Rambus Inc.

SafeZone FIPS SW Cryptographic Module

FIPS 140-3 Non-Proprietary Security Policy

Table of Contents

List of Tables

List of Figures

1 General

1.1 Overview

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** for a Security Level 1 module.

It has a one-to-one mapping to the **NIST SP 800-140Br1**.

1.2 Security Levels

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	N/A
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
	Overall Level	1

Table 1: Security Levels

1.3 Additional Information

SafeZone FIPS SW Cryptographic Module is part of Rambus' software security products.
Product URL: https://www.rambus.com/security/software-protocols/fips-crypto-library/

2 Cryptographic Module Specification

2.1 Description

Purpose and Use:

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"

Module Type: Software

Module Embodiment: Multi-Chip Standalone

Module Characteristics:

Cryptographic Boundary:

The cryptographic boundary of the module is shown in the Figure 1 below, along with the interfaces with the operational environment.

Tested Operational Environment's Physical Perimeter (TOEPP):

The TOEPP of the module is defined as the general-purpose computer on which the module is installed.

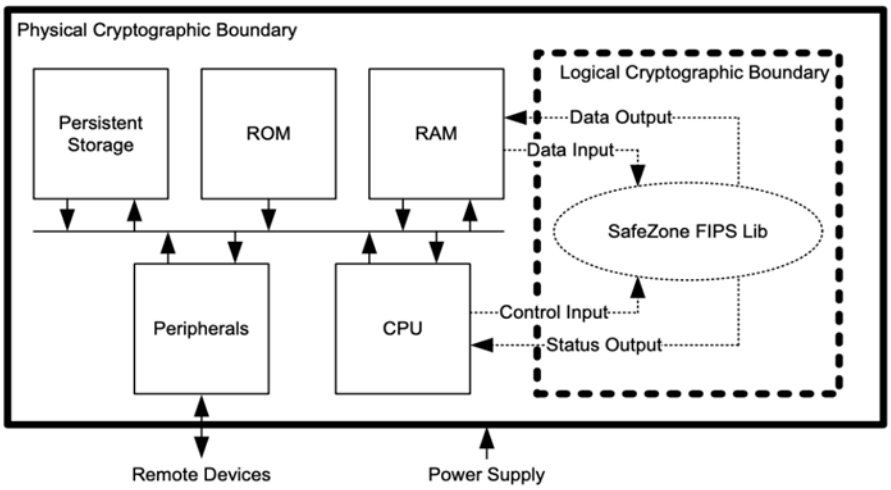

Figure 1: Block Diagram

2.2 Tested and Vendor Affirmed Module Version and Identification

Tested Module Identification – Hardware:

N/A for this module.

Tested Module Identification – Software, Firmware, Hybrid (Executable Code Sets):

Package or File Name	Software/ Firmware Version	Features	Integrity Test
libsafezone-sw-fips.so (ARMv7-a)	v2.0		ECDSA SigVer (FIPS 186-4)
libsafezone-sw-fips.so (ARMv8-a)	v2.0		ECDSA SigVer (FIPS 186-4)
libsafezone-sw-fips.so (x86)	v2.0		ECDSA SigVer (FIPS 186-4)
libsafezone-sw-fips.so (x86_64)	v2.0		ECDSA SigVer (FIPS 186-4)

Table 2: Tested Module Identification – Software, Firmware, Hybrid (Executable Code Sets)

Tested Module Identification – Hybrid Disjoint Hardware:

N/A for this module.

Tested Operational Environments - Software, Firmware, Hybrid:

Operating System	Hardware Platform	Processors	PAA/PAI	Hypervisor or Host OS	Version(s)
Linux Ubuntu 20.04 LTS (ARMv7-a 32-bit)	Raspberry Pi 2	Broadcom BCM2836	No		v2.0
Linux Ubuntu 20.04 LTS (ARMv7-a 32-bit)	Raspberry Pi 2	Broadcom BCM2836	Yes		v2.0
Linux Ubuntu 20.04 LTS (ARMv8-a 64-bit)	Raspberry Pi 4	Broadcom BCM2711	No		v2.0
Linux Ubuntu 20.04 LTS (ARMv8-a 64-bit)	Raspberry Pi 4	Broadcom BCM2711	Yes		v2.0
Linux Ubuntu 20.04 LTS (X86 32-bit)	AAEON UP Core UPC-CHT01-A20-0464-A11	Intel® AtomTM x5-Z8350	No		v2.0
Linux Ubuntu 20.04 LTS (X86 32-bit)	AAEON UP Core UPC-CHT01-A20-0464-A11	Intel® AtomTM x5-Z8350	Yes		v2.0
Linux Ubuntu 20.04 LTS (X86 64-bit)	AAEON UP Core UPC-CHT01-A20-0464-A11	Intel® AtomTM x5-Z8350	No		v2.0
Linux Ubuntu 20.04 LTS (X86 64-bit)	AAEON UP Core UPC-CHT01-A20-0464-A11	Intel® AtomTM x5-Z8350	Yes		v2.0

Table 3: Tested Operational Environments - Software, Firmware, Hybrid

The following PAA methods are used:

- AES NEON
- Cryptography Extensions
- AES-NI

Vendor-Affirmed Operational Environments - Software, Firmware, Hybrid:

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

Table 4: Vendor-Affirmed Operational Environments - Software, Firmware, Hybrid

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.

2.3 Excluded Components

N/A

2.4 Modes of Operation

Modes List and Description:

Mode Name	Description	Type	Status Indicator
Approved Mode	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.	Approved	Approved Indicator service returns 0 for service using approved security function.
Non-Approved Mode	Any use of non-approved services (as determined by the indicator) will move module into the non-approved mode of operation. The module needs to be re-initialized in order to move back into the approved mode of operation.	Non-Approved	Approved Indicator returns non-zero value for service.

Table 5: Modes List and Description

Mode Change Instructions and Status:

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.

2.5 Algorithms

Approved Algorithms:

Algorithm	CAVP Cert	Properties	Reference
AES-CBC	A2836	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-CCM	A2836	Key Length - 128, 192, 256 Tag Length - 112, 128, 64, 80, 96 IV Length - IV Length: 56-104 Increment 8 Payload Length - Payload Length: 0-256 Increment 8 AAD Length - AAD Length: 0-524288 Increment 8	SP 800-38C
AES-CMAC	A2836	Direction - Generation, Verification Key Length - 128, 192, 256 MAC Length - MAC Length: 128 Message Length - Message Length: 0-65536 Increment 8	SP 800-38B

Algorithm	CAVP Cert	Properties	Reference
AES-CTR	A2836	Direction - Decrypt, Encrypt Key Length - 128, 192, 256 Payload Length - Payload Length: 8-128 Increment 8 Supports Counter larger than maximum value - Yes Incremental Counter - Yes Counter Tests Performed - Yes	SP 800-38A
AES-ECB	A2836	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-GCM	A2836	Direction - Decrypt, Encrypt IV Generation - Internal IV Generation Mode - 8.2.2 Key Length - 128, 192, 256 Tag Length - 104, 112, 120, 128, 64, 96 IV Length - IV Length: 96 Payload Length - Payload Length: 0-65536 Increment 8 AAD Length - AAD Length: 0-65536 Increment 8	SP 800-38D
AES-GMAC	A2836	Direction - Decrypt, Encrypt IV Generation - Internal IV Generation Mode - 8.2.1 Key Length - 128, 192, 256 Tag Length - 104, 112, 120, 128, 64, 96 IV Length - IV Length: 96 AAD Length - AAD Length: 0-65536 Increment 8	SP 800-38D
AES-KW	A2836	Direction - Decrypt, Encrypt Cipher - Cipher Key Length - 128, 192, 256 Payload Length - Payload Length: 128-4096 Increment 64	SP 800-38F
AES-KWP	A2836	Direction - Decrypt, Encrypt Cipher - Cipher Key Length - 128, 192, 256 Payload Length - Payload Length: 8-4096 Increment 8	SP 800-38F
AES-OFB	A2836	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-XTS Testing Revision 2.0	A2836	Direction - Decrypt, Encrypt Key Length - 128, 256 Payload Length - Payload Length: 128-65536 Increment 8 Tweak Mode - Hex Data Unit Length Matches Payload Length - Yes	SP 800-38E
Counter DRBG	A2836	Prediction Resistance - No Supports Reseed - No, Yes Mode - AES-128, AES-256 Derivation Function Enabled - No, Yes Additional Input - Additional Input: 0, Additional Input: 0-256 Increment 64	SP 800-90A Rev. 1

Algorithm	CAVP Cert	Properties	Reference
		Entropy Input - Entropy Input: 256, Entropy Input: 256-1024 Increment 256 Nonce - Nonce: 0, Nonce: 256 Personalization String Length - Personalization String Length: 0, Personalization String Length: 0-256 Increment 256 Returned Bits - 512	
DSA KeyGen (FIPS186-4)	A2836	L - 2048, 3072 N - 224, 256	FIPS 186-4
DSA PQGGen (FIPS186-4)	A2836	P/Q Generation Methods - Probable G Generation Methods - Canonical L - 2048, 3072 N - 224, 256 Hash Algorithm - SHA2-224, SHA2-256, SHA2-384, SHA2-512	FIPS 186-4
DSA PQGVer (FIPS186-4)	A2836	P/Q Generation Methods - Probable G Generation Methods - Canonical L - 1024, 2048, 3072 N - 160, 224, 256 Hash Algorithm - SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	FIPS 186-4
DSA SigGen (FIPS186-4)	A2836	L - 2048, 3072 N - 224, 256 Hash Algorithm - SHA2-224, SHA2-256, SHA2-384, SHA2-512	FIPS 186-4
DSA SigVer (FIPS186-4)	A2836	L - 1024, 2048, 3072 N - 160, 224, 256 Hash Algorithm - SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	FIPS 186-4
ECDSA KeyGen (FIPS186-4)	A2836	Curve - P-224, P-256, P-384, P-521 Secret Generation Mode - Testing Candidates	FIPS 186-4
ECDSA SigGen (FIPS186-4)	A2836	Curve - P-224, P-256, P-384, P-521 Hash Algorithm - SHA2-224, SHA2-256, SHA2-384, SHA2-512	FIPS 186-4
ECDSA SigVer (FIPS186-4)	A2836	Curve - P-224, P-256, P-384, P-521 Hash Algorithm - SHA2-224, SHA2-256, SHA2-384, SHA2-512	FIPS 186-4
HMAC-SHA-1	A2836	MAC - MAC: 80-160 Increment 8 Key Length - Key Length: 112-512 Increment 8	FIPS 198-1
HMAC-SHA2-224	A2836	MAC - MAC: 80-224 Increment 8 Key Length - Key Length: 112-512 Increment 8	FIPS 198-1
HMAC-SHA2-256	A2836	MAC - MAC: 80-256 Increment 8 Key Length - Key Length: 112-512 Increment 8	FIPS 198-1
HMAC-SHA2-384	A2836	MAC - MAC: 80-384 Increment 8 Key Length - Key Length: 112-512 Increment 8	FIPS 198-1
HMAC-SHA2-512	A2836	MAC - MAC: 80-512 Increment 8 Key Length - Key Length: 112-512 Increment 8	FIPS 198-1

Algorithm	CAVP Cert	Properties	Reference
KAS-ECC-SSC Sp800-56Ar3	A2836	Domain Parameter Generation Methods - P-224, P-256, P-384, P-521 Scheme - ephemeralUnified - KAS Role - initiator, responder	SP 800-56A Rev. 3
KAS-FFC-SSC Sp800-56Ar3	A2836	Domain Parameter Generation Methods - FB, FC, ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, modp-2048, modp-3072, modp-4096, modp-6144, modp-8192 Scheme - dhEphem - KAS Role - initiator, responder	SP 800-56A Rev. 3
KDA HKDF SP800-56Cr2	A2836	Fixed Info Pattern - algorithmId||t||l||uPartyInfo||vPartyInfo Fixed Info Encoding - concatenation Derived Key Length - 2048 Shared Secret Length - Shared Secret Length: 224-8192 Increment 8 HMAC Algorithm - SHA2-224, SHA2-256, SHA2-384, SHA2-512 Perform Multiple Expansion Tests - No	SP 800-56C Rev. 2
KDA TwoStep SP800-56Cr2	A2836	MAC Salting Methods - default, random Fixed Info Pattern - algorithmId||t||l||uPartyInfo||vPartyInfo Fixed Info Encoding - concatenation KDF Mode - counter, feedback MAC Modes - CMAC-AES128, CMAC-AES192, CMAC-AES256, HMAC-SHA-1, HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512 Fixed Data Order - after fixed data, before fixed data Counter Lengths - 16, 24, 32, 8 The KDF supports an empty IV - No, Yes The KDF requires an empty IV - No, Yes Supported Lengths - Supported Lengths: 2048 Derived Key Length - 2048 Shared Secret Length - Shared Secret Length: 224-8192 Increment 8 Perform Multiple Expansion Tests - No	SP 800-56C Rev. 2
KDF IKEv1 (CVL)	A2836	Authentication Method - Public Key Encryption Initiator Nonce Length - Initiator Nonce Length: 64-2048 Increment 8 Responder Nonce Length - Responder Nonce Length: 64-2048 Increment 8 Preshared Key Length - Preshared Key Length: 8-224 Increment 8 Diffie-Hellman Shared Secret Length - Diffie-Hellman	SP 800-135 Rev. 1

Algorithm	CAVP Cert	Properties	Reference
		Shared Secret Length: 256-2048 Increment 8 Hash Algorithm - SHA-1, SHA2-224, SHA2-384	
KDF IKEv2 (CVL)	A2836	Initiator Nonce Length - Initiator Nonce Length: 2048 Responder Nonce Length - Responder Nonce Length: 2048 Diffie-Hellman Shared Secret Length - Diffie-Hellman Shared Secret Length: 2048 Derived Keying Material Length - Derived Keying Material Length: 3072 Hash Algorithm - SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	SP 800-135 Rev. 1
KDF SP800-108	A2836	KDF Mode - Counter, Double Pipeline Iteration, Feedback MAC Mode - CMAC-AES128, CMAC-AES192, CMAC-AES256, HMAC-SHA-1, HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512 Supported Lengths - Supported Lengths: 8-512 Increment 8 Fixed Data Order - After Fixed Data, Before Fixed Data Counter Length - 16, 24, 32, 8 Supports Empty IV - No, Yes Custom Key In Length - 0 Requires Empty IV - No	SP 800-108 Rev. 1
KDF SRTP (CVL)	A2836	AES Key Length - 128, 192, 256 Supports Empty KDR - No KDR Exponents - 1, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2, 20, 21, 22, 23, 24, 3, 4, 5, 6, 7, 8, 9	SP 800-135 Rev. 1
KTS-IFC	A2836	Function - keyPairGen, partialVal IUT ID - DEADDEAD Modulo - 2048, 3072, 4096 Key Generation Methods - rsakpg1-crt Fixed Public Exponent - 010001 Scheme - KTS-OAEP-basic - KAS Role - initiator, responder Key Transport Method - Hash Algorithms - SHA2-224, SHA2-256, SHA2-384, SHA2-512 Supports Null Associated Data - Yes Associated Data Encoding - concatenation Key Length - 512	SP 800-56B Rev. 2
PBKDF	A2836	Iteration Count - Iteration Count: 10-1000 Increment 1 HMAC Algorithm - SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512 Password Length - Password Length: 8-128 Increment 1	SP 800-132

Algorithm	CAVP Cert	Properties	Reference
		Salt Length - Salt Length: 128-4096 Increment 8 Key Data Length - Key Data Length: 112-4096 Increment 8	
RSA KeyGen (FIPS186-4)	A2836	Key Generation Mode - B.3.3, B.3.6 Modulo - 2048, 3072, 4096 Hash Algorithm - SHA2-256 Primality Tests - Table C.3 Info Generated By Server - No Public Exponent Mode - Fixed Fixed Public Exponent - 010001 Private Key Format - Standard	FIPS 186-4
RSA SigGen (FIPS186-4)	A2836	Signature Type - PKCS 1.5, PKCSPSS Modulo - 2048, 3072, 4096 Hash Pair - Hash Algorithm - SHA2-224	FIPS 186-4
RSA SigVer (FIPS186-4)	A2836	Signature Type - PKCS 1.5, PKCSPSS Modulo - 1024, 2048, 3072, 4096 Hash Pair - Hash Algorithm - SHA-1 Public Exponent Mode - Fixed Fixed Public Exponent - 010001	FIPS 186-4
Safe Primes Key Generation	A2836	Safe Prime Groups - ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, modp-2048, modp-3072, modp-4096, modp-6144, modp-8192	SP 800-56A Rev. 3
SHA-1	A2836	Message Length - Message Length: 0-65528 Increment 8	FIPS 180-4
SHA2-224	A2836	Message Length - Message Length: 0-65528 Increment 8	FIPS 180-4
SHA2-256	A2836	Message Length - Message Length: 0-65528 Increment 8	FIPS 180-4
SHA2-384	A2836	Message Length - Message Length: 0-65528 Increment 8	FIPS 180-4
SHA2-512	A2836	Message Length - Message Length: 0-65528 Increment 8	FIPS 180-4
SHA3-224	A2836	Message Length - Message Length: 0-65536 Increment 8	FIPS 202
SHA3-256	A2836	Message Length - Message Length: 0-65536 Increment 8	FIPS 202
SHA3-256	A2890	Message Length - Message Length: 0-65536 Increment 8	FIPS 202
SHA3-384	A2836	Message Length - Message Length: 0-65536 Increment 8	FIPS 202
SHA3-512	A2836	Message Length - Message Length: 0-65536 Increment 8	FIPS 202
SHAKE-128	A2836	Supports Bit-Oriented Messages - No Supports Empty Message - Yes Supports Bit-Oriented Output - No Output Length - Output Length: 16-65536 Increment 8	FIPS 202

Algorithm	CAVP Cert	Properties	Reference
SHAKE-256	A2836	Supports Bit-Oriented Messages - No Supports Empty Message - Yes Supports Bit-Oriented Output - No Output Length - Output Length: 16-65536 Increment 8	FIPS 202
TDES-CBC	A2836	Direction - Decrypt	SP 800-67 Rev. 2
TDES-ECB	A2836	Direction - Decrypt	SP 800-67 Rev. 2
TLS v1.2 KDF RFC7627 (CVL)	A2836	Hash Algorithm - SHA2-256, SHA2-384, SHA2-512	SP 800-135 Rev. 1

Table 6: Approved Algorithms

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

Vendor-Affirmed Algorithms:

Name	Properties	Implementation	Reference
CKG	Key Type:Symmetric and Asymmetric	N/A	Section 4, Example 1 where V is equivalent to a constant string of zeroes.
CKG (AES-XTS)	Key Type:Symmetric	N/A	Section 6.3, approved method 1

Table 7: Vendor-Affirmed Algorithms

Key Generation for symmetric and asymmetric keys using unmodified output from the DRBG.

Non-Approved, Allowed Algorithms:

N/A for this module.

The module does not implement Non-Approved Algorithms Allowed in the Approved Mode of Operation.

Non-Approved, Allowed Algorithms with No Security Claimed:

N/A for this module.

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

Non-Approved, Not Allowed Algorithms:

Name	Use and Function
AES-KEY WRAP	Key Wrapping
Brainpool	Key Establishment
ChaCha20-Poly1305	Symmetric encryption and decryption
DSA Key Pair Generation	Digital Signatures
DSA Signature Generation	Digital Signatures
ECC Diffie-Hellman	Key Establishment
ECDSA Key Pair Generation	Digital Signatures/Key Establishment
ECDSA Signature Generation	Digital Signatures
HMAC	Message Authentication Code
KDF NIST SP 800-108	Key Derivation
KTS (KEM NIST SP 800-56B)	Key Transport
KTS (OAEP NIST SP 800-56B)	Key Transport
MD5	Message Digest
RSA Encryption (PKCS #1 v1.5)	Key Wrapping
RSA Key Pair Generation	Digital Signatures
RSA Private Key Primitives (NIST SP 800-56B)	Key Transport
RSA Public Key Primitives (NIST SP 800-56B)	Key Transport
RSA Signature Generation	Digital Signatures
RSA Signature Validation	Digital Signatures
TLS1.0/1.1 KDF NIST SP 800-135rev1	Key Derivation
Triple-DES Encryption	Symmetric encryption
X25519 Key Agreement	Key Establishment

Table 8: Non-Approved, Not Allowed Algorithms

The Table above 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.

2.6 Security Function Implementations

Name	Type	Description	Properties	Algorithms
Block Ciphers (Legacy)	BC-UnAuth	Legacy Triple-DES decryption per FIPS 140-3 IG C.M.		TDES-CBC: (A2836) TDES-ECB: (A2836)
Block Ciphers (Unauthenticated)	BC-UnAuth CKG	Block Ciphers using unauthenticated encryption/decryption.		AES-CBC: (A2836) AES-CTR: (A2836) AES-ECB: (A2836) AES-OFB: (A2836) AES-XTS Testing Revision 2.0: (A2836) CKG (AES-

Name	Type	Description	Properties	Algorithms
				XTS): () Key Type: Symmetric
Block Ciphers AEAD (Authenticated)	BC-Auth	Block Ciphers using AEAD encryption/decryption.		AES-CCM: (A2836) AES-GCM: (A2836)
Block Ciphers AES KW/KWP (Authenticated)	BC-Auth	Block Ciphers using AES KW/KWP encryption/decryption.		AES-KW: (A2836) AES-KWP: (A2836)
Domain Parameter Generation (FFC)	AsymKeyPair-DomPar	Finite Field Cryptography domain parameter generation.		DSA PQGGen (FIPS186-4): (A2836)
Domain Parameter Verification (FFC)	AsymKeyPair-DomPar	Finite Field Cryptography domain parameter verification.		DSA PQGVer (FIPS186-4): (A2836)
Entropy Source	ENT-NP	JitterEntropy source.		SHA3-256: (A2890)
Hashing (SHA)	SHA	FIPS 180-4 and FIPS 202 secure hash algorithms.		SHA-1: (A2836) SHA2-224: (A2836) SHA2-256: (A2836) SHA2-384: (A2836) SHA2-512: (A2836) SHA3-224: (A2836) SHA3-256: (A2836) SHA3-384: (A2836) SHA3-512: (A2836)
Hashing (XOF)	XOF	FIPS 202 extensible output functions.		SHAKE-128: (A2836) SHAKE-256: (A2836)
KDF (ASKDF IKEv1)	KAS-135KDF	SP 800-135rev1 application specific key derivation for IKEv1.		KDF IKEv1: (A2836)
KDF (ASKDF IKEv2)	KAS-135KDF	SP 800-135rev1 application specific		KDF IKEv2: (A2836)

Name	Type	Description	Properties	Algorithms
		key derivation for IKEv2		
KDF (ASKDF SRTP)	KAS-135KDF	SP 800-135rev1 application specific key derivation for SRTP		KDF SRTP: (A2836)
KDF (ASKDF TLS v1.2)	KAS-135KDF	SP 800-135rev1 application specific key derivation for TLS v1.2.		TLS v1.2 KDF RFC7627: (A2836)
KDF (KBKDF)	KBKDF	SP 800-108rev1 key-based key derivation.		KDF SP800-108: (A2836)
KDF (KDA)	KAS-56CKDF	SP 800-56Crev2 key derivation.		KDA HKDF SP800-56Cr2: (A2836) KDA TwoStep SP800-56Cr2: (A2836)
KDF (PBKDF)	PBKDF	SP 800-132 password-based key derivation.		PBKDF: (A2836)
Key Generation (ECC)	AsymKeyPair-KeyGen CKG	Elliptic Curve Cryptography key generation.		CKG: () Key Type: Symmetric and Asymmetric ECDSA KeyGen (FIPS186-4): (A2836)
Key Generation (FFC)	AsymKeyPair-KeyGen CKG	Finite Field Cryptography key generation.		CKG: () Key Type: Symmetric and Asymmetric DSA KeyGen (FIPS186-4): (A2836) Safe Primes Key Generation: (A2836)
Key Generation (RSA)	AsymKeyPair-KeyGen CKG	RSA key generation.		CKG: () Key Type: Symmetric and Asymmetric RSA KeyGen

Name	Type	Description	Properties	Algorithms
				(FIPS186-4): (A2836)
Key Generation (Symmetric)	CKG	SP 800-133rev2 Symmetric Key Generation.		CKG: () Key Type: Symmetric and Asymmetric Counter DRBG: (A2836)
KTS-IFC	AsymKeyPair-Decap AsymKeyPair-Encap	SP 800-56Br2 key encapsulation and un-encapsulation.	Standard:SP 800-56Brev2; IG D.G: Method 1; Key Confirmation: No; Caveat: Key establishment methodology provides between 112 and 152 bits of security strength.	KTS-IFC: (A2836)
MAC Generation	MAC	Generation of message authentication codes.		AES-CMAC: (A2836) AES-GMAC: (A2836) HMAC-SHA-1: (A2836) HMAC-SHA2-224: (A2836) HMAC-SHA2-256: (A2836) HMAC-SHA2-384: (A2836) HMAC-SHA2-512: (A2836)
MAC Verification	MAC	Verification of message authentication codes.		AES-CMAC: (A2836) AES-GMAC: (A2836) HMAC-SHA-1: (A2836) HMAC-SHA2-224: (A2836) HMAC-SHA2-256: (A2836) HMAC-SHA2-

Name	Type	Description	Properties	Algorithms
				384: (A2836) HMAC-SHA2-512: (A2836)
Random Bit Generation	DRBG	Generate random bits using the DRBG.		Counter DRBG: (A2836)
Shared Secret Computation (ECC)	KAS-SSC	Elliptic Curve Cryptography shared secret computation.		KAS-ECC-SSC Sp800-56Ar3: (A2836)
Shared Secret Computation (FFC)	KAS-SSC	Finite Field Cryptography shared secret computation.		KAS-FFC-SSC Sp800-56Ar3: (A2836)
Signature Generation (DSA)	DigSig-SigGen	Generation of DSA signatures.		DSA SigGen (FIPS186-4): (A2836)
Signature Generation (ECDSA)	DigSig-SigGen	Generation of ECDSA signatures.		ECDSA SigGen (FIPS186-4): (A2836)
Signature Generation (RSA)	DigSig-SigGen	Generation of RSA signatures.		RSA SigGen (FIPS186-4): (A2836)
Signature Verification (DSA)	DigSig-SigVer	Verification of DSA signatures.		DSA SigVer (FIPS186-4): (A2836)
Signature Verification (ECDSA)	DigSig-SigVer	Verification of ECDSA signatures.		ECDSA SigVer (FIPS186-4): (A2836)
Signature Verification (RSA)	DigSig-SigVer	Verification of RSA signatures.		RSA SigVer (FIPS186-4): (A2836)

Table 9: Security Function Implementations

2.7 Algorithm Specific Information

2.7.1 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.

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

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 construction 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 800-56Cr2** 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 ≤ s ≤ 160
(HMAC-)SHA-224	up-to 512	HMAC-SHA-224	224	112 ≤ s ≤ 224
(HMAC-)SHA-256	up-to 512	HMAC-SHA-256	256	112 ≤ s ≤ 256
(HMAC-)SHA-384	up-to 1024	HMAC-SHA-384	384	112 ≤ s ≤ 384
(HMAC-)SHA-512	up-to 1024	HMAC-SHA-512	512	112 ≤ s ≤ 512
AES-128-CMAC	128	AES-128-CMAC	128	112 ≤ s ≤ 128
AES-192-CMAC	192	AES-192-CMAC	128	112 ≤ s ≤ 128
AES-256-CMAC	256	AES-256-CMAC	128	112 ≤ s ≤ 128

2.7.3 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.

2.7.4 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).

2.7.4 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.

2.7.5 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. AES-XTS is only approved for storage purposes. Per IG C.I, AES-XTS Key_1 and/or Key_2, when entered into the module by the operator, shall be generated and/or established independently of each other according to the rules for component symmetric keys from NIST SP 800-133rev2, Sec. 6.3. for an approved use of AES-XTS. The AES-XTS key is parsed as the 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, attempts to perform AES-XTS encryption or decryption will fail.

2.7.6 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.

The module also generates keys according to NIST SP 800-133-r2, Section 6.3, Method 1 for the concatenation of Key_1 and Key_2 as part of AES-XTS.

2.7.7 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.

2.7.8 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.

2.7.9 SHA-1 Allowed Only for SigVer

SHA-1 use is not permitted for any purposes other than Digital Signature Verification in accordance with I.G. Annex C.M Legacy Algorithms.

2.7.10 “KTS” Authenticated encryption/decryption as a service

The module does not establish SSPs using an approved key transport scheme (KTS). However, it does offer approved authenticated algorithms that can be used by an external operator/application as part of an approved KTS.

2.8 RBG and Entropy

N/A for this module.

Name	Type	Operational Environment	Sample Size	Entropy per Sample	Conditioning Component
CPU Jitter random number generator	Non-Physical	Linux Ubuntu 20.04 LTS on ARM Cortex-A7 (ARMv7-a 32-bit); Linux Ubuntu 20.04 LTS on ARM Cortex-A72 (ARMv8-a 64-bit); Linux Ubuntu 20.04 LTS on Intel Atom x5 (X86 32-bit); Linux Ubuntu 20.04 LTS on Intel Atom x5 (X86 64-bit)	256 bits	Full entropy	SHA3-256 (A2890)

Table 10: Entropy Sources

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. If another entropy source is used, the following caveat applies to the module: *No assurance of the minimum strength of generated SSPs*

2.9 Key Generation

- Symmetric Key Generation
  - Type: CKG (NIST SP 800-133r2 6.1, FIPS 140-3 IG D.H)
  - AES keys, key-derivation keys, MAC keys
- DH keypair generation

    - Type: CKG (NIST SP 800-133r2 5.2, FIPS 140-3 IG D.H, NIST SP 800-56Ar3)
    - 2048-8192 bits
- ECDH keypair generation
    - CKG (NIST SP 800-133r2 5.2, FIPS 140-3 IG D.H, NIST SP 800-56Ar3)
    - NIST P-224, P-256, P-384 and P-521 curves
- DSA keypair generation
    - CKG (NIST SP 800-133r2 5.1, FIPS 140-3 IG D.H, FIPS 186-4)
    - (L,N) = (2048, 224), (2048, 256), (3072, 256)
- ECDSA keypair generation
    - CKG (NIST SP 800-133r2 5.1, FIPS 140-3 IG D.H, FIPS 186-4)
    - NIST P-224, P-256, P-384 and P-521 curves
- RSA keypair generation
    - CKG (NIST SP 800-133r2 5.1, FIPS 140-3 IG D.H, FIPS 186-4)
    - 2048, 3072, 4096 bits
- KDF SP800-108
    - Key Derivation (NIST SP 800-108r1)
    - Method: Counter, feedback, double pipelines modes, using SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, AES-CMAC
- KDA HKDF
    - Key Derivation (NIST SP 800-56Cr2)
    - Using HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512
- KDA TwoStep
    - Key Derivation (NIST SP 800-56Cr2)
    - Method: counter and feedback modes, using SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512 or AES-CMAC
- KDF IKEv1 (CVL)
    - Key Derivation (Application specific – NIST SP 800-135r1)
    - SHA-1, SHA2-224, SHA2-384
- KDF IKEv2 (CVL)
    - Key Derivation (Application specific – NIST SP 800-135r1)
    - SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512
- KDF SRTP (CVL)
    - Key Derivation (Application specific - NIST SP 800-135r1)
    - AES-128, -192, -256
- PBKDF
    - Key Derivation (Application specific – NIST SP 800-132)
    - SHA-1, SHA2-256
- TLS v1.2 KDF RFC7627 (CVL)
    - Key Derivation (Application specific – NIST SP 800-135r1)
    - SHA2-256, SHA2-384, SHA2-512

2.10 Key Establishment

- AES-KW, AES KWP
    - Key Wrapping and Unwrapping (NIST SP 800-38F, FIPS 140-3 IG D.G)
    - AES-128, -192, -256
- KTS IFC
    - Key Encapsulation and Un-encapsulation (NIST SP 800-56Br2, FIPS 140-3 IG D.G)

    - KTS-OAEP basic
    - 2048, 3072, 4096 bits – SHA2-224, SHA2-256, SHA2-384, SHA2-512
- KAS-ECC CDH-Component (CVL)
    - Key Agreement Scheme (CDH component) (NIST SP 800-56Ar3)
    - NIST P-224, P-256, P-384 and P-521 curves
- KAS-ECC-SSC
    - Key Agreement Scheme (Shared Secret Computation) (NIST SP 800-56Ar3, FIPS 140-3 IG D.F scenario 2 path (1))
    - NIST P-224, P-256, P-384 and P-521 curves
- KAS-FFC-SSC
    - Key Agreement Scheme (Shared Secret Computation) (NIST SP 800-56Ar3, FIPS 140-3 IG D.F scenario 2 path (1))
    - ffdhe2048, ffdhe 3072, ffdhe 4096, ffdhe 6144, ffdhe 8192, MODP2048, MODP3072, MODP4096, MODP6144, MODP8192, FIPS 186-type FFC parameter-size sets FB and FC

2.11 Industry Protocols

The module implements KDF for the TLS v1.2 protocol.

No parts of the TLS v1.2 protocol, other than the key derivation function mentioned above, have been tested by the CAVP and CMVP.

The module implements HKDF for the TLS v1.3 protocol.

No parts of the TLS v1.3 protocol, other than the key derivation function mentioned above, have been tested by the CAVP and CMVP.

The module implements IKEv1 KDF for the IKEv1 protocol.

No parts of the IKEv1 protocol, other than the key derivation function mentioned above, have been tested by the CAVP and CMVP.

The module implements IKEv1 KDF for the IKEv2 protocol.

No parts of the IKEv2 protocol, other than the key derivation function mentioned above, have been tested by the CAVP and CMVP.

The module does not contain the full implementation of TLS or IKE protocols.

3 Cryptographic Module Interfaces

3.1 Ports and Interfaces

Physical Port	Logical Interface(s)	Data That Passes
N/A	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	Control Input	The API function invoked and function parameters designated as control inputs.

Physical Port	Logical Interface(s)	Data That Passes
N/A	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	Status Output	The return value/data of the invoked API function.

Table 11: Ports and 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.

4 Roles, Services, and Authentication

4.1 Authentication Methods

Method Name	Description	Security Mechanism	Strength Each Attempt	Strength per Minute

Table 12: Authentication Methods

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

4.2 Roles

Name	Type	Operator Type	Authentication Methods
Crypto Officer	Role	Crypto Officer	
User	Role	User	

Table 13: Roles

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.

4.3 Approved Services

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
AES data wrapping	Wrap AES data 38F	ARG	FLS_CryptKw() Key asset, data to be wrapped	FLR_OK (0) or error, Wrapped data	Block Ciphers AES KW/KWP (Authenticated)	Crypto Officer - AES key: E User - AES key: E
AES key wrapping	Wrap AES key 38F	ARG	FLS_AssetsWrapAes38F() FLS_AssetsUnwrapAes38F() Key asset, derivation parameters	FLR_OK (0) or error, Derived key	Block Ciphers AES KW/KWP (Authenticated)	Crypto Officer - AES key: E User - AES key: E
Asymmetric key pair generation	Generate asymmetric key pairs and DSA/Diffie-Hellman Domain Parameter	ARG	FLS_AssetGenerateKeyPair() FLS_DH_KeyGen() Key generation parameters	FLR_OK (0) or error, Generated key	Domain Parameter Generation (FFC) Key Generation (ECC) Key Generation (FFC) Key Generation (RSA)	Crypto Officer - DH private key: G,W - DH public key: G,W - DSA private key: G,W - DSA public key: G,W - EC private key: G,W - EC public key: G,W - RSA

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						private key: G,W - RSA public key: G,W User - DH private key: G,W - DH public key: G,W - DSA private key: G,W - DSA public key: G,W - EC private key: G,W - EC public key: G,W - RSA private key: G,W - RSA public key: G,W
Authenticated encryption and decryption	Service for data encryption and decryption with added authentication	ARG	FLS_CryptAuthInit() FLS_CryptGcmAadContinue() FLS_CryptGcmAadFinish() FLS_CryptAuthContinue() FLS_EncryptAuthFinish() FLS_EncryptAuth	FLR_OK (0) or error, Plaintext / ciphertext	Block Ciphers AEAD (Authenticated)	Crypto Officer - AES key: E User - AES key: E

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
			PacketFinish() FLS_DecryptAuthFinish() FLS_EncryptAuth InitRandom() FLS_EncryptAuthInitDeterministic() FLS_CryptAuthInitTls13() FLS_EncryptAuthTls13() FLS_EncryptAuthFinishTls13() FLS_DecryptAuthTls13() FLS_DecryptAuthFinishTls13() FLS_EncryptAuthSrtp() FLS_DecryptAuthSrtp() FLS_EncryptAuthSrtcp() FLS_DecryptAuthSrtcp() Key asset, crypto parameters, additional authenticated data, plaintext/ciphertext			
Copy key asset	Copies key value	ARG	FLS_AssetCopyValue() Source asset	FLR_OK (0) or error, Target asset	None	Crypto Officer - AES key: W - DH private key: W - DH public key: W - DSA private key: W - DSA public key: W - EC private key: W - EC public key: W - Key Derivation Key: W - MAC key: W

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						- PBKDF Password: W - RSA private key: W - RSA public key: W - Triple-DES key: W - Trusted Root Key: W User - AES key: W - DH private key: W - DH public key: W - DSA private key: W - DSA public key: W - EC private key: W - EC public key: W - Key Derivation Key: W - MAC key: W - PBKDF Password: W - RSA

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						private key: W - RSA public key: W - Triple-DES key: W - Trusted Root Key: W
Create key asset	Setup key policy, allocate memory for key, and load key value	ARG	FLS_AssetAllocateBasic() FLS_AssetAllocate() FLS_AssetAllocateAndAssociateKeyExtra() FLS_AssetLoadValue() FLS_AssetLoadMultipart() FLS_AssetLoadMultipartConvertBigInt() FLS_AssetPoke() FL_LocalAllocate() FL_LocalAllocateEx() Key policy, key value	FLR_OK (0) or error, Created asset	None	Crypto Officer - AES key: W - DH private key: W - DH public key: W - DSA private key: W - DSA public key: W - EC private key: W - EC public key: W - Key Derivation Key: W - MAC key: W - PBKDF Password: W - RSA private key: W - RSA public

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						key: W - Triple-DES key: W - Trusted Root Key: W User - AES key: W - DH private key: W - DH public key: W - DSA private key: W - DSA public key: W - EC private key: W - EC public key: W - Key Derivation Key: W - MAC key: W - PBKDF Password: W - RSA private key: W - RSA public key: W - Triple-DES key: W -

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						Trusted Root Key: W
Create trusted root key	Allocate and set data for new root key asset	ARG	FLS_RootKeyAllocateAndLoadValue() Key material	FLR_OK (0) or error	KDF (KBKDF)	Crypto Officer - Trusted Root Key: G,W
Delete key asset	Deletes a key and zeroes the data	ARG	FLS_AssetFree() FLS_LocalFree() Asset to be deleted	FLR_OK (0) or error	None	Crypto Officer - AES key: Z - CTR_DRBG entropy: Z - CTR_DRBG internal state (Key): Z - CTR_DRBG internal state (V): Z - CTR_DRBG seed: Z - DH private key: Z - DH public key: Z - DH shared secret: Z - DSA private key: Z - DSA

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						public key: Z - EC private key: Z - EC public key: Z - ECDH shared secret: Z - KBKDF Derived Key: Z - KDA HKDF Derived Key: Z - KDA TwoStep Derived Key: Z - KDF IKEv1 Derived Key: Z - KDF IKEv2 Derived Key: Z - KDF SRTP Derived Key: Z - KDF TLS v1.2 Derived Key: Z - Key Derivation Key: Z - MAC key: Z -

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						PBKDF derived key: Z - PBKDF Password: Z - RSA private key: Z - RSA public key: Z - Triple-DES key: Z - Trusted Root Key: Z User - AES key: Z - CTR_DRBG entropy: Z - CTR_DRBG internal state (Key): Z - CTR_DRBG internal state (V): Z - CTR_DRBG seed: Z - DH private key: Z - DH public

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						key: Z - DH shared secret: Z - DSA private key: Z - DSA public key: Z - EC private key: Z - EC public key: Z - ECDH shared secret: Z - KBKDF Derived Key: Z - KDA HKDF Derived Key: Z - KDA TwoStep Derived Key: Z - KDF IKEv1 Derived Key: Z - KDF IKEv2 Derived Key: Z - KDF SRTP Derived Key: Z - KDF TLS v1.2

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						Derived Key: Z - Key Derivation Key: Z - MAC key: Z - PBKDF derived key: Z - PBKDF Password: Z - RSA private key: Z - RSA public key: Z - Triple-DES key: Z - Trusted Root Key: Z
Diffie-Hellman key agreement	DH key agreement	ARG	FLS_DeriveDh() FLS_DH_Derive() Private and public values	FLR_OK (0) or error, Derived key	Shared Secret Computation (FFC)	Crypto Officer - DH private key: E - DH public key: E - DH shared secret: G User - DH private key: E - DH public key: E - DH

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						shared secret: G
Digest computation	Compute a digest	ARG	FLS_HashInit() FLS_HashContinue() FLS_HashFinish() FLS_HashSingle() Input data / message	FLR_OK (0) or error, Hash value	Hashing (SHA)	Crypto Officer User
DSA/Diffie-Hellman Domain Parameter verification	DSA/Diffie-Hellman Domain Parameter verification	ARG	FLS_AssetCheck() Key asset to be examined	FLR_OK (0) or error	Domain Parameter Verification (FFC)	Crypto Officer - DSA private key: E - DSA public key: E User - DSA private key: E - DSA public key: E
Elliptic Curve Diffie-Hellman key agreement	Elliptic Curve DH agreement	ARG	FLS_DeriveDh() Private and public values	FLR_OK (0) or error, Derived key	Shared Secret Computation (ECC)	Crypto Officer - EC private key: E - EC public key: E - ECDH shared secret: G User - EC private key: E - EC public key: E - ECDH shared secret: G
Encryption and	Service for data	ARG	FLS_CipherInit() FLS_CipherContinue()	FLR_OK (0) or	Block Ciphers	Crypto Officer

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
decryption	encryption and decryption		FLS_CipherFinish() Key asset, crypto parameters, plaintext/ciphertext	error, Plaintext / ciphertext	(Unauthenticated) Block Ciphers (Legacy)	- AES key: E - Triple-DES key: E User - AES key: E - Triple-DES key: E
Enter User role	Enter regular user mode	None	FLS_LibEnterUserRole()	FLR_OK (0) or error	None	Crypto Officer
Erase asset	Erase asset	None	FLS_EraseAsset() Asset to be erased	FLR_OK (0) or error	None	Crypto Officer - AES key: Z - CTR_DRBG entropy: Z - CTR_DRBG internal state (Key): Z - CTR_DRBG internal state (V): Z - CTR_DRBG seed: Z - DH private key: Z - DH public key: Z - DH shared secret:

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						Z - DSA private key: Z - DSA public key: Z - EC private key: Z - EC public key: Z - ECDH shared secret: Z - KBKDF Derived Key: Z - KDA HKDF Derived Key: Z - KDA TwoStep Derived Key: Z - KDF IKEv1 Derived Key: Z - KDF IKEv2 Derived Key: Z - KDF SRTP Derived Key: Z - KDF TLS v1.2 Derived Key: Z - Key Derivati

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						on Key: Z - MAC key: Z - PBKDF derived key: Z - PBKDF Password: Z - RSA private key: Z - RSA public key: Z - Triple-DES key: Z - Trusted Root Key: Z User - AES key: Z - CTR_DRBG entropy: Z - CTR_DRBG internal state (Key): Z - CTR_DRBG internal state (V): Z - CTR_DRBG seed: Z

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						- DH private key: Z - DH public key: Z - DH shared secret: Z - DSA private key: Z - DSA public key: Z - EC private key: Z - EC public key: Z - ECDH shared secret: Z - KBKDF Derived Key: Z - KDA HKDF Derived Key: Z - KDA TwoStep Derived Key: Z - KDF IKEv1 Derived Key: Z - KDF IKEv2 Derived Key: Z - KDF SRTP

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						Derived Key: Z - KDF TLS v1.2 Derived Key: Z - Key Derivation Key: Z - MAC key: Z - PBKDF derived key: Z - PBKDF Password: Z - RSA private key: Z - RSA public key: Z - Triple-DES key: Z - Trusted Root Key: Z
Erase data from memory	Erase data	None	FLS_Erase() Memory area to be erased	FLR_OK (0) or error	None	Crypto Officer - AES key: Z - CTR_DRBG entropy: Z - CTR_DRBG internal state (Key): Z

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						- CTR_DRBG internal state (V): Z - CTR_DRBG seed: Z - DH private key: Z - DH public key: Z - DH shared secret: Z - DSA private key: Z - DSA public key: Z - EC private key: Z - EC public key: Z - ECDH shared secret: Z - KBKDF Derived Key: Z - KDA HKDF Derived Key: Z - KDA TwoStep Derived Key: Z

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						- KDF IKEv1 Derived Key: Z - KDF IKEv2 Derived Key: Z - KDF SRTP Derived Key: Z - KDF TLS v1.2 Derived Key: Z - Key Derivation Key: Z - MAC key: Z - PBKDF derived key: Z - PBKDF Password: Z - RSA private key: Z - RSA public key: Z - Triple-DES key: Z - Trusted Root Key: Z User - AES key: Z - CTR_D

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						RBG entropy: Z - CTR_DRBG internal state (Key): Z - CTR_DRBG internal state (V): Z - CTR_DRBG seed: Z - DH private key: Z - DH public key: Z - DH shared secret: Z - DSA private key: Z - DSA public key: Z - EC private key: Z - EC public key: Z - ECDH shared secret: Z - KBKDF Derived Key: Z

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						- KDA HKDF Derived Key: Z - KDA TwoStep Derived Key: Z - KDF IKEv1 Derived Key: Z - KDF IKEv2 Derived Key: Z - KDF SRTP Derived Key: Z - KDF TLS v1.2 Derived Key: Z - Key Derivation Key: Z - MAC key: Z - PBKDF derived key: Z - PBKDF Password: Z - RSA private key: Z - RSA public key: Z - Triple-DES key: Z

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						- Trusted Root Key: Z
Examine key asset policy, size	Get key size and check	ARG	FLS_AssetShow() FLS_AssetCheck() Asset to be examined	FLR_OK (0) or error, Key policy key size	None	Crypto Officer - AES key: R - CTR_DRBG entropy: R - CTR_DRBG internal state (Key): R - CTR_DRBG internal state (V): R - CTR_DRBG seed: R - DH private key: R - DH public key: R - DH shared secret: R - DSA private key: R - DSA public key: R - EC private key: R - EC

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						public key: R - ECDH shared secret: R - KBKDF Derived Key: R - KDA HKDF Derived Key: R - KDA TwoStep Derived Key: R - KDF IKEv1 Derived Key: R - KDF IKEv2 Derived Key: R - KDF SRTP Derived Key: R - KDF TLS v1.2 Derived Key: R - Key Derivation Key: R - MAC key: R - PBKDF derived key: R - PBKDF Passwo

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						rd: R - RSA private key: R - RSA public key: R - Software Integrity Public Key: R - Triple-DES key: R - Trusted Root Key: R User - AES key: R - CTR_DRBG entropy: R - CTR_DRBG internal state (Key): R - CTR_DRBG internal state (V): R - CTR_DRBG seed: R - DH private key: R - DH public

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						key: R - DH shared secret: R - DSA private key: R - DSA public key: R - EC private key: R - EC public key: R - ECDH shared secret: R - KBKDF Derived Key: R - KDA HKDF Derived Key: R - KDA TwoStep Derived Key: R - KDF IKEv1 Derived Key: R - KDF IKEv2 Derived Key: R - KDF SRTP Derived Key: R - KDF TLS v1.2

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						Derived Key: R - Key Derivation Key: R - MAC key: R - PBKDF derived key: R - PBKDF Password: R - RSA private key: R - RSA public key: R - Software Integrity Public Key: R - Triple-DES key: R - Trusted Root Key: R
Extensible Output Function	Compute an extended output	ARG	FLS_HashSingle() Input data / message	FLR_OK (0) or error, extended value	Hashing (XOF)	Crypto Officer User
Generate key	Generate random key	ARG	FLS_AssetLoadRandom()	FLR_OK (0) or error, KeyAsset	Key Generation (Symmetric)	Crypto Officer - AES key: G - MAC key: G User - AES key: G

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						- MAC key: G
Get crypto module description	Returns description of the module	None	FLS_LibDescription()	Module description	None	Crypto Officer User
Get crypto module status (Show Status)	Return state of the module	None	FLS_LibStatus()	Module status	None	Crypto Officer User
Get crypto module version (Show Version)	Returns version of the module	None	FLS_LibVersion()	Module version	None	Crypto Officer User
Get key asset value	Get key value	ARG	FLS_AssetPeek() Asset to peek	FLR_OK (0) or error, Key value	None	Crypto Officer - AES key: R - CTR_DRBG entropy: R - CTR_DRBG internal state (Key): R - CTR_DRBG internal state (V): R - CTR_DRBG seed: R - DH private key: R - DH

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						public key: R - DH shared secret: R - DSA private key: R - DSA public key: R - EC private key: R - EC public key: R - ECDH shared secret: R - KBKDF Derived Key: R - KDA HKDF Derived Key: R - KDA TwoStep Derived Key: R - KDF IKEv1 Derived Key: R - KDF IKEv2 Derived Key: R - KDF SRTP Derived Key: R - KDF TLS

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						v1.2 Derived Key: R - Key Derivation Key: R - MAC key: R - PBKDF derived key: R - PBKDF Password: R - RSA private key: R - RSA public key: R - Software Integrity Public Key: R - Triple-DES key: R - Trusted Root Key: R User - AES key: R - CTR_DRBG entropy: R - CTR_DRBG internal state

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						(Key): R - CTR_DRBG internal state (V): R - CTR_DRBG seed: R - DH private key: R - DH public key: R - DH shared secret: R - DSA private key: R - DSA public key: R - EC private key: R - EC public key: R - ECDH shared secret: R - KBKDF Derived Key: R - KDA HKDF Derived Key: R - KDA TwoStep Derived

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						Key: R - KDF IKEv1 Derived Key: R - KDF IKEv2 Derived Key: R - KDF SRTP Derived Key: R - KDF TLS v1.2 Derived Key: R - Key Derivation Key: R - MAC key: R - PBKDF derived key: R - PBKDF Password: R - RSA private key: R - RSA public key: R - Software Integrity Public Key: R - Triple-DES key: R - Trusted

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						Root Key: R
Get module information	Return basic information of the module	None	FLS_StaticConfig() FL_IntactID()	Build configuration, identifiers	None	Crypto Officer User
Get status of the asset store	Returns the status of the asset store	None	FLS_AssetStoreStatus()	Asset store memory status	None	Crypto Officer User
Get/set information on current module configuration	Return current module configuration	None	FLS_RuntimeConfigSetProperty, FLS_RuntimeConfigGetProperty Runrime configuration value	FLR_OK (0), error or current runtime value	None	Crypto Officer User
IKEv1 key derivation	Key derivation for IKEv1	ARG	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	KDF (ASKDF IKEv1)	Crypto Officer User
IKEv2 key derivation	Key derivation for IKEv2	None	FLS_IkePrfExtract() FLS_IKEv2ExtractSKEYSEED() FLS_IKEv2DeriveDKM() FLS_IKEv2ExtractSKEYSEEDrekey() Derivation parametes, input key material	FLR_OK (0) or error, Derived key	KDF (ASKDF IKEv2)	Crypto Officer User
Initialize the module	Initializes the module	None	FLS_LibInit()	FLR_OK (0) or error	None	Crypto Officer
Install entropy source	Install entropy source to be used	None	FL_RbgInstall EntropySource() FLS_RbgRequest SecurityStrength() Entropy source	FLR_OK (0) or error	None	Crypto Officer

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Key derivation	Key derivation	ARG	FLS_KeyDeriveKdk() Key asset, derivation parameters	FLR_OK (0) or error, Derived key	KDF (KBKDF)	Crypto Officer - KBKDF Derived Key: G,W - Key Derivation Key: E User - KBKDF Derived Key: G,W - Key Derivation Key: E
Key Derivation Through Extraction-then-expansion	HKDF key derivation or key derivation with two steps	ARG	FLS_HkdfExtract() FLS_HkdfExpandAsset() FLS_HkdfExpand() FLS_Hkdf() FLS_KeyDeriveKdk() Derivation parameters, input key material	FLR_OK (0) or error, Derived key	KDF (KDA)	Crypto Officer User
Load precomputed digest	Load a precomputed digest	ARG	FLS_LoadFinishedHashStateAlgo() Digest parameters	FLR_OK (0) or error, Target asset	Hashing (SHA)	Crypto Officer User
MAC generation	Service for MAC generation	ARG	FLS_MacGenerateInit() FLS_MacGenerateContinue() FLS_MacGenerateFinish() Key asset, crypto parameters, input data	FLR_OK (0) or error, MAC	MAC Generation	Crypto Officer - AES key: E - MAC key: E User - AES key: E - MAC key: E
MAC verification	Service for MAC	ARG	FLS_MacVerifyInit() FLS_MacVerifyContinue() FLS_MacVerifyFinish()	FLR_OK (0) or error,	MAC Verification	Crypto Officer - AES

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
	verification		Key asset, crypto parameters, input data	Verification status		key: E - MAC key: E User - AES key: E - MAC key: E
PBKDF2 key derivation	PBKDF2 key derivation	ARG	FLS_KeyDerivePbkdf2() Password, key derivation parameters	FLR_OK (0) or error, Derived key	KDF (PBKDF)	Crypto Officer - PBKDF derived key: G,W - PBKDF Password: R,E User - PBKDF derived key: G,W - PBKDF Password: R,E
Perform module self-tests	Execute self-tests	None	FLS_LibSelfTest()	FLR_OK (0) or error	None	Crypto Officer User
Random number generation	Generate random numbers by DRBG	ARG	FLS_RbgGenerateRandom() Random data generation parameters	FLR_OK (0) or error, Random data	Random Bit Generation	Crypto Officer - CTR_DRBG entropy: R - CTR_DRBG internal state (Key): R - CTR_DRBG

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						internal state (V): R - CTR_DRBG seed: G,E User - CTR_DRBG entropy: G,E - CTR_DRBG internal state (Key): E,W - CTR_DRBG internal state (V): W,E - CTR_DRBG seed: G,E
Random number generator reseeding	Force reseeding of random number generator	ARG	FLS_RbgReseed() Seed	FLR_OK (0) or error,	Entropy Source	Crypto Officer - CTR_DRBG entropy: G,E - CTR_DRBG internal state (Key): G,W - CTR_D

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						RBG internal state (V): G,W - CTR_DRBG seed: G,E User - CTR_DRBG entropy: G,E - CTR_DRBG internal state (Key): G,W - CTR_DRBG internal state (V): G,W - CTR_DRBG seed: G,E
Reset module state	Reset, zeroize and finalizes the module	None	FLS_LibUnInit()	FLR_OK (0) or error	None	Crypto Officer User
RSA-OAEP key wrapping	RSA-OAEP key wrap	ARG	FLS_AssetsWrapRsaOaep() FLS_AssetsUnwrapRsaOaep() Key asset, input data / wrapped key	FLR_OK (0) or error, Wrapped key / unwrapped key asset	KTS-IFC	Crypto Officer - RSA private key: E - RSA public key: E

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						User - RSA private key: E - RSA public key: E
Signature generation	Generate signature	ARG	FLS_HashSignFips186() FLS_HashSignPkcs1() FLS_HashSignPkcs1Pss() Key asset, hash value	FLR_OK (0) or error, Signature	Signature Generation (DSA) Signature Generation (ECDSA) Signature Generation (RSA)	Crypto Officer - DSA private key: E - DSA public key: E - EC private key: E - EC public key: E - RSA private key: E - RSA public key: E User - DSA private key: E - DSA public key: E - EC private key: E - EC public key: E - RSA private key: E - RSA public key: E
Signature	Verify signature	ARG	FLS_HashVerifyFips186() FLS_HashVerifyRecover	FLR_OK (0) or error,	Signature Verification (DSA)	Crypto Officer - DSA

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
verification			Pkcs1() FLS_HashVerifyPkcs1() FLS_HashVerifyPkcs1Pss() Key asset, signature	verification result	Signature Verification (ECDSA) Signature Verification (RSA)	private key: E - DSA public key: E - EC private key: E - EC public key: E - RSA private key: E - RSA public key: E User - DSA private key: E - DSA public key: E - EC private key: E - EC public key: E - RSA private key: E - RSA public key: E
SRTP key derivation	Key derivation for SRTP	None	FLS_SrtpKeyDerive() Key asset, derivation paramenters	FLR_OK (0) or error, Derived key	KDF (ASKDF SRTP)	Crypto Officer User
TLS v1.2 key derivation	Key derivation	ARG	FLS_DeriveTlsPrf() FLS_KeyDeriveKdk() Key asset, derivation parameters	FLR_OK (0) or error, Derived key	KDF (ASKDF TLS v1.2)	Crypto Officer User

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Trusted KDK key derivation	KDK key derivation	ARG	FLS_TrustedKeyDerive() Key asset, derivation parameters	FLR_OK (0) or error, Derived key	KDF (KBKDF)	Crypto Officer - KBKDF Derived Key: G,W - Key Derivation Key: E User - KBKDF Derived Key: G,W - Key Derivation Key: E
Trusted key wrapping	Wrap trusted key	ARG	FLS_AssetWrapTrusted() FLS_AssetUnwrapTrusted() Key assets, wrapping parameters	FLR_OK (0) or error, Wrapped key	Block Ciphers AES KW/KWP (Authenticated) KDF (KBKDF)	Crypto Officer - AES key: G,E - Key Derivation Key: R,E User - AES key: G,E - Key Derivation Key: R,E
Trusted root key derivation	Derive root key	ARG	FLS_TrustedKdkDerive() FLS_TrustedKekdkDerive() Key asset, derivation parameters	FLR_OK (0) or error, Derived key	KDF (KBKDF)	Crypto Officer - KBKDF Derived Key: G,W - Trusted Root Key: E

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						User - KBKDF Derived Key: G,W - Trusted Root Key: E

Table 14: Approved Services

The indicator column may be 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.

4.4 Non-Approved Services

Name	Description	Algorithms	Role
AES key wrapping	AES key wrapping that does not fulfill NIST SP 800-38F	AES-KEY WRAP	Crypto Officer; User
Asymmetric key generation	Generate keys with unapproved parameters (e.g., key lengths and curves)	Brainpool DSA Key Pair Generation ECDSA Key Pair Generation RSA Key Pair Generation	Crypto Officer; User
Digest computation	Compute a digest with MD5	MD5	Crypto Officer; User
Elliptic Curve Diffie-Hellman key agreement	Elliptic Curve DH agreement with unapproved curve	Brainpool ECC Diffie-Hellman	Crypto Officer; User
Encryption and decryption	Service for data encryption and decryption	ChaCha20-Poly1305 Triple-DES Encryption	Crypto Officer; User
KDK key derivation	KDK key derivation	KDF NIST SP 800-108	Crypto Officer; User
MAC generation	Service for MAC generation with unapproved key lengths	HMAC	Crypto Officer; User

Name	Description	Algorithms	Role
MAC verification	Service for MAC verification with unapproved key lengths	HMAC	Crypto Officer; User
RSA decryption primitive	RSA decryption primitive only	RSA Private Key Primitives (NIST SP 800-56B)	Crypto Officer; User
RSA encryption primitive	RSA encryption primitive only	RSA Public Key Primitives (NIST SP 800-56B)	Crypto Officer; User
RSA signature generation primitive	Generate RSA primitive only	RSA Private Key Primitives (NIST SP 800-56B)	Crypto Officer; User
RSA signature verification primitive	RSA verification primitive only	RSA Public Key Primitives (NIST SP 800-56B)	Crypto Officer; User
RSA-KEM key wrapping	RSA-KEM key wrap	KTS (KEM NIST SP 800-56B)	Crypto Officer; User
RSA-OAEP key wrapping	RSA-OAEP key wrap	KTS (OAEP NIST SP 800-56B)	Crypto Officer; User
RSA-PKCS#1 v1.5 key wrapping	RSA-PKCS#1 key wrap	RSA Encryption (PKCS #1 v1.5)	Crypto Officer; User
Signature generation	Generate signatures with unapproved parameters (e.g., key lengths and curves)	Brainpool DSA Signature Generation ECDSA Signature Generation RSA Signature Generation	Crypto Officer; User
Signature verification	Verify signature with unapproved key length	RSA Signature Validation	Crypto Officer; User
TLS v1.0/1.1 key derivation	Key derivation, SP 800-135 rev 1	TLS1.0/1.1 KDF NIST SP 800-135rev1	Crypto Officer; User
Trusted key wrapping	Wrap trusted key	KDF NIST SP 800-108	Crypto Officer; User
X25519	X25519 Key agreement	X25519 Key Agreement	Crypto Officer; User

Table 15: Non-Approved Services

4.5 External Software/Firmware Loaded

The module does not load external software or firmware.

5 Software/Firmware Security

5.1 Integrity Techniques

The SafeZone FIPS SW Cryptographic Module integrity is checked with an ECDSA signature verification using NIST P-224 and SHA2-224 (Cert. #A2836). If the integrity test fails, the module enters the error state.

5.2 Initiate on Demand

The integrity test is executed as part of the pre-operational self-tests. These tests are executed upon module initialization. The integrity test can be invoked on-demand by unloading (FLS_LibUninit command) and re-initializing (FLS_LibInit command) the module. It can also be invoked by calling the LibSelfTest command which executes several tests including the software integrity test.

5.3 Open-Source Parameters

Even if the module is not open source, but contains some open source software, in accordance with ISO19790 Annex B.2.5, the compilers and tools used to build the module as tested are:

- gcc v5.4.0
- perl v5.22.1
- make v4.1

The module's executable form is the compiled binaries listed in Section 2.2.

5.4 Additional Information

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

6 Operational Environment

6.1 Operational Environment Type and Requirements

Type of Operational Environment: Modifiable

How Requirements are Satisfied:

All processes spawned by the module are child processes of the module, and ownership of a process cannot be changed.

7 Physical Security

This section does not apply as the module is software-only.

8 Non-Invasive Security

This section does not apply as the module does not implement non-invasive attack mitigation mechanisms.

9 Sensitive Security Parameters Management

9.1 Storage Areas

Storage Area Name	Description	Persistence Type
Memory	Volatile storage consisting of the modules allocated memory space on the host operating system.	Dynamic

Table 16: Storage Areas

9.2 SSP Input-Output Methods

Name	From	To	Format Type	Distribution Type	Entry Type	SFI or Algorithm
API Inputs	CM Software	App via TOEPP Path	Plaintext	Manual	Electronic	
API Outputs	App via TOEPP Path	CM Software	Plaintext	Manual	Electronic	

Table 17: SSP Input-Output Methods

9.3 SSP Zeroization Methods

Zeroization Method	Description	Rationale	Operator Initiation
Asset deletion	By invoking FLS_Erase(),FLS_EraseAsset(), FLS_AssetFree(), FLS_LocalFree() API functions, assets are zeroized by the user.	All SSPs are zeroized and deallocated from memory when no longer needed.	API function call.
Module uninitialization	Upon uninitialization, the module performs a zeroization of the allocated assets (SSPs).	The module zeroizes all allocated SSPs at uninitialization.	By uninitializing the module.

Table 18: SSP Zeroization Methods

9.4 SSPs

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
AES key	Key used for symmetric encryption and decryption	128, 192, 256, 512 (AES-XTS) bits - 128, 192, 256 bits	Symmetric Key - CSP	KDF (ASKDF IKEv1) KDF (ASKDF IKEv2) KDF (ASKDF SRTP) KDF (ASKDF TLS v1.2) KDF (KBKDF) KDF (KDA) KDF (PBKDF) Key Generation (Symmetric) CKG (AES-XTS)		Block Ciphers AEAD (Authenticated) Block Ciphers AES KW/KWP (Authenticated) Block Ciphers (Unauthenticated)
CTR_DRBG entropy	Entropy input used to seed the DRBG	256-1024 bits - 128-256 bits	Entropy input - CSP	Entropy Source		Random Bit Generation
CTR_DRBG internal state (Key)	Key for DRBG used for random number and key/key pair generation purposes	384 bits - 256 bits	DRBG internal state - CSP	Entropy Source		Random Bit Generation
CTR_DRBG internal state (V)	V value for DRBG used for random number	256 bits - 128 bits	DRBG internal state - CSP	Entropy Source		Random Bit Generation

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
	and key/key pair generation purposes					
CTR_DRBG seed	DRBG seed derived from entropy input	256-384 bits - 128-256 bits	Seed - CSP	Entropy Source		Random Bit Generation
DH private key	Private key used for shared secret computation	2048-8192 bits - 112-200 bits	Private Key - CSP	Key Generation (FFC)		Shared Secret Computation (FFC)
DH public key	Public key used for shared secret computation	2048-8192 bits - 112-200 bits	Public Key - PSP	Key Generation (FFC)		Shared Secret Computation (FFC)
DH shared secret	Shared secret established by DH	2048-8192 bits - 112-200 bits	Shared Secret - CSP	KAS-FFC-SSC Sp800-56Ar3 (A2836)	Shared Secret Computation (FFC)	KDF (KDA)
DSA private key	Private key used for DSA signature generation	(L,N) = (2048, 224), (2048, 256), (3072, 256) - 112-128 bits	Private Key - CSP	Key Generation (FFC)		Signature Generation (DSA)
DSA public key	Public key used for DSA signature verification	(L,N) = (2048, 224), (2048, 256), (3072, 256) - 112-128 bits	Public Key - PSP	Key Generation (FFC)		Signature Verification (DSA)

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
EC private key	Private key used for ECDSA signature generation or shared secret computation	NIST P-224, P-256, P-384, P-521 curves - 112-256 bits	Private Key - CSP	Key Generation (ECC)		Signature Generation (ECDSA) Shared Secret Computation (ECC)
EC public key	Public key used for ECDSA signature generation or shared secret computation	NIST P-224, P-256, P-384, P-521 curves - 112-256 bits	Public Key - PSP	Key Generation (ECC)		Signature Verification (ECDSA) Shared Secret Computation (ECC)
ECDH shared secret	Shared secret established by ECDH	NIST P-224, P-256, P-384, P-521 curves - 112-256 bits	Shared Secret - CSP	KAS-ECC-SSC Sp800-56Ar3 (A2836)	Shared Secret Computation (FFC)	KDF (KDA)
KBKDF Derived Key	KBKDF derived key	112-512 bits - 112-256 bits	Derived key - CSP	KDF (KBKDF)		Block Ciphers AEAD (Authenticated) Block Ciphers AES KW/KWP (Authenticated) Block Ciphers (Unauthenticated) MAC Generation MAC Verification
KDA HKDF Derived Key	KDA HKDF derived key	2048 bits - 112-256 bits	Derived key - CSP	KDF (KDA)		Block Ciphers AEAD (Authenticated) Block Ciphers AES KW/KWP (Authenticated

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
						) Block Ciphers (Unauthenticated) MAC Generation MAC Verification
KDA TwoStep Derived Key	KDA TwoStep derived key	2048 bits - 112-256 bits	Derived key - CSP	KDF (KDA)		Block Ciphers AEAD (Authenticated) Block Ciphers AES KW/KWP (Authenticated) Block Ciphers (Unauthenticated) MAC Generation MAC Verification
KDF IKEv1 Derived Key	KDF IKEv1 derived key	- 112 bits	Derived key - CSP	KDF (ASKDF IKEv1)		Block Ciphers AEAD (Authenticated) Block Ciphers AES KW/KWP (Authenticated) Block Ciphers (Unauthenticated) MAC Generation MAC Verification
KDF IKEv2 Derived Key	KDF IKEv2 derived key	3072 bits - 112 bits	Derived key - CSP	KDF (ASKDF IKEv2)		Block Ciphers AEAD (Authenticated) Block Ciphers AES KW/KWP (Authenticated) Block Ciphers (Unauthenticat

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
						ed) MAC Generation MAC Verification
KDF SRTP Derived Key	KDF SRTP derived key	N/A - 112-256 bits	Derived key - CSP	KDF (ASKDF SRTP)		Block Ciphers AEAD (Authenticated) Block Ciphers AES KW/KWP (Authenticated) Block Ciphers (Unauthenticated) MAC Generation MAC Verification
KDF TLS v1.2 Derived Key	KDF TSL v1.2 derived key	112-256 bits - 112-256 bits	Derived key - CSP	KDF (ASKDF TLS v1.2)		Block Ciphers AEAD (Authenticated) Block Ciphers AES KW/KWP (Authenticated) Block Ciphers (Unauthenticated) MAC Generation MAC Verification
Key Derivation Key	Key Derivation Key used for key-based key derivation	112-256 bits - 112-256 bits	Symmetric Key - CSP	External		KDF (ASKDF IKEv1) KDF (ASKDF IKEv2) KDF (ASKDF SRTP) KDF (ASKDF TLS v1.2) KDF (KBKDF) KDF (KDA)
MAC key	HMAC key used for HMAC	112-512 bits -	Authentication Key - CSP	Key Generation		MAC Generation

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
	generation and verification	112-256 bits		(Symmetric) KDF (ASKDF IKEv1) KDF (ASKDF IKEv2) KDF (ASKDF SRTP) KDF (ASKDF TLS v1.2) KDF (KBKDF) KDF (KDA) KDF (PBKDF)		MAC Verification
PBKDF derived key	Key derived from PBKDF password	112-4096 bits - 112-256 bits	Derived Key - CSP	KDF (PBKDF)		Block Ciphers AEAD (Authenticated) Block Ciphers AES KW/KWP (Authenticated) Block Ciphers (Unauthenticated) MAC Generation MAC Verification
PBKDF Password	Used for PBKDF2 key derivation	Varies (at least 12 characters) - N/A	Password or passphrase - CSP	External		KDF (PBKDF)
RSA private key	Private key used for RSA signature generation or; key transport (OAEP	2048, 3072, 4096 bits - 112, 128, 150 bits	Private Key - CSP	Key Generation (RSA)		Signature Generation (RSA) KTS-IFC

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
	Unwrapping)					
RSA public key	Public key used for RSA signature verification or key transport (OAEP Wrapping)	1024 (only if imported), 2048, 3072, 4096 bits - 80 (only if imported), 112, 128, 150 bits	Public Key - PSP	Key Generation (RSA)		Signature Verification (RSA) KTS-IFC
Software Integrity Public Key	Public key used by Power-on Software Integrity to ensure the integrity of the Cryptographic Module.	NIST P-224 curve - 112 bits	Public Key - Neither	Externally: Manufacturer pre-loaded		Signature Verification (ECDSA)
Triple-DES key	Key used for symmetric decryption	192 bits - 112 bits	Symmetric Key - CSP	External		Block Ciphers (Unauthenticated)
Trusted Root Key	Key used for deriving other keys as per NIST SP 800-108. Can only derive Trusted KDK and Trusted KEKDK keys.	256 bits - 256 bits	Symmetric Key - CSP	External		KDF (KBKDF)

Table 19: SSP Table 1

Name	Input - Output	Storage	Storage Duration	Zeroization	Related SSPs
AES key	API Inputs	Memory:Plaintext	For the duration of the service	Asset deletion Module uninitialization	

Name	Input - Output	Storage	Storage Duration	Zeroization	Related SSPs
CTR_DRBG entropy		Memory:Plaintext	From generation to DRBG seeding	Asset deletion Module uninitialization	
CTR_DRBG internal state (Key)		Memory:Plaintext	From instantiation to uninstantiation of the DRBG	Asset deletion Module uninitialization	
CTR_DRBG internal state (V)		Memory:Plaintext	From instantiation to uninstantiation of the DRBG	Module uninitialization Asset deletion	
CTR_DRBG seed		Memory:Plaintext	While seeding the DRBG	Asset deletion Module uninitialization	
DH private key	API Inputs API Outputs	Memory:Plaintext	For the duration of the service	Asset deletion Module uninitialization	
DH public key	API Inputs API Outputs	Memory:Plaintext	For the duration of the service	Asset deletion Module uninitialization	
DH shared secret	API Outputs	Memory:Plaintext	For the duration of the service	Asset deletion Module uninitialization	
DSA private key	API Inputs API Outputs	Memory:Plaintext	For the duration of the service	Asset deletion Module uninitialization	
DSA public key	API Inputs API Outputs	Memory:Plaintext	For the duration of the service	Asset deletion Module uninitialization	
EC private key	API Inputs API Outputs	Memory:Plaintext	For the duration of the service	Asset deletion Module uninitialization	
EC public key	API Inputs API Outputs	Memory:Plaintext	For the duration of the service	Asset deletion Module uninitialization	
ECDH shared secret	API Outputs	Memory:Plaintext	For the duration of the service	Asset deletion Module uninitialization	
KBKDF Derived Key	API Outputs	Memory:Plaintext	For the duration of the service	Asset deletion Module uninitialization	
KDA HKDF Derived Key	API Outputs	Memory:Plaintext	For the duration of the service	Asset deletion Module uninitialization	

Name	Input - Output	Storage	Storage Duration	Zeroization	Related SSPs
KDA TwoStep Derived Key	API Outputs	Memory:Plaintext	For the duration of the service	Asset deletion Module uninitialization	
KDF IKEv1 Derived Key	API Outputs	Memory:Plaintext	For the duration of the service	Asset deletion Module uninitialization	
KDF IKEv2 Derived Key	API Outputs	Memory:Plaintext	For the duration of the service	Asset deletion Module uninitialization	
KDF SRTP Derived Key	API Outputs	Memory:Plaintext	For the duration of the service	Asset deletion Module uninitialization	
KDF TLS v1.2 Derived Key	API Outputs	Memory:Plaintext	For the duration of the service	Asset deletion Module uninitialization	
Key Derivation Key	API Inputs	Memory:Plaintext	For the duration of the service	Asset deletion Module uninitialization	
MAC key	API Inputs	Memory:Plaintext	For the duration of the service	Asset deletion Module uninitialization	
PBKDF derived key	API Outputs	Memory:Plaintext	For the duration of the service	Asset deletion Module uninitialization	
PBKDF Password	API Inputs	Memory:Plaintext	For the duration of the service	Asset deletion Module uninitialization	
RSA private key	API Inputs API Outputs	Memory:Plaintext	For the duration of the service	Asset deletion Module uninitialization	
RSA public key	API Inputs API Outputs	Memory:Plaintext	For the duration of the service	Asset deletion Module uninitialization	
Software Integrity Public Key		Memory:Plaintext	N/A	N/A	
Triple-DES key	API Inputs	Memory:Plaintext	For the duration of the service	Asset deletion Module uninitialization	
Trusted Root Key	API Inputs	Memory:Plaintext	N/A	Asset deletion Module uninitialization	

Table 20: SSP Table 2

9.5 Transitions

DSA, RSA and ECDSA algorithms implemented by the module conform to FIPS 186-4 which was superseded by FIPS 186-5 on February 3, 2024. This is a soft transition as specified by FIPS 140-3 IG C.K.

10 Self-Tests

10.1 Pre-Operational Self-Tests

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details
ECDSA SigVer (FIPS 186-4)	NIST P-224 curve with SHA2-224	KAT	SW/FW Integrity	Self-test successful	Verification of precomputed digital signature

Table 21: Pre-Operational Self-Tests

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.

10.2 Conditional Self-Tests

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
AES-CBC Decrypt	128-bit key	KAT	CAST	FLS_LibInit or FLS_LibSelftest returns FLR_OK	Decrypt	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
AES-CBC Encrypt	128-bit key	KAT	CAST	FLS_LibInit or FLS_LibSelftest returns FLR_OK	Encrypt	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
AES-CCM Decrypt	128-bit key	KAT	CAST	FLS_LibInit or FLS_LibSelftest	Decrypt	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
				returns FLR_OK		
AES-CCM Encrypt	128-bit key	KAT	CAST	FLS_LibInit or FLS_LibSelftest returns FLR_OK	Encrypt	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
AES-CMAC	192-bit key	KAT	CAST	FLS_LibInit or FLS_LibSelftest returns FLR_OK	MAC Verification	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
AES-GCM Decrypt	128-bit key	KAT	CAST	FLS_LibInit or FLS_LibSelftest returns FLR_OK	Decrypt	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
AES-GCM Encrypt	128-bit key	KAT	CAST	FLS_LibInit or FLS_LibSelftest returns FLR_OK	Encrypt	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
AES-XTS Decrypt	128-bit key	KAT	CAST	FLS_LibInit or FLS_LibSelftest returns FLR_OK	Decrypt	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
AES-XTS Encrypt	128-bit key	KAT	CAST	FLS_LibInit or FLS_LibSelftest returns FLR_OK	Encrypt	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
AES-XTS Key Comparison	Key_1 != Key_2	Key non-equivalency test	Critical Function	Invoked API function returns FLR_OK	Cipher initialization	API function initializing AES-XTS operations (FLS_CipherInit)
CTR_DRBG	AES-256	KAT	CAST	FLS_LibInit or FLS_LibSelftest	Instantiate, Generate, Reseed	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
				returns FLR_OK		
DH	Key establishment	KAT	CAST	Invoked API function returns FLR_OK	DH Keypair Generation	API functions invoking keypair generation (FLS_AssetGenerateKeyPair(), FLS_DH_KeyGen(), FLS_DeriveDh(), FLS_DH_Derive())
DSA	Signature generation and verification	PCT	PCT	Invoked API function returns FLR_OK	DSA Keypair Generation	API functions invoking keypair generation (FLS_AssetGenerateKeyPair(), FLS_DH_KeyGen(), FLS_DeriveDh(), FLS_DH_Derive())
DSA Signature Generation	P=2048/N=160 with SHA2-224	KAT	CAST	FLS_LibInitor FLS_LibSelftest returns FLR_OK	Signature Generation	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
DSA Signature Verification	P=2048/N=160 with SHA2-224	KAT	CAST	FLS_LibInitor FLS_LibSelftest returns FLR_OK	Signature Verification	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
ECDH	Key establishment	KAT	CAST	Invoked API function returns FLR_OK	ECDH Keypair Generation	API functions invoking keypair generation (FLS_AssetGenerateKeyPair(), FLS_DH_KeyGen(), FLS_DeriveDh(), FLS_DH_Derive())
ECDSA	Signature generation and verification	PCT	PCT	Invoked API function returns FLR_OK	ECDSA Keypair Generation	API functions invoking keypair generation (FLS_AssetGenerateKeyPair(), FLS_DH_KeyGen(), FLS_DeriveDh(), FLS_DH_Derive())
ECDSA Signature Generation	NIST P-224 with SHA2-224	KAT	CAST	FLS_LibInitor FLS_LibSelftest	Signature Generation	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
				returns FLR_OK		
ECDSA Signature Verification	NIST P-224 with SHA2-224	KAT	CAST	FLS_LibInit or FLS_LibSelftest returns FLR_OK	Signature Verification	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
HMAC-SHA2-256		KAT	CAST	FLS_LibInit or FLS_LibSelftest returns FLR_OK	MAC	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
KAS-ECC-SSC	NIST P-224	KAT	CAST	FLS_LibInit or FLS_LibSelftest returns FLR_OK	Shared Secret Computation	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
KAS-FFC-SSC	P=2048/N=224	KAT	CAST	FLS_LibInit or FLS_LibSelftest returns FLR_OK	Shared Secret Computation	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
KDA HKDF	Feedback Mode	KAT	CAST	FLS_LibInit or FLS_LibSelftest returns FLR_OK	Key Derivation	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
KDF IKEv1		KAT	CAST	FLS_LibInit or FLS_LibSelftest returns FLR_OK	Key Derivation	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
KDF IKEv2		KAT	CAST	FLS_LibInit or FLS_LibSelftest returns FLR_OK	Key Derivation	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
KDF SP800-	Counter Mode	KAT	CAST	FLS_LibInit or FLS_LibSel	Key Derivation	Module initialization via FLS_LibInit or on

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
108 Counter				ftest returns FLR_OK		demand via FLS_LibSelfTest
KDF SP800-108 Double Pipeline	Double Pipeline Mode	KAT	CAST	FLS_LibInitor FLS_LibSelftest returns FLR_OK	Key Derivation	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
KTS-IFC Decap	2048-bit (RSA-OAEP)	KAT	CAST	FLS_LibInitor FLS_LibSelftest returns FLR_OK	Un-encapsulation	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
KTS-IFC Encap	2048-bit (RSA-OAEP)	KAT	CAST	FLS_LibInitor FLS_LibSelftest returns FLR_OK	Encapsulation	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
PBKDF	HMAC-SHA-1	KAT	CAST	FLS_LibInitor FLS_LibSelftest returns FLR_OK	MK Key Derivation	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
RSA	Signature generation and verification	PCT	PCT	Invoked API function returns FLR_OK	RSA Keypair Generation	API functions invoking keypair generation (FLS_AssetGenerateKeyPair(), FLS_DH_KeyGen(), FLS_DeriveDh(), FLS_DH_Derive())
RSA Signature Generation	2048-bit with SHA2-256 (PKCS#1v1.5)	KAT	CAST	FLS_LibInitor FLS_LibSelftest returns FLR_OK	Signature Generation	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
RSA Signature Verification	2048-bit with SHA2-256 (PKCS#1v1.5)	KAT	CAST	FLS_LibInitor FLS_LibSelftest returns FLR_OK	Signature Verification	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
SHA-1		KAT	CAST	FLS_LibInitor FLS_LibSelftest returns FLR_OK	Message Digest	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
SHA2-512		KAT	CAST	FLS_LibInitor FLS_LibSelftest returns FLR_OK	Message Digest	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
SHA3-224		KAT	CAST	FLS_LibInitor FLS_LibSelftest returns FLR_OK	Message Digest	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
SP800-90B health-tests	RCT, APT	Fault Detection Test	CAST	FLS_LibInitor FLS_LibSelftest returns FLR_OK	Health-Tests	Module initialization via FLS_LibInit or on demand via FLS_RbgReseed()
TDES-CBC Decrypt	192-bit key	KAT	CAST	FLS_LibInitor FLS_LibSelftest returns FLR_OK	Decrypt	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
TDES-CBC Encrypt	192-bit key	KAT	CAST	FLS_LibInitor FLS_LibSelftest returns FLR_OK	Encrypt	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest
TLS v1.2 KDF RFC7627		KAT	CAST	FLS_LibInitor FLS_LibSelftest returns FLR_OK	Key Derivation	Module initialization via FLS_LibInit or on demand via FLS_LibSelfTest

Table 22: Conditional Self-Tests

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.

10.3 Periodic Self-Test Information

Algorithm or Test	Test Method	Test Type	Period	Periodic Method
ECDSA SigVer (FIPS 186-4)	KAT	SW/FW Integrity	On Demand	Manually

Table 23: Pre-Operational Periodic Information

Algorithm or Test	Test Method	Test Type	Period	Periodic Method
AES-CBC Decrypt	KAT	CAST	On Demand	Manually
AES-CBC Encrypt	KAT	CAST	On Demand	Manually
AES-CCM Decrypt	KAT	CAST	On Demand	Manually
AES-CCM Encrypt	KAT	CAST	On Demand	Manually
AES-CMAC	KAT	CAST	On Demand	Manually
AES-GCM Decrypt	KAT	CAST	On Demand	Manually
AES-GCM Encrypt	KAT	CAST	On Demand	Manually
AES-XTS Decrypt	KAT	CAST	On Demand	Manually
AES-XTS Encrypt	KAT	CAST	On Demand	Manually

Algorithm or Test	Test Method	Test Type	Period	Periodic Method
AES-XTS Key Comparison	Key non-equivalency test	Critical Function	On Demand	Manually
CTR_DRBG	KAT	CAST	On Demand	Manually
DH	KAT	CAST	On Demand	Manually
DSA	PCT	PCT	On Demand	Manually
DSA Signature Generation	KAT	CAST	On Demand	Manually
DSA Signature Verification	KAT	CAST	On Demand	Manually
ECDH	KAT	CAST	On Demand	Manually
ECDSA	PCT	PCT	On Demand	Manually
ECDSA Signature Generation	KAT	CAST	On Demand	Manually
ECDSA Signature Verification	KAT	CAST	On Demand	Manually
HMAC-SHA2-256	KAT	CAST	On Demand	Manually
KAS-ECC-SSC	KAT	CAST	On Demand	Manually
KAS-FFC-SSC	KAT	CAST	On Demand	Manually
KDA HKDF	KAT	CAST	On Demand	Manually
KDF IKEv1	KAT	CAST	On Demand	Manually
KDF IKEv2	KAT	CAST	On Demand	Manually
KDF SP800-108 Counter	KAT	CAST	On Demand	Manually
KDF SP800-108 Double Pipeline	KAT	CAST	On Demand	Manually
KTS-IFC Decap	KAT	CAST	On Demand	Manually
KTS-IFC Encap	KAT	CAST	On Demand	Manually
PBKDF	KAT	CAST	On Demand	Manually
RSA	PCT	PCT	On Demand	Manually
RSA Signature Generation	KAT	CAST	On Demand	Manually
RSA Signature Verification	KAT	CAST	On Demand	Manually
SHA-1	KAT	CAST	On Demand	Manually
SHA2-512	KAT	CAST	On Demand	Manually
SHA3-224	KAT	CAST	On Demand	Manually
SP800-90B health-tests	Fault Detection Test	CAST	On Demand	Manually
TDES-CBC Decrypt	KAT	CAST	On Demand	Manually
TDES-CBC Encrypt	KAT	CAST	On Demand	Manually
TLS v1.2 KDF RFC7627	KAT	CAST	On Demand	Manually

Table 24: Conditional Periodic Information

The module does not implement periodic self-tests.

10.4 Error States

Name	Description	Conditions	Recovery Method	Indicator
Error	The module has failed a self-test and is not returning FLR_OK for API functions.	The module has failed a self-test.	FLS_LibUnit followed by FLS_LibInit	FLS_LibStatus returns FL_STATUS_ERROR

Table 25: Error States

10.5 Operator Initiation of Self-Tests

The self-tests can be invoked on-demand by unloading and re-initializing the module, or by calling the FL_LibSelfTest API function. The PCTs can be invoked on-demand by calling an API function implementing keypair generation.

10.6 Additional Information

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.

11 Life-Cycle Assurance

11.1 Installation, Initialization, and Startup Procedures

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.

11.2 Administrator Guidance

Module documentation and this security policy document contain all the guidance information required.

11.3 Non-Administrator Guidance

None.

11.4 End of Life

When the module is uninitialized all SSPs are zeroized.

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