage only. These legacy algorithms can only be used on data that was generated prior to the Legacy Date specified in FIPS 140-3 IG C.M

N/A	ENT (NP) [NIST SP 800-90B]	N/A	N/A	JitterEntropy
Vendor Affirmed	CKG [NIST SP 800-133r2]	Sections 5.1, 5.2 and 6.1	Cryptographic Key Generation; SP 800-133 and IG D.H (symmetric keys and asymmetric seeds).	Key Generation for symmetric and asymmetric keys using unmodified output from the DRBG.

Table 4 – Approved Algorithms

The module does not implement Non-Approved Algorithms Allowed in the Approved Mode of Operation or Non-Approved Algorithms Allowed in the approved Mode of Operation with No Security Claimed.

The Table 5 below lists the Non-approved Algorithms Not Allowed in the Approved Mode of Operation implemented by the module. The module restricts these algorithms to only be available as part of the non-approved mode.

Algorithm	Caveat	Use / Function
AES-KEY WRAP	AES-128, AES-192, AES-256 key wrapping superseded by the methods in SP 800-38F	Key Wrapping
Brainpool	All services applying Brainpool standard curves (P224r1, P256r1, P384r1, P512r1) are non-approved	Key Establishment
ChaCha20-Poly1305	ChaCha20-Poly1305 is not a service in NIST specifications, thus not approved	Symmetric encryption and decryption
DSA Key Pair Generation	P=1024/N=160 is non-approved since security strength provided is less than 112	Digital Signatures
DSA Signature Generation	P=1024/N=160 is non-approved since security strength provided is less than 112	Digital Signatures
ECC Diffie-Hellman	NIST P-192 curve is non-approved since security strength provided is less than 112	Key Establishment
ECDSA Key Pair Generation	NIST P-192 curve is non-approved since security strength provided is less than 112	Digital Signatures/Key Establishment
ECDSA Signature Generation	NIST P-192 curve is non-approved since security strength provided is less than 112	Digital Signatures
HMAC	80-104 bit keys are non-approved since security strength provided is less than 112	Message Authentication Code
KDF NIST SP 800-108	80-104 bit keys are non-approved since security strength provided is less than 112	Key Derivation
KTS (KEM NIST SP 800-56B)	RSA-KEM-KWS-basic key wrapping scheme is non-approved since not compliant with SP 800-56Brev2	Key Transport
KTS (OAEP NIST SP 800-56B)	1024 and 1536 bit keys are non-approved for RSA-OAEP scheme key wrapping	Key Transport
MD5	MD5 is non-approved except for the use in conjunction with SHA-1 in the TLS 1.0 / 1.1 KDF	Message Digest
RSA Encryption (PKCS #1 v1.5)	Non-approved since not compliant with SP 800-56Brev2	Key Wrapping
RSA Key Pair Generation	1024 and 1536 bit keys are non-approved since security strength provided is less than 112	Digital Signatures
RSA Private Key Primitives (NIST SP 800-56B)	Non-approved since not compliant with SP 800-56Brev2, including decryption primitives and signature generation primitives	Key Transport
RSA Public Key Primitives (NIST SP 800-56B)	Non-approved since not compliant with SP 800-56Brev2, including encryption primitives and signature verification primitives	Key Transport
RSA Signature	1024 and 1536 bit keys are non-approved since	Digital Signatures

Generation	security strength provided is less than 112	
RSA Signature Validation	1536 bit key is non-approved; legacy support for FIPS 186-2	Digital Signatures
TLS1.0/1.1 KDF NIST SP 800-135rev1	Deprecated in favor of RFC7627 extended master secret computation	Key Derivation
Triple-DES Encryption	Deprecated by the end of 2023	Symmetric encryption
X25519 Key Agreement	X25519 key agreement services are non-approved since not included in SP 800-56Arev3	Key Establishment

Table 5 -Non-Approved Algorithms Not Allowed in the Approved Mode of Operation

The Finite State Model (FSM) of the module is provided in a separate document *SafeZone-FIPS-Lib-2.0-FSM* distributed with this security policy.

The cryptographic boundary of the module is defined in the Figure 1 below.

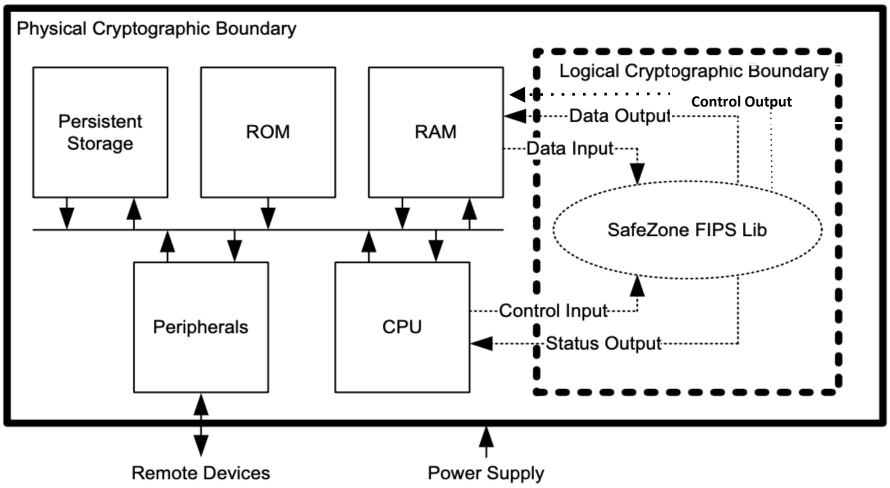

Figure 1 - Cryptographic Boundary

2.1 Approved Mode of Operation

By default, the module is in Approved mode once initialized and does not require any special initialization. The module will remain in approved mode of operation and the operator must avoid using any non-approved services. Any use of non-approved services (as determined by the indicator) will move module into the non-approved mode of operation. Any keys generated using non-approved mode or services must not be used in the approved mode of operation. The module needs to be re-initialized in order to move back into the approved mode of operation.

3 Cryptographic module interfaces

As a software-only module, SafeZone FIPS SW Cryptographic Module provides a C-programming language API for invocation of the FIPS 140-3 approved cryptographic functions.

The functions shall be called by the application which assumes the operator role during application execution. The API with input parameters, output parameters, and function return values, defines the four FIPS 140-3 logical interfaces: data input, data output, control input and status output.

Physical port	Logical Interface	API
N/A - API	Data Input	The data read from memory area(s) provided to the invoked function via parameters that point to the memory area(s).
N/A - API	Control Input	The API function invoked and function parameters designated as control inputs.
N/A - API	Data Output	The data written to memory area(s) provided to the invoked function via parameters that point to the memory area(s).
N/A - API	Status Output	The return value/data of the invoked API function.

Table 6. Ports and Interfaces

4 Roles, services, and authentication

The SafeZone FIPS SW Cryptographic Module supports the *Crypto Officer (CO)* and *User (U)* roles. The operator of the module will assume one of these two roles. Only one role may be active at a time. The Crypto Officer role is assumed implicitly upon module installation, uninstallation, initialization, zeroization, and power-up self-testing. If initialization and self-testing are successful, a transition to the User role is allowed and the User will be able to use all keys and cryptographic operations provided by the module, and to create any CSPs (except Trusted Root Key CSPs which may only be created in the Crypto Officer role).

The four unique run-time services given only to the Crypto Officer role are the ability to initialize the module, to set-up key material for Trusted Root Key CSP(s), to modify the entropy source, and to switch to the User role to perform any activities allowed for the User role. The SafeZone FIPS SW Cryptographic Module does not support concurrent operators.

The module does not authenticate the User or the Crypto Officer role.

The roles, services, and the API are described in the table below.

Role	Service	Input	Output
CO, U	Get module information	`FLS_StaticConfig()` `FL_IntactID()`	Build configuration, identifiers

CO, U	Get/set information on current module configuration	`FLS_RuntimeConfigSetProperty,`
		`FLS_RuntimeConfigGetProperty`
		`Runrime configuration value`
			FLR_OK (0), error or current runtime value

CO, U	Get crypto module status	`FLS_LibStatus()`	Module status
CO, U	Get crypto module version	`FLS_LibVersion()`	Module version
CO, U	Get crypto module description	`FLS_LibDescription()`	Module description
CO, U	Get status of the asset store	`FLS_AssetStoreStatus()`	Asset store memory status
CO, U	Perform module self-tests	`FLS_LibSelfTest()`	FLR_OK (0) or error
CO	Initialize the module	`FLS_LibInit()`	FLR_OK (0) or error

CO, U	Reset module state	`FLS_LibUnInit()`	FLR_OK (0) or error
CO	Enter User role	`FLS_LibEnterUserRole()`	FLR_OK (0) or error
CO, U	Erase data from memory	`FLS_Erase()` Memory area to be erased	FLR_OK (0) or error
CO, U	Erase asset	`FLS_EraseAsset()` Asset to be erased	FLR_OK (0) or error
CO	Install entropy source	`FL_RbgInstall EntropySource() FLS_RbgRequest SecurityStrength()` Entropy source	FLR_OK (0) or error
CO	Create trusted root key	`FLS_RootKeyAllocateAndLoadValue()` Key material	FLR_OK (0) or error

CO, U	Create key asset	`FLS_AssetAllocateBasic()`
		`FLS_AssetAllocate()`
		`FLS_AssetAllocateAnd AssociateKeyExtra()`
		`FLS_AssetLoadValue()`
		`FLS_AssetLoadMultipart()`
		`FLS_AssetLoadMultipart ConvertBigInt()`
		`FLS_AssetPoke()`
		`FL_LocalAllocate()`
		`FL_LocalAllocateEx()`
		Key policy, key value
			FLR_OK (0) or error, Created asset

CO, U	Copy key asset	`FLS_AssetCopyValue()` Source asset	FLR_OK (0) or error, Target asset

CO, U	Delete key asset	`FLS_AssetFree()`
		`FLS_LocalFree()`
		Asset to be deleted
			FLR_OK (0) or error

CO, U	Examine key asset policy, size	`FLS_AssetShow()`
		`FLS_AssetCheck()`
		Asset to be examined
			FLR_OK (0) or error, Key policy key size

CO, U	Get key asset value	`FLS_AssetPeek()` Asset to peek	FLR_OK (0) or error, Key value
CO, U	Generate key	`FLS_AssetLoadRandom()`	

CO, U	Encryption and decryption	`FLS_CipherInit()`
		`FLS_CipherContinue()`
		`FLS_CipherFinish()`
		Key asset, crypto parameters, plaintext/ciphertext
			FLR_OK (0) or error, Plaintext / ciphertext

CO, U	Authenticated encryption and decryption	`FLS_CryptAuthInit()`
		`FLS_CryptGcmAadContinue()`
		`FLS_CryptGcmAadFinish()`
		`FLS_CryptAuthContinue()`
		`FLS_EncryptAuthFinish()`
		`FLS_EncryptAuth PacketFinish()`
		`FLS_DecryptAuthFinish()`
		`FLS_EncryptAuth InitRandom()`
		`FLS_EncryptAuthInitDeterministic()`
		`FLS_CryptAuthInitTls13()`
		`FLS_EncryptAuthTls13()`
		`FLS_EncryptAuthFinishTls13()`
		`FLS_DecryptAuthTls13()`
		`FLS_DecryptAuthFinishTls13()`
		`FLS_EncryptAuthSrtp()`
		`FLS_DecryptAuthSrtp()`
			FLR_OK (0) or error, Plaintext / ciphertext

		FLS_EncryptAuthSrtcp() FLS_DecryptAuthSrtcp() Key asset, crypto parameters, additional authenticated data, plaintext/ciphertext	
CO, U	MAC generation	FLS_MacGenerateInit() FLS_MacGenerateContinue() FLS_MacGenerateFinish() Key asset, crypto parameters, input data	FLR_OK (0) or error, MAC
CO, U	MAC verification	FLS_MacVerifyInit() FLS_MacVerifyContinue() FLS_MacVerifyFinish() Key asset, crypto parameters, input data	FLR_OK (0) or error, Verification status
CO, U	Random number generation	FLS_RbgGenerateRandom() Random data generation parameters	FLR_OK (0) or error, Random data
CO, U	Random number generator reseeding	FLS_RbgReseed() Seed	FLR_OK (0) or error,
CO, U	Key derivation	FLS_KeyDeriveKdk() Key asset, derivation parameters	FLR_OK (0) or error, Derived key
CO, U	TLS v1.2 key derivation	FLS_DeriveTlsPrf() FLS_KeyDeriveKdk() Key asset, derivation parameters	FLR_OK (0) or error, Derived key
CO, U	Key Derivation Through Extraction-then-expansion	FLS_HkdfExtract() FLS_HkdfExpandAsset() FLS_HkdfExpand() FLS_Hkdf() FLS_KeyDeriveKdk() Derivation parameters, input key material	FLR_OK (0) or error, Derived key
CO, U	IKEv1 key derivation	FLS_IkePrfExtract() FLS_IKEv1ExtractSKEYID_DSA() FLS_IKEv1ExtractSKEYID_PSK() FLS_IKEv1ExtractSKEYID_PKE() FLS_IKEv1DeriveKeyingMaterial() Derivation parametes, input key material	FLR_OK (0) or error, Derived key
CO, U	IKEv2 key derivation	FLS_IkePrfExtract() FLS_IKEv2ExtractSKEYSEED() FLS_IKEv2DeriveDKM() FLS_IKEv2ExtractSKEYSEEDrekey() Derivation parametes, input key material	FLR_OK (0) or error, Derived key
CO, U	SRTP key derivation	FLS_SrtpKeyDerive() Key asset, derivation paramenters	FLR_OK (0) or error, Derived key
CO, U	AES key wrapping	FLS_AssetsWrapAes38F() FLS_AssetsUnwrapAes38F() Key asset, derivation parameters	FLR_OK (0) or error, Derived key
CO, U	AES data wrapping	FLS_CryptKw() Key asset, data to be wrapped	FLR_OK (0) or error, Wrapped data
CO, U	Trusted root key derivation	FLS_TrustedKdkDerive() FLS_TrustedKekdkDerive() Key asset, derivation parameters	FLR_OK (0) or error, Derived key
CO, U	Trusted KDK key derivation	FLS_TrustedKeyDerive() Key asset, derivation parameters	FLR_OK (0) or error, Derived key
CO, U	Trusted key wrapping	FLS_AssetWrapTrusted() FLS_AssetUnwrapTrusted() Key assets, wrapping parameters	FLR_OK (0) or error, Wrapped key
CO, U	PBKDF2 key derivation	FLS_KeyDerivePbkdf2() Password, key derivation parameters	FLR_OK (0) or error, Derived key
CO, U	DSA/Diffie-Hellman Domain Parameter verification	FLS_AssetCheck() Key asset to be examined	FLR_OK (0) or error
CO, U	Asymmetric key pair generation	FLS_AssetGenerateKeyPair() FLS_DH_KeyGen()	FLR_OK (0) or error, Generated

		Key generation parameters	key

CO, U	Signature generation	`FLS_HashSignFips186()`
		`FLS_HashSignPkcs1()`
		`FLS_HashSignPkcs1Pss()`
		Key asset, hash value
			FLR_OK (0) or error, Signature

CO, U	Signature verification	`FLS_HashVerifyFips186()`
		`FLS_HashVerifyRecoverPkcs1()`
		`FLS_HashVerifyPkcs1()`
		`FLS_HashVerifyPkcs1Pss()`
		Key asset, signature
			FLR_OK (0) or error, verification result

CO, U	RSA-OAEP key wrapping	`FLS_AssetsWrapRsaOaep()`
		`FLS_AssetsUnwrapRsaOaep()`
		Key asset, input data / wrapped key
			FLR_OK (0) or error, Wrapped key / unwrapped key asset

CO, U	Diffie-Hellman key agreement	`FLS_DeriveDh()`
		`FLS_DH_Derive()`
		Private and public values
			FLR_OK (0) or error, Derived key

CO, U	Elliptic Curve Diffie-Hellman key agreement	`FLS_DeriveDh()` Private and public values	FLR_OK (0) or error, Derived key

CO, U	Digest computation	`FLS_HashInit()`
		`FLS_HashContinue()`
		`FLS_HashFinish()`
		`FLS_HashSingle()`
		Input data / message
			FLR_OK (0) or error, Hash value

CO, U	Extensible Output Function	`FLS_HashSingle()` Input data / message	FLR_OK (0) or error, extended value
CO, U	Load precomputed digest	`FLS_LoadFinishedHash StateAlgo()` Digest parameters	FLR_OK (0) or error, Target asset

CO, U	Digest computation (non-approved)	`FLS_HashInit()`
		`FLS_HashContinue()`
		`FLS_HashFinish()`
		`FLS_HashSingle()`
		`Input data / message`
			FLR_OK (0) or error, Hash value

CO, U	Asymmetric key pair generation (non-approved)	`FLS_AssetGenerateKeyPair()`
		`FLS_DH_KeyGen()`
		`Key generation parameters`
			FLR_OK (0) or error, Generated key

CO, U	Signature generation (non-approved)	`FLS_HashSignFips186()`
		`FLS_HashSignPkcs1()`
		`FLS_HashSignPkcs1Pss()`
		`Key asset, hash value`
			FLR_OK (0) or error, Signature

CO, U	Signature verification (non-approved)	`FLS_HashVerifyFips186()`
		`FLS_HashVerifyRecoverPkcs1()`
		`FLS_HashVerifyPkcs1()`
		`FLS_HashVerifyPkcs1Pss()`
		`Key asset, signature`
			FLR_OK (0) or error, verification result

CO, U	Elliptic Curve Diffie-Hellman key agreement (non-approved)	`FLS_DeriveDh()` `Private and public values`	FLR_OK (0) or error, Derived key

CO, U	MAC generation (non-approved)	`FLS_MacGenerateInit()`
		`FLS_MacGenerateContinue()`
		`FLS_MacGenerateFinish()`
		`Key asset, crypto parameters, input data`
			FLR_OK (0) or error, MAC

CO, U	MAC verification (non-approved)	`FLS_MacVerifyInit()`
		`FLS_MacVerifyContinue()`
		`FLS_MacVerifyFinish()`
		`Key asset, crypto parameters, input data`
			FLR_OK (0) or error, Verification status

CO, U	RSA-KEM key wrapping (non-approved)	`FLS_AssetsWrapRsaKem()`
		`FLS_AssetsUnwrapRsaKem()`
		`Key asset, input data / wrapped`
			FLR_OK (0) or error, Wrapped key / unwrapped

		key	key asset

CO, U	RSA-OAEP key wrapping (non-approved)	`FLS_AssetsWrapRsaOaep()`
		`FLS_AssetsUnwrapRsaOaep()`
		`Key asset, input data / wrapped key`
			FLR_OK (0) or error, Wrapped key / unwrapped key asset

CO, U	KDK key derivation (non-approved)	`FLS_KeyDeriveKdk()` `Key asset, derivation parameters`	FLR_OK (0) or error, Derived key

CO, U	Trusted key wrapping (non-approved)	`FLS_AssetWrapTrusted()`
		`FLS_AssetUnwrapTrusted()`
		`Key assets, wrapping parameters`
			FLR_OK (0) or error, Wrapped key

CO, U	RSA-PKCS#1 v1.5 key wrapping (non-approved)	`FLS_AssetsWrapPkcs1v15()`
		`FLS_AssetsUnwrapPkcs1v15()`
		`Key asset, input data /wrapped key`
			FLR_OK (0) or error, Wrapped key / unwrapped key asset

CO, U	AES key wrapping (non-approved)	`FLS_AssetsWrapAes38F()`
		`FLS_AssetsUnwrapAes38F()`
		`Key asset, derivation parameters`
			FLR_OK (0) or error, Derived key

CO, U	Encryption and decryption (non-approved)	`FLS_CipherInit()`
		`FLS_CipherContinue()`
		`FLS_CipherFinish()`
		`Key asset, crypto parameters, plaintext/ciphertext`
			FLR_OK (0) or error, Plaintext / ciphertext

CO, U	X25519 (non-approved)	`FLS_X25519_KeyGen()`
		`FLS_X25519_Derive()`
		Private and public values
			FLR_OK (0) or error, Derived key

CO, U	TLS v1.0/1.1 key derivation (non-approved)	`FLS_DeriveTlsPrf()`
		`FLS_KeyDeriveKdk()`
		`Key asset, derivation parameters`
			FLR_OK (0) or error, Derived key

CO, U	RSA signature generation primitive (non-approved)	`FLS_Pkcs1RSASP1()` `Key asset, data to be signed`	FLR_OK (0) or error, Signature
CO, U	RSA signature verification primitive (non-approved)	`FLS_Pkcs1RSAVP1()` `Key asset, signature`	FLR_OK (0) or error, verification result
CO, U	RSA encryption primitive (non-approved)	`FLS_Pkcs1RSAEP()` `Key asset, plaintext`	FLR_OK (0) or error, Ciphertext
CO, U	RSA decryption primitive (non-approved)	`FLS_Pkcs1RSADP()` `Key asset, ciphertext`	FLR_OK (0) or error, Plaintext

Table 7. Roles, Service Commands, Input and Output

The approved services are described in the table below.

The indicator column maybe None or ARG. None is used for non-security functions, and they do not utilize the approved indicator. For services utilizing security functions the indicator is specified as ARG. In this case the user of the service or function passes a pointer as argument and the indicator value is returned to that pointer. A value of 0 means that the service is approved and any non-zero value means it is a non-approved service.

```
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 = Zeroize: The module zeroizes the SSP. The previous value is overwritten by zeroes and is
no longer accessible.
ARG = Approved indicator
```

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and / or SSPs	Indicator
Get module information	Return basic information of the module	N/A	None	CO, U	-	None
Get/set information on current module configuration	Return current module configuration	N/A	None	CO, U		None
Get crypto module status	Return state of the module	N/A	None	CO, U	-	None
Get crypto module version	Returns version of the module	N/A	None	CO, U	-	None
Get crypto module description	Returns description of the module	N/A	None	CO, U	-	None
Get status of the asset store	Returns the status of the asset store	N/A	None	CO, U	-	None
Perform module self-tests	Execute self-tests	N/A	None	CO, U	-	None
Initialize the module	Initializes the module	N/A	None	CO	-	None
Reset module state	Reset, zeroize and finalizes the module	N/A	None	CO, U	-	None
Enter User role	Enter regular user mode	N/A	None	CO	-	None
Erase data from memory	Erase data	N/A	Any	CO, U	Z	None
Erase asset	Erase asset	N/A	Any	CO, U	Z	None
Install entropy source	Install entropy source to be used	N/A	None	CO	-	None
Create trusted root key	Allocate and set data for new root key asset	KDF SP800-108	Trusted Root Key	CO	G	ARG
Create key asset	Setup key policy, allocate memory for key, and load key value	N/A	Any keys	CO, U	G, W	ARG
Copy key asset	Copies key value	N/A	Any keys	CO, U	R, W	ARG
Delete key asset	Deletes a key and zeroes the data	N/A	Any keys	CO, U	Z	ARG
Examine key asset policy, size	Get key size and check	N/A	Any keys	CO, U	R	ARG
Get key asset value	Get key value	N/A	Any keys	CO, U	R	ARG

Generate key	Generate random key	CKG (Section 6.1)	AES keys	CO, U	G, W	ARG
		TDES-CBC CKG (Section 6.1)	Triple-DES keys	CO, U	G, W	ARG
		CKG (Section 6.1)	Key Derivation keys	CO, U	G, W	ARG
		CKG (Section 6.1)	MAC keys	CO, U	G, W	ARG
Encryption and decryption	Service for data encryption and decryption	AES-ECB, AES-CBC, AES-CTR, AES-OFB, AES-XTS	AES keys	CO, U	E	ARG
		TDES-CBC, Triple-DES ECB	Triple-DES keys	CO, U	E	ARG
Authenticated encryption and decryption	Service for data encryption and decryption with added authentication	AES-CCM, AES-GCM	AES keys	CO, U	E	ARG
MAC generation	Service for MAC generation	AES-CMAC, AES-GMAC	AES keys	CO, U	E	ARG
		HMAC-SHA-1, HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512	MAC keys	CO, U	E	ARG
MAC verification	Service for MAC verification	AES-CMAC, AES-GMAC	AES keys	CO, U	E	ARG
		HMAC-SHA-1, HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512	MAC keys	CO, U	E	ARG
Random number generation	Generate random numbers by DRBG	Counter DRBG	DRBG CTR-256 entropy, DRBG CTR-256 seed, DRBG CTR-256 state: Key, DRBG CTR-256 state: V	CO, U	R	None
Random number generator reseeding	Force reseeding of random number generator	Counter DRBG	DRBG CTR-256 entropy, DRBG CTR-256 seed, DRBG CTR-256 state: Key, DRBG CTR-256 state: V	CO, U	W	None
Key derivation	Key derivation	KDF SP800-108	Key Derivation keys, KDF Derived Keys	CO, U	W	ARG
TLS v1.2 key derivation	Key derivation	TLS v1.2 KDF RFC7627	Key Derivation keys, KDF	CO, U	W	ARG

			Derived Keys			
Key Derivation Through Extraction-then-expansion	HKDF key derivation or key derivation with two steps	KDA HKDF, KDA TwoStep	Key Derivation keys, KDF Derived Keys	CO, U	W	ARG
IKEv1 key derivation	Key derivation for IKEv1	KDF IKEv1	Key Derivation keys, KDF Derived Keys	CO, U	W	ARG
IKEv2 key derivation	Key derivation for IKEv2	KDF IKEv2	Key Derivation keys, KDF Derived Keys	CO, U	W	ARG
SRTP key derivation	Key derivation for SRTP	KDF SRTP	Key Derivation keys, KDF Derived Keys	CO, U	W	ARG
AES key wrapping	Wrap AES key 38F	AES-KW (KTS), AES-KWP (KTS)	AES keys	CO, U	E	ARG
AES data wrapping	Wrap AES data 38F	AES-KW, AES-KWP	AES keys	CO, U	E	ARG
Trusted root key derivation	Derive root key	KDF SP800-108	Trusted Root Key, KDF Derived Keys	CO, U	W	ARG
Trusted KDK key derivation	KDK key derivation	KDF SP800-108	Key Derivation keys, KDF Derived Keys	CO, U	R, E	ARG
Trusted key wrapping	Wrap trusted key	KDF SP800-108 and AES-KW (KTS)/AES-KWP (KTS)K	Key Derivation keys, AES keys	CO, U	R, E	ARG
PBKDF2 key derivation	PBKDF2 key derivation	PBKDF	KDF Derived Keys, PBKDF password	CO, U	R, E	ARG
DSA/Diffie-Hellman Domain Parameter verification	DSA/Diffie-Hellman Domain Parameter verification	DSA PQGVer	DSA keys	CO, U	E	ARG
Asymmetric key pair generation	Generate asymmetric key pairs and DSA/Diffie-Hellman	RSA KeyGen	RSA keys	CO, U	E	ARG
		DSA KeyGen, DSA PQGGen	DSA keys	CO, U	E	ARG

	Domain Parameter	ECDSA KeyGen	ECDSA keys	CO, U	E	ARG
		KTS-IFC	RSA keys	CO, U	E	ARG
		KAS-FFC-SSC	DH keys	CO, U	E	ARG
		KAS-ECC-SSC	ECDH keys	CO, U	E	ARG
Signature generation	Generate signature	RSA SigGen	RSA keys	CO, U	E	ARG
		DSA SigGen	DSA keys	CO, U	E	ARG
		ECDSA SigGen	ECDSA keys	CO, U	E	ARG
Signature verification	Verify signature	RSA SigVer	RSA keys	CO, U	E	ARG
		DSA SigVer	DSA keys	CO, U	E	ARG
		ECDSA SigVer	ECDSA keys, Software Integrity Public Key	CO, U	E	ARG
RSA-OAEP key wrapping	RSA-OAEP key wrap	KTS-IFC (KTS)	RSA keys	CO, U	E	ARG
Diffie-Hellman key agreement	DH key agreement	KAS-FFC-SSC	DH keys, DH Shared secrets	CO, U	E	ARG
Elliptic Curve Diffie-Hellman key agreement	Elliptic Curve DH agreement	KAS-ECC-SSC	ECDH keys, ECDH Shared secrets	CO, U	E	ARG
Digest computation	Compute a digest	SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512	None	CO, U	-	ARG
Extensible Output Function	Compute an extended output	SHAKE-128, SHAKE-256	None	CO, U	-	ARG
Load precomputed digest	Load a precomputed digest	SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512	None	CO, U	-	ARG

Table 8. Approved Services

Service	Description	Algorithms Accessed	Roles	Indicator
Digest computation	Compute a digest with MD5	MD5	CO, U	ARG
Asymmetric key generation	Generate keys with unapproved parameters (e.g., key lengths and curves)	RSA Key Pair Generation, DSA Key Pair Generation, ECDSA Key Pair Generation, Brainpool	CO, U	ARG
Signature generation	Generate signatures with unapproved parameters (e.g., key lengths and curves)	RSA Signature Generation, DSA Signature	CO, U	ARG

		Generation, ECDSA Signature Generation, Brainpool		
Signature verification	Verify signature with unapproved key length	RSA Signature Validation	CO, U	ARG
Elliptic Curve Diffie-Hellman key agreement	Elliptic Curve DH agreement with unapproved curve	ECC Diffie-Hellman, Brainpool	CO, U	ARG
MAC generation	Service for MAC generation with unapproved key lengths	HMAC	CO, U	ARG
MAC verification	Service for MAC verification with unapproved key lengths	HMAC	CO, U	ARG
RSA-KEM key wrapping	RSA-KEM key wrap	KTS (KEM NIST SP 800-56B)	CO, U	ARG
RSA-OAEP key wrapping	RSA-OAEP key wrap	KTS (OAEP NIST SP 800-56B)	CO, U	ARG
KDK key derivation	KDK key derivation	KDF NIST SP 800-108	CO, U	ARG
Trusted key wrapping	Wrap trusted key	KDF NIST SP 800-108	CO, U	ARG
RSA-PKCS#1 v1.5 key wrapping	RSA-PKCS#1 key wrap	RSA Encryption (PKCS #1 v1.5)	CO, U	ARG
AES key wrapping	AES key wrapping that does fulfill NIST SP 800-38F	AES-KEY WRAP	CO, U	ARG
Encryption and decryption	Service for data encryption and decryption	ChaCha20-Poly1305, Triple-DES Encryption	CO, U	ARG
X25519	X25519 Key agreement	X25519 Key Agreement	CO, U	ARG
TLS v1.0/1.1 key derivation	Key derivation, SP 800-135 rev 1	TLS1.0/1.1 KDF NIST SP 800-135rev1	CO, U	ARG
RSA signature generation primitive	Generate RSA primitive only	RSA Private Key Primitives (NIST SP 800-56B)	CO, U	ARG
RSA signature verification primitive	RSA verification primitive only	RSA Public Key Primitives (NIST SP 800-56B)	CO, U	ARG
RSA encryption primitive	RSA encryption primitive only	RSA Public Key Primitives (NIST SP 800-56B)	CO, U	ARG
RSA decryption primitive	RSA decryption primitive only	RSA Private Key Primitives (NIST SP 800-56B)	CO, U	ARG

Table 9. Non-approved Services

5 Software/Firmware security

The SafeZone FIPS SW Cryptographic Module must be linked with an application to become executable. The software code of the module (`libsafezone-sw-fips.so` dynamically loadable library) is linked with an end application producing an executable application for the target platform. The application is installed in a platform-specific way, e.g., when purchased from an application store for the platform. In some cases, there is no need for installation, e.g., when a mobile equipment vendor includes the application with the equipment.

The SafeZone FIPS SW Cryptographic Module is loaded by loading an application that links the library statically. The SafeZone FIPS SW Cryptographic Module is initialized automatically upon loading. On some platforms the module is implemented as a dynamically loadable module. In this case, the module is loaded as needed by the dynamic linker.

The integrity check of the cryptographic module is described in section 10.

The SafeZone FIPS SW Cryptographic Module does not support operator authentication and thus does not require any authentication itself.

6 Operational environment

The operational environments are defined in Table 2 - Tested Operational Environments and Table 3 - Vendor Affirmed Operational Environments.

The module runs in a modifiable operating environment and uses modern operating systems, i.e., Linux. All processes spawned by the module are child processes of the module, and ownership of a process cannot be changed.

7 Physical security

The cryptographic module is software-only and does not have any physical security mechanisms.

8 Non-invasive security

The cryptographic module does not have non-invasive attack mitigation mechanisms.

9 Sensitive security parameters management

Key / SSP Name / Type	Strength	Security function and Cert	Generation	**Import/Export**	Establishment	Storage	Zeroisation	Use and related keys
AES keys	128, 192, 256 bits	AES-ECB, AES-CBC, AES-CTR, AES-OFB, AES-XTS, AES-GCM, AES-CCM, AES-CMAC, AES-GMAC,	KDF SP800-108, PBKDF, KDF IKEv1, KDF IKEv2, TLS v1.2 KDF RFC7627, KDF SRTP, KDA HKDF, KDA	Plaintext imported from TOEPP	N/A	Memory, plaintext	Asset deletion, module uninitialization	Symmetric encryption/ decryption,

		AES-KW, AES-KWP AES-KW (KTS), AES-KWP (KTS) (Cert. A2836)	TwoStep, CKG (SP 800-133r2 6.1)					
Triple-DES keys	192 bits	TDES-ECB Decryption, TDES-CBC Decryption (Cert. A2836)	KDF SP800-108, PBKDF, KDF IKEv1, KDF IKEv2, TLS v1.2 KDF RFC7627, KDF SRTP, KDA HKDF, KDA TwoStep, CKG (SP 800-133r2 6.1)	Plaintext imported from TOEPP	N/A	Memory, plaintext	Asset deletion, module uninitialization	Symmetric decryption
MAC keys	112-512 bits	HMAC-SHA-1, HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512 (Cert. A2836)	KDF SP800-108, PBKDF, KDF IKEv1, KDF IKEv2, TLS v1.2 KDF RFC7627, KDF SRTP, KDA HKDF, KDA TwoStep, CKG (SP 800-133r2 6.1)	Plaintext imported from TOEPP	N/A	Memory, plaintext	Asset deletion, module uninitialization	Generating and verifying HMAC authentication codes
RSA keys	1024, 2048, 3072, 4096 bits	RSA SigGen, RSA SigVer, RSA KeyGen, KTS-IFC, KTS-IFC (KTS) (Cert. A2836)	RSA KeyGen (FIPS 186-4), CKG (SP 800-133r2 5.1, FIPS 186-4)	Plaintext imported from TOEPP/Plaintext exported to RAM	N/A	Memory, plaintext	Asset deletion, module uninitialization	Digital signature, Key Transport
DSA keys	(L,N) = (2048, 224), (2048, 256), (3072, 256)	DSA SigGen, DSA SigVer, DSA KeyGen (Cert. A2836)	DSA KeyGen (FIPS 186-4), CKG (SP 800-133r2 5.1, FIPS 186-4)	Plaintext imported from TOEPP/Plaintext exported to RAM	N/A	Memory, plaintext	Asset deletion, module uninitialization	Digital signature
ECDSA keys	NIST P-224, P-256, P-384, P-521 curves	ECDSA SigGen, ECDSA SigVer, ECDSA KeyGen (Cert. A2836)	ECDSA KeyGen (FIPS 186-4), CKG (SP 800-133r2 5.1, FIPS 186-4)	Plaintext imported from TOEPP/Plaintext exported to RAM	N/A	Memory, plaintext	Asset deletion, module uninitialization	Digital signature
DH keys	2048-8192 bits	KAS-FFC-SSC (Cert. A2836)	NIST SP 800-56Ar3 key pair generation, CKG (SP 800-133r2 5.2,	Plaintext imported from TOEPP/Plaintext exported	N/A	Memory, plaintext	Asset deletion, module uninitialization	Key agreement

			NIST SP 800-56Ar3)	to RAM				
ECDH keys	NIST P-224, P-256, P-384 and P-521 curves	KAS-ECC-SSC (Cert. A2836)	NIST SP 800-56Ar3 key pair generation, CKG (SP 800-133r2 5.2, NIST SP 800-56Ar3)	Plaintext imported from TOEPP/Plaintext exported to RAM	N/A	Memory, plaintext	Asset deletion, module uninitialization	Key agreement
DH Shared secrets	2048-8192 bits	KAS-FFC-SSC (Cert. A2836)	N/A	Plaintext exported to RAM	N/A	Memory, plaintext	Asset deletion, module uninitialization	Key agreement
ECDH Shared secrets	NIST P-224, P-256, P-384 and P-521 curves	KAS-ECC-SSC (Cert. A2836)	N/A	Plaintext exported to RAM	N/A	Memory, plaintext	Asset deletion, module uninitialization	Key agreement
Key Derivation keys	112-200 bits	KDF SP800-108, KDF IKEv1, KDF IKEv2, TLS v1.2 KDF RFC7627, KDF SRTP, KDA HKDF (Cert. A2836)	N/A	Plaintext imported from TOEPP	N/A	Memory, plaintext	Asset deletion, module uninitialization	Key derivation
KDF Derived keys	112-200 bits	KDF SP800-108, PBKDF, KDF IKEv1, KDF IKEv2, TLS v1.2 KDF RFC7627, KDF SRTP, KDA HKDF (Cert. A2836)	KDF SP800-108, PBKDF, KDF IKEv1, KDF IKEv2, TLS v1.2 KDF RFC7627, KDF SRTP, KDA HKDF	Plaintext exported to RAM	N/A	Memory, plaintext	Asset deletion, module uninitialization	Key derivation
DRBG CTR-256 entropy	256-1024 bits	Counter DRBG (Cert. A2836)	ENT (NP)	N/A	N/A	Memory, Plaintext	Asset deletion, module uninitialization	Entropy input materials
DRBG CTR-256 seed	256-1024 bits	Counter DRBG (Cert. A2836)	Counter DRBG	N/A	N/A	Memory, Plaintext	Asset deletion, module uninitialization	Entropy input materials
DRBG CTR-256 state: Key	256 bits	Counter DRBG (Cert. A2836)	Counter DRBG	N/A	N/A	Memory, Plaintext	Asset deletion, module uninitialization	Key for DRBG used for random number and key/key

								pair Generation purposes.
DRBG CTR-256 state: V	128 bits	Counter DRBG (Cert. A2836)	Counter DRBG	N/A	N/A	Memory, Plaintext	Asset deletion, module uninitialization	V value for DRBG used for random number and key/key pair generation purposes.
PBKDF password	varies (at least 12 characters)	PBKDF (Cert. A2836)	N/A	Plaintext imported from TOEPP	N/A	Memory, plaintext	Asset deletion, module uninitialization	Password or passphrase
Trusted Root Key	256 bits	KDF SP800-108 (Cert. A2836)	N/A	Plaintext imported from TOEPP	N/A	Memory, plaintext	Asset deletion, module uninitialization	Key used for deriving other keys as per NIST SP 800-108. Can only derive 'Trusted KDK' and 'Trusted KEKDK' keys.
Software Integrity Public Key (not SSP)	NIST P-224	ECDSA SigVer (Cert. A2836)	N/A	N/A	Embedded in the software, plaintext	Persistent storage, plaintext	none	Public key used by Power-on Software Integrity to ensure the integrity of the Cryptographic Module.

Table 10. SSPs

9.1 Random bit generators

The cryptographic module contains a Counter DRBG which uses AES-256 and derivation function. By default, the DRBG is seeded with CPU Time Jitter Based Non-Physical True Random Number Generator (JitterEntropy).

It is also possible that the crypto officer may install another entropy source to the cryptographic module. The installed entropy source must be NIST SP 800-90B and FIPS 140-3 compliant. Enough bits of entropy must be provided according to the need of cryptographic algorithms.

The RBG entropy sources are defined in the table below.

Entropy sources	Minimum number of bits of entropy	Details
JitterEntropy ENT (NP)	256 bits of entropy minimum required to seed Counter DRBG AES-256	Implemented by Stephan Mueller. It is an entropy source based on CPU execution time jitter. The entropy source has NIST SP 800-90B compliance. The version of the Jitterentropy library applied in this cryptographic module is 3.4.0. The full description and documentation of the CPU Jitter entropy source and Jitterentropy library can be found at https://www.chronox.de/jent.html JitterEntropy ENT has a vetted SHA3-256 conditioning component, each SHA3-256 output contains 255 bits of entropy (h_submitter = 0.333) The Counter DRBG reads 512 bits from JitterEntropy ENT for seeding each time when instantiating and reseeding. 2 samples of SHA3-256 output from JitterEntropy ENT are gathered for each seed, which contains 510 bits of entropy.

Table 11. Non-Deterministic Random Number Generation Specification

10 Self-tests

The SafeZone FIPS SW Cryptographic Module includes the following self-tests:

- Pre-operational self-tests:
    - Pre-operational software integrity test:
        - Integrity test by ECDSA signature verify with NIST P-224 and SHA2-224
- Conditional self-tests:
    - Conditional cryptographic algorithm self-tests (CAST):
        - KAT test for SHA-1
        - KAT test for SHA2-512
        - KAT test for SHA3-224
        - KAT test for HMAC-SHA2-256
        - KAT test for AES-CBC encryption (128-bit key)
        - KAT test for AES-CBC decryption (128-bit key)
        - KAT test for AES-CCM encryption (128-bit key)
        - KAT test for AES-CCM decryption (128-bit key)
        - KAT test for AES-GCM encryption (128-bit key)
        - KAT test for AES-GCM decryption (128-bit key)
        - KAT test for AES-XTS encryption (128-bit key strength)
        - KAT test for AES-XTS decryption (128-bit key strength)
        - KAT test for AES-CMAC, 192-bit key

- KAT test for TDES-CBC encryption (192-bit key)
- KAT test for TDES-CBC decryption (192-bit key)
- KAT for Counter DRBG AES-256 (instantiate, generate and reseed; includes AES-ECB mode encryption)
- KAT for RSA SigGen 2048-bit with SHA2-256 (PKCS #1 v1.5)
- KAT for RSA SigVer 2048-bit with SHA2-256 (PKCS #1 v1.5)
- KAT for DSA SigGen (P=2048/N=256 with SHA2-224)
- KAT for DSA SigVer (P=1024/N=160 with SHA2-224)
- KAT for ECDSA SigGen (NIST P-224 with SHA2-224)
- KAT for ECDSA SigVer (NIST P-224 with SHA2-224)
- KAT for KTS-IFC Key Encapsulation 2048-bit (RSA-OAEP)
- KAT for KTS-IFC Key Un-encapsulation 2048-bit (RSA-OAEP)
- KAT for KAS-FFC-SSC (P=2048/N=224)
- KAT for KAS-ECC-SSC (NIST P-224)
- KAT for KDF SP800-108 (Counter Mode)
- KAT for KDA HKDF (includes KDF SP800-108 Feedback Mode)
- KAT for KDF SP800-108 (Double Pipeline Mode)
- KAT for KDF IKEv1
- KAT for KDF IKEv2
- KAT for TLS v1.2 KDF RFC7627
- KAT for SP 800-90A-r1, Section 11.3 (Instantiate/Generate) health tests for Counter DRBG AES-256
- KAT for SP 800-90A-r1, Section 11.3 (Reseed) health test Counter DRBG AES-256
- Fault Detection Tests for SP 800-90B, Section 4 start-up and continuous health tests (RCT and APT) for JitterEntropy ENT (NP).

- Conditional pair-wise consistency tests:
  - Pair-wise consistency check for key pairs created for digital signature purposes (DSA, FIPS 186-4)
  - Pair-wise consistency check for key pairs created for digital signature purposes (ECDSA, FIPS 186-4)
  - Pair-wise consistency check for RSA key pairs created for digital signature (FIPS 186-4) or key transport (NIST SP 800-56B) purposes.
  - Pair-wise consistency check for key pairs created for key establishment purposes (FFC DH and ECC DH, NIST SP 800-56A rev3)

All CAST are performed before pre-operational self-tests.

The KAT's are done by performing memory comparing (C library function memcmp) between calculations with known inputs and the expected correct results matching the inputs.

Self-tests are invoked automatically upon loading the cryptographic module. The initialization function `FLS_LibInit` is executed automatically. Any error during the self-tests will result the module in an error state, from where only recovered by invoking the `FLS_LibUninit` or `FLS_LibInit` function.

The `FL_LibStatus` API function can be used to obtain the module status. It returns FL_STATUS_INIT when the module has not yet been initialized and FL_STATUS_ERROR when the module is in error state.

As it is recommended to self-test cryptographic components (like DRBG) frequently, the module provides the capability to invoke the self-tests manually (on demand) with the `FL_LibSelfTest` API function. The important difference between the manually invoked self-tests and the automatically invoked self-tests when initializing the module is that the manually invoked self-tests will not cause zeroization of the key material currently loaded in the module, providing the tests execute successfully.

In general, if a self-test fails, the module will transition to the error state and the

return value (status) of the invoked API function will be something other than FLR_OK, depending on the current situation.

Conditional self-tests for manual key entry and software/firmware load or bypass are not provided, as these are not applicable. Any error during the conditional self-tests will result in a module transition to the error state

The cryptographic module uses the ECDSA NIST P-224 signature of the module binary for the integrity tests with SHA2-224 as the hash function. The public part of the key is always included with the module. The private part is stored in Rambus version control system and signing of the module is performed automatically by Rambus build system.

Before running the integrity test, tests for the signature algorithms are executed. In case of failure in the integrity test the module will immediately transition to error state.

11 Life-cycle assurance

11.1 Version control

The SafeZone FIPS SW Cryptographic Module source code is maintained in a version control system (Mercurial). Changes are reviewed and automatically built and tested with continuous integration system (Jenkins).

11.2 CVE

There are no CVE’s which currently affect the module as the module is published.

11.3 User guidance for specific services

Some of the FIPS Publications or NIST Special Publications require that the Cryptographic Module Security Policy mentions important configuration items for those algorithms. The user of the module shall observe these rules.

11.3.1 NIST SP 800-108 Rev 1: Key Derivation Functions

All three key derivation functions, Counter Mode, Feedback Mode and Double-Pipeline Iteration Mode are supported.

11.3.2 NIST SP 800-56C Rev 2: Key-Derivation Methods in Key-Establishment Schemes

The SafeZone FIPS SW Cryptographic module provides hash and HMAC functions that can be used for One-Step Key Derivation as introduced in NIST SP 800-56C Rev 2. The module also offers Extraction-then-Expansion function that can be used for Two-Step Key Derivation as introduced in NIST SP 800-56C Rev 2. The Two-Step Key Derivation function uses HMAC with SHA-1/SHA224/SHA256/SHA384 or AES-CMAC and SHA512 and NIST SP 800-108 Key Derivation Function with Feedback Mode. The construct is compatible with some uses of RFC 5869.

The following rules the user of the functions for NIST SP 800-56C Rev 2 Key Derivation functions shall observe:

- Key derived using NIST SP 800-56C Rev 2 shall only be used as secret keying material — such as a symmetric key used for data encryption or message integrity, a secret initialization vector, or, perhaps, a key-derivation key that will be used to generate additional keying material.
- The derived keying material shall not be used as a key stream for a stream cipher.
- When using HMAC algorithm for key derivation, the algorithms require a key. This key corresponds to salt in NIST SP 800-56C Rev 2. If salt is to be omitted, use all-zero-byte key at exactly the bit length of the hash algorithm.
- HKDF expansion function always uses NIST SP 800-108 Feedback Mode Key Derivation Function with single byte counter. This is interoperable with RFC 5869.
- The two-part extraction and expansion operation always uses the same underlying hash function or AES-CMAC for both extraction and expansion.
- AES-CMAC can be used to generate keys up to 128 bit security. For higher security hash- or HMAC-based schemes shall be used.
- HMAC-SHA-1 and HMAC-SHA-2 functions can be used to generate keys with 112-512 bit strength. See table below for details.
- If HMAC is used for key derivation, salt can be up-to one hash input block.
- If AES-CMAC is used, the key extraction phase may use 128 bit, 192 bit or 256 bit salt, but the key-expansion step will always use AES-128-CMAC.
- The module does not support NIST SP 800-56C Rev 2 Single-Step Key Derivation.
- If the input for NIST SP 800-56C Key Derivation Function is a shared secret, the input must be destroyed after extraction (e.g., with `FLS_AssetFree`).
- Two-Step Key Derivation will make use of both salt and KDK.

The input attributes and security strength of generated keys follows this table:

Hash or MAC for service	Length of optional salt (in bits)	MAC algorithm for optional KDK	The length of optional KDK (in bits)	Security strength s supported (in bits)
(HMAC-)SHA-1	up-to 512	HMAC-SHA-1	160	$112 \leq s \leq 160$
(HMAC-)SHA-224	up-to 512	HMAC-SHA-224	224	$112 \leq s \leq 224$

(HMAC-)SHA-256	up-to 512	HMAC-SHA-256	256	$112 \leq s \leq 256$
(HMAC-)SHA-384	up-to 1024	HMAC-SHA-384	384	$112 \leq s \leq 384$
(HMAC-)SHA-512	up-to 1024	HMAC-SHA-512	512	$112 \leq s \leq 512$
AES-128-CMAC	128	AES-128-CMAC	128	$112 \leq s \leq 128$
AES-192-CMAC	192	AES-192-CMAC	128	$112 \leq s \leq 128$
AES-256-CMAC	256	AES-256-CMAC	128	$112 \leq s \leq 128$

11.3.3 HMAC-based Extract-and-Expand Key Derivation Function (HKDF)

The SafeZone FIPS SW Cryptographic module provides HMAC-based Key Derivation Function from RFC 5869, known as HKDF. This function is similar to NIST SP 800-56C Rev 2 Two-Step Key Derivation, but not the same.

11.3.4 NIST SP 800-132: PBKDF Function

The key derived using NIST SP 800-132 shall only be used for storage purposes. The options 1a, 1b, 2a and 2b presented in NIST SP 800-132 for deriving the DPK (Data Protection Key) from the MK (Master Key) are supported. The SafeZone FIPS Lib does not limit the length of the password used in NIST SP 800-132 PBKDF key derivation. The upper bound for the strength of passwords usually used is between 5 or 6 bits per character, which indicates the upper bound for the probability of the password being randomly guessed is 1 / (64 ^ *length_of_password*). To achieve security over 64 bits, the passwords must generally be longer than 12 characters.

With compliance to NIST SP 800-132 and the FIPS 140-3 Implementation Guidance D.N, these requirements and limits must be followed by user:

- There is no maximum length of salt used, but at least 128 bits (16 bytes) of salt value must be randomly generated.
- The iteration count shall be as large as possible. The iteration count used must be at least 1000 to meet the minimum requirements of NIST SP 800-132. However, often it is recommendable to use much larger iteration counts, such as 100000 or 1000000, when user-perceived performance is not critical.
- Resulting MK must be used in the way as one of the following options as stated in NIST SP 800-132:
    - Option 1a, the MK is used directly as the DPK.

- Option 1b, the MK is used as input to an approved KDF (NIST SP 800-108 or NIST SP 800-56C-r2) in order to derive the DPK.
- Option 2a, the MK is used to protect a randomly generated DPK, the DPK is protected with approved authenticated encryption algorithm (AES-GCM or AES-CCM) or approved authentication technique (HMAC or AES-GMAC/CMAC) and approved encryption algorithm (AES-ECB/CBC/CTR/OFB/XTS).
- Option 2b, the MK is used as input to an approved KDF (NIST SP 800-108 or NIST SP 800-56C-r2) in order to derive the key to protect a randomly generated DPK, the DPK is protected with approved authenticated encryption algorithm (AES-GCM or AES-CCM) or approved authentication technique (HMAC or AES-GMAC/CMAC) and approved encryption algorithm (AES-ECB/CBC/CTR/OFB/XTS).

11.3.5 NIST SP 800-38D: Galois/Counter Mode (GCM) and GMAC

The FIPS 140-3 Implementation Guidance C.H applies to AES-GCM and GMAC usage with this module. Scenario/technique 1, 2 and 3 in IG C.H are supported by this module.

With compliance to technique 1 in IG C.H, the module supports AES-GCM with IPSec and TLS v1.2, both must be initialized with `FLS_EncryptAuthInitDeterministic` function for encryption and with `FLS_CryptAuthInit` for decryption. The `FLS_CryptAuthInit` function is also used for subsequent encryption operations for operation sequences started with the `FLS_EncryptAuthInitDeterministic` function (In this case the input IV/Nonce must be NULL since IG C.H forbids using external IV for encryption).

With compliance to technique 2 or 3 in IG C.H, the operator must use the `FLS_EncryptAuthInitRandom` function if random IV generation (IG C.H Technique 2) is required, or in case of deterministic IV generation (IG C.H Technique 3), the `FLS_EncryptAuthInitDeterministic` function. It is not possible to use random IV generated externally.

The module supports AES-GCM with SRTP (RFC 7714). For SRTP, functions `FLS_EncryptAuthSrtp`, `FLS_EncryptAuthSrtcp` have been introduced. These functions provide equivalent functionality than `FLS_EncryptAuthInitDeterministic`, but work with SRTP protocols. SRTP IV consists of 32-bit field, SSRC (synchronization source), which acts like 32-bit Module Name of IG C.H Technique 3. SRTP uses 48-bit counter ROC || SEQ. This counter is incremented internal to the cryptographic module. The module will detect overflow of counter. It is the responsibility of the users to rekey upon counter overflow. In addition, SRTP uses 96-bit Encryption Salt that is XORed with other fields. For control purposes, SRTP has an additional protocol, SRTCP. SRTCP protocol is otherwise identical to SRTP, but it uses different keys, and IV format where ROC || SEQ is replaced by SRTCP index. SRTCP index is incremented internal to the cryptographic module. The module will allow only 2^31 packets to be produced with SRTCP prior rekeying.

- **Note**: If IV is generated internally in a deterministic manner, then in case a module's power is lost and then restored, the key used for the AES GCM encryption/decryption must be re-distributed.

11.3.6 NIST SP 800-38E: XTS Mode

The module supports XTS Mode for Confidentiality on Storage Devices. Both XTS-AES-128 (256 bit key) and XTS-AES-256 (512 bit key) are supported. The XTS-AES key is parsed as concatenation of two AES keys Key_1 and Key_2. As is explained in FIPS 140-3 Implementation Guidance C.I, it is required that Key_1 ≠ Key_2. If Key_1 = Key_2, attempt to perform XTS-AES encryption or decryption will fail. The XTS Mode is only approved for usage in storage applications.

11.3.7 NIST SP 800-133 Rev 2: Key Generation (CKG)

The module allows key generation and generates keys according to the following NIST SP 800-133-r2 sections: 5.1, 5.2, 6.1. Key generation will use NIST SP 800-90A Rev1 DRBG-CTR AES-256. The output of the approved DRBG is used unmodified when symmetric keys are generated. It is also used unmodified as random input for asymmetric key generation.

11.3.8 NIST SP 800-107 Rev 1: Truncated HMAC

The module supports truncation of HMAC results for all SHA-1 and SHA-2 family hash functions. These include e.g., HMAC-SHA-1-80, HMAC-SHA-1-96, HMAC-SHA-256-128, HMAC-SHA-384-192 and HMAC-SHA-512-256. Following guidance of NIST SP 800-107 Rev 1, it is not allowed to truncate HMAC to less than 32-bits. Therefore, minimum allowed mac output length argument for the `FLS_MacGenerateFinish` or `FLS_MacVerifyFinish` is 4.

11.3.9 NIST SP 800-56A Rev 3: Pair-Wise Key-Establishment Schemes (KAS-ECC and KAS-FFC)

The module provides Discrete Logarithm Cryptographic-based key agreements compliant with NIST SP 800-56A-r3 according to scenario 2 path (1) of FIPS 140-3 Implementation Guidance D.F. The KAS-ECC-SSC schemes provide between 112 and 256 bits of security strength. The KAS-FFC-SSC schemes provide between 112 and 200 bits of security strength.

11.4 Porting maintaining validation

SafeZone FIPS 140-3 product is typically delivered in binary format. The product in binary format is validated for platforms mentioned in the validation certificate. There is also another product available from Rambus: source code product. This package can be recompiled by the user for their target platform. However, if changes to the module are required, they need to be processed as a revalidation for FIPS 140-3.

When ported to an operational environment which is not listed on the validation certificate, no claims can be made as to the correct operation of the module and the security strengths of any keys generated by the module.

12 Mitigation of other attacks

The cryptographic module does not implement security mechanisms to mitigate other attacks.

Appendix A. Glossary and Abbreviations

AES	Advanced Encryption Standard
CAVP	Cryptographic Algorithm Validation Program
CBC	Cipher Block Chaining
CFB	Cipher Feedback
CMAC	Cipher-based Message Authentication Code
CMVP	Cryptographic Module Validation Program
CO	Crypto Officer
CTR	Counter Mode
DSA	Digital Signature Algorithm
DRBG	Deterministic Random Bit Generator
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
FIPS	Federal Information Processing Standards Publication
FSM	Finite State Model
GCM	Galois Counter Mode
HMAC	Hash Message Authentication Code
KAS	Key Agreement Scheme
KAT	Known Answer Test
KDF	Key Derivation Function
KW	AES Key Wrap
KWP	AES Key Wrap with Padding
MAC	Message Authentication Code
NIST	National Institute of Science and Technology
OFB	Output Feedback
PSS	Probabilistic Signature Scheme
RNG	Random Number Generator
RSA	Rivest, Shamir, Adleman
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
SSP	Sensitive Security Parameter
U	User

Appendix B. References

FIPS 140-3	FIPS PUB 140-3 – Security Requirements for Cryptographic Modules March 2021
FIPS 140-3 IG	Implementation Guidance for FIPS 140-3 and the Cryptographic Module Validation Program November 2021
FIPS 180-4	FIPS PUB 180-4 – Secure Hash Standard (SHS) August 2015
FIPS 186-4	FIPS PUB 186-4 – Digital Signature Standard (DSS) July 2013
FIPS 198-1	FIPS PUB 198-1 – The Keyed-Hash Message Authentication Code (HMAC) July 2008
FIPS 202	FIPS PUB 202 – SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions August 2015
ISO/IEC 24759	ISO/IEC 24759 – Information technology – Security techniques – Test requirements for cryptographic modules March 2017
NIST SP 800-38A	NIST Special Publication 800-38A – Recommendation for Block Cipher Modes of Operation: Methods and Techniques December 2001
NIST SP 800-38B	NIST Special Publication 800-38B – Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication May 2005
NIST SP 800-38C	NIST Special Publication 800-38C – Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality May 2004
NIST SP 800-38D	NIST Special Publication 800-38C – Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC November 2007

NIST SP 800-38E	NIST Special Publication 800-38E – Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices January 2010
NIST SP 800-38F	NIST Special Publication 800-38F – Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping December 2012
NIST SP 800-56A-r3	NIST Special Publication 800-56A Revision 3 – Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography April 2018
NIST SP 800-56B-r2	NIST Special Publication 800-56B Revision 2 – Recommendation for Pair-Wise Key Establishment Using Integer Factorization Cryptography March 2019
NIST SP 800-56C-r2	NIST Special Publication 800-56C Revision 2 – Recommendation for Key-Derivation Methods in Key-Establishment Schemes August 2020
NIST SP 800-67-r2	NIST Special Publication 800-67 Revision 2 – Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher November 2017
NIST SP 800-90A-r1	NIST Special Publication 800-90A Revision 1 – Recommendation for Random Number Generation Using Deterministic Random Bit Generators June 2015
NIST SP 800-108	NIST Special Publication 800-108 – Recommendation for Key Derivation Using Pseudorandom Functions (Revised) October 2009
NIST SP 800-132	NIST Special Publication 800-132 – Recommendation for PBKDF, Part 1: Storage Applications December 2010
NIST SP 800-133-r2	NIST Special Publication 800-133 Revision 2 – Recommendation for Cryptographic Key Generation December 2011
NIST SP 800-135-r1	NIST Special Publication 800-135 Revision 1 – Recommendation for Existing Application-Specific Key Derivation Functions December 2011

NIST SP 800-140B	NIST Special Publication 800-140B – CMVP Security Policy Requirements March 2020