FIPS 140-2 Security Policy

SafeZone FIPS Cryptographic Module

Rambus Global Inc., Finnish branch
Sokerilinnantie 11 C
FI-02600 Espoo
Finland
Phone: +358 50 3560966

Rambus Inc.
1050 Enterprise Way
Sunnyvale
CA 94089
United States

2020-03-21

Revision A

Software Version 1.2.0

Document Number: FIPS-2020-0113

TABLE OF CONTENTS

1 INTRODUCTION 4

1.1 PURPOSE 6
1.2 SECURITY LEVEL 6
1.3 GLOSSARY 6

2 PORTS AND INTERFACES 8

3 ROLES, SERVICES, AND AUTHENTICATION 9

3.1 ROLES AND SERVICES 9
3.1.1 User Role 9
3.1.2 Crypto-officer Role 10
3.2 AUTHENTICATION MECHANISMS AND STRENGTH 11

4 SECURE OPERATION AND SECURITY RULES 12

4.1 SECURITY RULES 12
4.2 PHYSICAL SECURITY RULES 13
4.3 SECURE OPERATION INITIALIZATION RULES 13

5 DEFINITION OF SRDIS AND MODES OF ACCESS 14

5.1 FIPS APPROVED AND ALLOWED ALGORITHMS 14
5.2 NON-FIPS MODE OF OPERATION 18
5.3 LIST OF SERVICES 21
6 CRYPTOGRAPHIC KEYS, CSPS, AND SRDIS 26
6.1 ACCESS CONTROL POLICY 32
6.2 USER GUIDE 38
6.2.1 NIST SP 800-108: Key Derivation Functions 38
6.2.2 NIST SP 800-56C Rev1: Key-Derivation Methods in Key-Establishment Schemes 38
6.2.3 HMAC-based Extract-and-Expand Key Derivation Function (HKDF) for TLS v1.3 40
6.2.4 NIST SP 800-132: Password-Based Key Derivation Function 40
6.2.5 NIST SP 800-38D: Galois/Counter Mode 40
6.2.6 NIST SP 800-38E: XTS Mode 41
6.2.7 NIST SP 800-67 Rev 2: Triple-DES Encryption 42
6.2.8 NIST SP 800-90A Rev1: Deterministic Random Bit Generator 42
6.2.9 NIST SP 800-133: Key Generation 43
6.2.10 NIST SP 800-107 Rev 1: Truncated HMAC 43
6.2.11 Support of Industry Protocols 43
6.2.12 Processor Algorithmic Acceleration 44
6.3 PORTING MAINTAINING VALIDATION 44

7 SELF TESTS 46

7.1 POWER-UP SELF-TESTS 46
7.2 CONDITIONAL SELF TESTS 47

8 MITIGATION OF OTHER ATTACKS 48

Modification History

2020-03-21	Updated operational environment according to validation comments. Added new vendor affirmed environments. Updated company info to Rambus Inc. and rebranded from Inside Secure FIPS Cryptographic Module to SafeZone FIPS Cryptographic Module.
2020-01-08	Updated according to validation comments.
2019-05-10	Updated for CMVP validation.
2018-11-28	Updated list of services available.
2018-10-25	Added HKDF and AES-GCM IV #4 rationale.
2018-09-21	Policy revision A: FIPS Lib 1.2.0 policy, based on FIPS Lib 1.1.0. Rebranding: SafeZone FIPS to Inside Secure FIPS.
2014-12-12	Policy revision A: FIPS Lib 1.1.0 policy, based on FIPS Lib 1.0.3 (A)
2014-05-12	Policy revision D: Revalidation
2014-05-07	Updated according to NIST SP 800-131A
2014-05-02	Added more vendor affirmed platforms
2014-04-25	Added several vendor affirmed platforms
2014-04-25	Added validated one platform: Samsung Galaxy Note 3 (ARMv7-a)
2013-12-31	Updated contact addresses
2013-03-15	Policy revision C: The original validation

FIPS 140-2 Security Policy

SafeZone FIPS Cryptographic Module

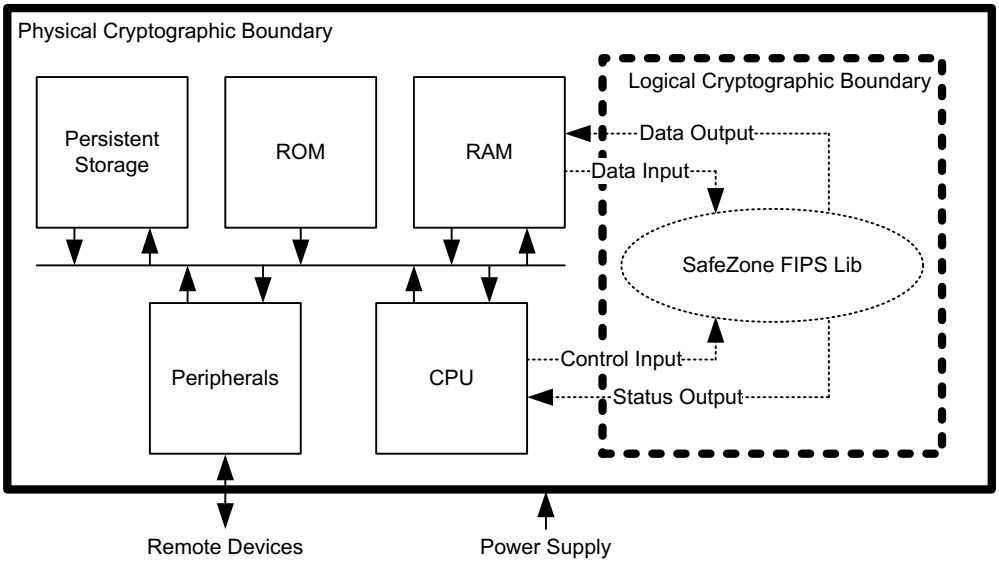
1 Introduction

SafeZone FIPS Cryptographic Module is a FIPS 140-2 Security Level 1 validated software cryptographic module from Rambus. This module is a toolkit that provides the most commonly used cryptographic primitives for a wide range of applications, including primitives needed for VPN (Virtual Private Network), TLS (Transport Layer Security), DAR (Data-At-Rest), and DRM (Digital Rights Management) clients.

SafeZone FIPS Cryptographic Module is a software-based product with a custom, small-footprint API (Application Programming Interface). The cryptographic module has been designed to provide the necessary cryptographic capabilities for other Rambus products. However, it can also be used stand-alone in custom-developed products to provide the required cryptographic functionality.

The module is primarily intended for embedded products with a general-purpose operating system.

Figure 1: SafeZone FIPS Cryptographic Module Cryptographic Boundary



For FIPS 140-2 purposes, SafeZone FIPS Cryptographic Module is classified as a multi-chip standalone cryptographic module. Within the *logical* boundary of SafeZone FIPS Cryptographic Module is the `libsafezone-sw-fips.a/so` object code library, also known as SafeZone FIPS Lib. The *physical* cryptographic boundary

of the module is the enclosure of a general-purpose computing device executing the application that embeds the SafeZone FIPS Cryptographic Module.

The SafeZone FIPS Cryptographic Module has been tested for validation on the following operational environments:

Operating System	CPU	Device	Version
Xubuntu 18.04.1 LTS	Intel Atom x5 with PAA (X86 32-bit)	Lenovo MIIX 320	1.2.0
Xubuntu 18.04.1 LTS	Intel Atom x5 without PAA (X86 32-bit)	Lenovo MIIX 320	1.2.0
Xubuntu 18.04.1 LTS	Intel Atom x5 with PAA (X86 64-bit)	Lenovo MIIX 320	1.2.0
Xubuntu 18.04.1 LTS	Intel Atom x5 without PAA (X86 64-bit)	Lenovo MIIX 320	1.2.0
Debian 9 Linux	ARM Cortex-A53 with PAA (ARMv8-a 64-bit)	ROCK64	1.2.0
Debian 9 Linux	ARM Cortex-A53 without PAA (ARMv8-a 64-bit)	ROCK64	1.2.0
Debian 9 Linux	ARM Cortex-A7 (ARMv7-a 32-bit)	Raspberry Pi 2	1.2.0

Compliance is maintained on platforms for which the binary executable remains unchanged. The module has been confirmed by the vendor to be operational on the following platforms. As allowed by the FIPS 140-2 Implementation Guidance G.5, the validation status of the Cryptographic Module is maintained when operated in the following additional operating environments:

Implementation Guidance G.5 Recompilation	
Operating System	**CPU**
Ubuntu 16.04.4 Linux	X86 Intel Core i7-4790K (64-bit)
Ubuntu 18.04.2 Linux	ARMv8-a with and without PAA (64-bit)
Debian 9 Linux	ARMv8-a with and without PAA (64-bit)
Wind River Linux 6.0	ARMv7-a (32-bit)
Android 5.0 - 5.1	ARMv7-a (32-bit)
Android 5.0 - 5.1	ARMv8-a with and without PAA (64-bit)
Android 6.0	ARMv7-a (32-bit)
Android 7.0 - 7.1	ARMv8-a with and without PAA (64-bit)
Android 8.0 - 8.1	ARMv8-a with and without PAA (64-bit)
Android 9.0	ARMv8-a with and without PAA (64-bit)
Android 10.0	ARMv8-a with and without PAA (64-bit)
Arago 2013.05	ARMv5 ARM926EJ-S (32-bit)
Nucleus 3.0	ARMv5 ARM926EJ-S (32-bit)
PhotonOS 2.0	X86 Intel Xeon (32-bit)
Ubuntu Linux 18.04	X86 Intel Xeon (32-bit)
Ubuntu Linux 18.04	X86 Intel Xeon (64-bit)

Yocto Linux 2.6	X86 Intel Xeon (64-bit)
Ubuntu Linux 18.04	X86 Intel Core (32-bit)
Ubuntu Linux 18.04	X86 Intel Core (64-bit)
Yocto Linux 2.6	X86 Intel Core (64-bit)
Rasbian Linux	ARMv7-a (32-bit)

The CMVP makes no statement as to the correct operation of the module or the security strengths of the generated keys when the specific operational environment is not listed on the validation certificate. Source code package for user performed post-validation porting of the module for these platforms is available.

1.1 Purpose

The purpose of this document is to describe the secure operation of the SafeZone FIPS Cryptographic Module including the initialization, roles, and responsibilities of operating the product in a secure, in FIPS 140 mode of operation.

1.2 Security level

The cryptographic module meets the overall requirements applicable to Level 1 security of FIPS 140-2.

Security Level	
Security Requirements Specification	**Level**
Cryptographic Module Specification	1
Module Ports and Interfaces	1
Roles, Services, and Authentication	1
Finite State Model	1
Physical Security	N/A
Operational Environment	1
Cryptographic Key Management	1
EMI/EMC	1
Self-Tests	1
Design Assurance	1
Mitigation of Other Attacks	N/A

1.3 Glossary

Term/Acronym	Description
AES	Advanced Encryption Standard
API	Application Programming Interface
CMVP	Cryptographic Module Validation Program (FIPS 140)
CSP	Critical Security Parameter
DEP	Default Entry Point
DRM	Digital Rights Management
DSS	Digital Signature Standard
EC	Elliptic Curve

FIPS	Federal Information Processing Standard
IKE	Internet Key Exchange
KEM	Key-Encapsulation Mechanism (See NIST SP 800-56B)
KTS	Key Transport Scheme
OAEP	Optimal Asymmetric Encryption Padding
PAA	Processor Algorithm Accelerator
PRF	Pseudo-Random Function
SHS	Secure Hash Standard
SRDI	Security Relevant Data Item
TLS	Transport Layer Security
Triple-DES	Triple Data Encryption Standard
VPN	Virtual Private Network

2 Ports and Interfaces

As a software-only module, the SafeZone FIPS Cryptographic Module provides an API logical interface for invocation of FIPS140-2 approved cryptographic functions. The functions shall be called by the referencing application, which assumes the operator role during application execution. The API, through the use of input parameters, output parameters, and function return values, defines the four FIPS 140-2 logical interfaces: data input, data output, control input and status output.

Logical Interfaces	API
Data Input	The data read from memory area(s) provided to the invoked function via parameters that point to the memory area(s).
Control Input	The API function invoked and function parameters designated as control inputs.
Data Output	The data written to memory area(s) provided to the invoked function via parameters that point to the memory area(s).
Status Output	The return value of the invoked API function.
Power Interface	Not accessible via the API. The power interface is used as applicable on the physical device.

3 Roles, Services, and Authentication

The SafeZone FIPS Cryptographic Module supports the *Crypto Officer* and *User* 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 Cryptographic Module does not support concurrent operators.

3.1 Roles and Services

The module does not authenticate the operator role.

3.1.1 User Role

The User role is assumed once the Crypto Officer role is finished with module initialization and explicitly switches the role using the `FL_LibEnterUserRole` API function. The User role is intended for common cryptographic use. The full list of cryptographic services available to the User role is supplied in section 5.3 of this document.

Service	Description
All services except installation, initialization, entropy source nomination, and creation of Trusted Root Key CSPs.	All standard cryptographic operations of the module, such as symmetric encryption, message authentication codes, and digital signatures. The User role may also allocate the key assets and load values for any of these cryptographic purposes. The SafeZone FIPS Cryptographic Module also provides a 'Show Status' service (API function `FL_LibStatus`) that can be used to query the current status of the cryptographic module. A macro based on `FL_LibStatus` is provided (`FL_IS_IN_APPROVED_MODE`), which returns true if the module is currently in an approved mode of operation.

3.1.2 Crypto-officer Role

The Crypto Officer role can perform all the services allowed for the User role plus a handful of additional ones. Separate from the run-time services of the module, the tasks of installing and uninstalling the module to and from the host system imply the role of a Crypto Officer. The four run-time services available only to the Crypto Officer are initializing the module for use, creating key material for Trusted Root Key CSPs, modifying the entropy source, and switching to the User role.

Service	Description
All services allowed for User role	See sections 3.1.1 and 5.3.
Initialization	Loading and preparing the module for use.
Trusted Root Key creation	Load key material into the module for local security purposes (`FL_RootKeyAllocateAndLoadValue`).
Entropy Source	Select the provider of the external entropy source. (`FL_RbgInstallEntropySource`, `FL_RbgRequestSecurityStrength`, `FL_RbgUseNonblockingEntropySource`).
Switch to the User Role	Uses the `FL_LibEnterUserRole` API function to switch to User role.
Installation	When the module is installed to a host system.
Uninstallation	When the module is removed from a host system.

3.2 Authentication Mechanisms and Strength

FIPS 140-2 Security Level 1 does not require *role-based* or *identity-based* operator authentication. The SafeZone FIPS Cryptographic Module will not authenticate the operator.

4 Secure Operation and Security Rules

In order to operate the SafeZone FIPS Cryptographic Module securely, the operator should be aware of the security rules enforced by the module and should adhere to the rules for physical security and secure operation.

4.1 Security Rules

To operate the SafeZone FIPS Cryptographic Module securely, the operator of the module must follow these instructions:

1. The operating environment that executes the SafeZone FIPS Cryptographic Module must ensure single operator mode of operation to be compliant with the requirements for the FIPS 140-2 Level 1.
2. The correct operation of the module depends on the Default Entry Point. It is not allowed to prevent execution of the Default Entry Point (the function `FLS_LibInit`).
3. The operator must not call `ptrace` or `strace` functions, or run `gdb` or other debugger when the module is in the FIPS mode.
4. If the hardware platform has a connector for an external debugger (for example JTAG), that connector must not be used while the module is in FIPS mode.
5. The SafeZone FIPS Cryptographic Module keeps all CSPs and other protected objects in Random Access Memory (RAM). The operator(s) must only use these objects via the handles provided by the SafeZone FIPS Cryptographic Module. It is not permissible to directly access these objects in the memory.
6. The operator must not call functions provided by the SafeZone FIPS Cryptographic Module that are not explicitly specified in the appropriate guidance document for User or Crypto Officer.
7. When using cryptographic services provided by the SafeZone FIPS Cryptographic Module, the operator must follow the appropriate guidance for each cryptographic algorithm. Although the cryptographic algorithms provided by the SafeZone FIPS Cryptographic Module are recommended or allowed by NIST, secure operation of these algorithms requires thorough understanding of the recommendations and appropriate limitations.
8. The SafeZone FIPS Cryptographic Module aims to be flexible and therefore it includes support for cryptographic algorithms or key lengths that were considered secure until 2013 according to NIST SP 800-131A. It is the responsibility of the SafeZone FIPS Cryptographic Module user to ensure that disallowed algorithms or key lengths are not used.
9. Some of the implemented cryptographic algorithms offer key lengths exceeding the current NIST specifications. Such key lengths must not be used, unless following newer guidance from NIST.
    a. RSA Key Pair Generation provided by the module (FIPS 186-4 B.3.3 or B.3.6) is FIPS-approved or FIPS allowed for RSA modulus sizes of 2048 bits, 3072 bits and 4096 bits. It is not permissible to generate keys using other RSA modulus sizes.

10. The Crypto Officer must ensure that the Trusted Root Key has sufficient entropy to meet all FIPS 140-2 requirements for its usage in the module.

4.2 Physical Security Rules

The physical device on which the SafeZone FIPS Cryptographic Module is executed must follow the physical security rules applicable to the purpose of the device. The SafeZone FIPS Cryptographic Module is software-based and does not provide physical security.

4.3 Secure Operation Initialization Rules

The SafeZone FIPS Cryptographic Module must be linked with an application to become executable. The software code of the module (the `libsafezone-sw-fips.a` object code library or the `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 Cryptographic Module is loaded by loading an application that links the library statically. The SafeZone FIPS Cryptographic Module is initialized automatically upon loading. On some platforms the module is implemented as a dynamically loadable module. In this case, the module is loaded as needed by the dynamic linker.

The SafeZone FIPS Cryptographic Module does not support operator authentication and thus does not require any authentication itself. The SafeZone FIPS Cryptographic Module is by default in FIPS-approved mode once initialized. Usually, the module does not require any special set-up or initialization except for installation.

5 Definition of SRDIs and Modes of Access

This chapter specifies security relevant data items (SRDIs) as well as the access control policy that is enforced by the SafeZone FIPS Cryptographic Module.

Each SRDI is held in the asset store accompanied by a security usage policy. The policy is set when the asset is allocated with `FL_RootKeyAllocateAndLoadValue`, `FL_AssetAllocate`, `FL_AssetAllocateBasic`, `FL_AssetAllocateSamePolicy` or `FL_AssetAllocateAndAssociateKeyExtra`. When the asset is accessed for use in a cryptographic operation, the policy is tested to ensure that the asset is eligible for the requested use. A policy typically consists of the allowed algorithm(s), the allowed strength of the algorithm, and the direction of the operation (encryption or decryption).

5.1 FIPS Approved and Allowed algorithms

The SafeZone FIPS Cryptographic Module implements the following FIPS-approved algorithms:

Algorithm	Implementation Details	Algorithm Certificate(s)
RSA FIPS 186-4 Signature Generation Key Pair Generation	2048, 3072 and 4096 bit keys; PKCS #1 v1.5 and PSS; SHA-224, SHA-256, SHA-384, SHA-512	C 510
RSA FIPS 186-4 Signature Validation	1024, 2048, 3072 and 4096 bit keys; PKCS #1 v1.5 and PSS	C 510
CVL RSA Private Key Primitives (NIST SP 800-56B)	2048, 3072 and 4096 bit keys; PKCS #1: RSADP and RSASP1 primitives	C 510
RSA Public Key Primitives (NIST SP 800-56B)	1024, 2048, 3072 and 4096 bit keys; PKCS #1: RSAEP and RSAVP1 primitives	
DSA FIPS 186-4 Signature Generation Domain Parameter Generation Key Pair Generation	P=2048/N=224, P=2048/N=256, P=3072/N=256; SHA-224, SHA-256, SHA-384, SHA-512	C 510

Algorithm	Implementation Details	Algorithm Certificate(s)
DSA FIPS 186-4 Signature Validation Domain Parameter Validation	P=1024/N=160, P=2048/N=224, P=2048/N=256, P=3072/N=256	C 510
ECDSA FIPS 186-4 Signature Generation Key Pair Generation	NIST P-224, P-256, P-384 and P-521 curves; SHA-224, SHA-256, SHA-384, SHA-512	C 510
ECDSA FIPS 186-4 Signature Validation Public Key Verification	NIST P-192, P-224, P-256, P-384 and P-521 curves	C 510
AES FIPS 197, NIST SP 800-38A	128, 192, 256 bit keys; ECB, CBC, CTR mode	C 510
AES CCM NIST SP 800-38C	128, 192, 256 bit keys	C 510
AES GCM / GMAC NIST SP 800-38D	128, 192, 256 bit keys; with deterministic internal IV generation, random internal IV generation, TLS v1.3 internal IV generation and SRTP internal IV generation	C 510
XTS-AES NIST SP 800-38E	256, 512 bit keys (128-bit or 256-bit encryption strength)	C 510
Triple-DES NIST SP 800-67	192 bit keys; ECB and CBC mode	C 510
CMAC NIST SP 800-38B	128, 192, 256 bit keys	C 510
HMAC FIPS 198-1	112-512 bit keys; SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	C 510
SHS FIPS 180-4	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512; BYTE only	C 510
SHA-3 FIPS 202	SHA3-224, SHA3-256, SHA3-384, SHA3-512; SHAKE-128, SHAKE-256; BYTE only	C 510

Algorithm	Implementation Details	Algorithm Certificate(s)
DRBG NIST SP 800-90A Rev 1	AES-128-CTR without df or reseed AES-256-CTR with df and reseed	C 510
CKG (NIST SP 800-133)	Key Generation	N/A, Vendor-affirmed
KTS (KEM NIST SP 800-56B)	2048, 3072, and 4096 bit keys; RSA-KEM-KWS-basic (section 9.3.3); vendor affirmed; key-wrapping; key establishment methodology provides between 112 and 150 bits of encryption strength	N/A, Vendor-affirmed
KTS (OAEP NIST SP 800-56B)	2048, 3072, and 4096 bit keys; RSA-OAEP (section 9.2.3); vendor affirmed; key-wrapping; key establishment methodology provides between 112 and 150 bits of encryption strength	N/A, Vendor-affirmed
PBKDF NIST SP 800-132	with SHA-1, SHA-256	N/A, Vendor-affirmed
KDF NIST SP 800-108	112-512 bit keys; SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, AES-CMAC; counter, feedback and double pipeline modes	C 510 Key derivation methodology provides between 112 and 256 bits of encryption strength.
CVL Application Specific Key Derivation Functions NIST SP 800-135rev1	IKEv1 Key Derivation Functions IKEv2 Key Derivation Functions TLS 1.0/1.1 Key Derivation Functions TLS 1.2 Key Derivation Functions SRTP Key Derivation Functions	C 510
Key Derivation through Extraction-then-Expansion NIST SP 800-56C Rev1	Two-Step Key Derivation with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 or AES-CMAC.	N/A; Vendor affirmed

Algorithm	Implementation Details	Algorithm Certificate(s)
CVL FFC Diffie-Hellman primitive; A part of NIST SP 800-56A	Key Agreement Primitives; 2048-8192 bit modular Diffie-Hellman groups; including FFDHE2048, FFDHE3072, FFDHE4096, FFDHE6144, FFDHE8192, MODP2048, MODP3072, MODP4096, MODP6144, MODP8192	C 510 Key establishment methodology provides between 112 and 200 bits of encryption strength.
CVL ECC CDH primitive; A part of NIST SP 800-56A	Key Agreement Primitives; NIST P-224, P-256, P-384 and P-521 curves	C 510 Key establishment methodology provides between 112 and 256 bits of encryption strength.
CVL ECC Diffie-Hellman; NIST SP 800-56A without KDF	Key Agreement Scheme; NIST P-224, P-256, P-384 and P-521 curves	C 510 Key establishment methodology provides between 112 and 256 bits of encryption strength.
KTS (NIST SP 800-38F Key Wrapping)	Key Wrapping function KW CIPH=AES; 128, 192, 256 bit keys Key Wrapping function KWP CIPH=AES; 128, 192, 256 bit keys	C 510 Key establishment methodology provides between 128 and 256 bits of encryption strength

The cryptographic module supports the following non-approved algorithms in the approved mode of operation as allowed:

Algorithm	Algorithm Type	Utilization
RSA Encryption (PKCS #1 v1.5)	Key Transport; 2048, 3072 and 4096 bit keys	(C 510) Key establishment methodology provides between 112 and 150 bits of encryption strength.
Diffie-Hellman	Key Agreement; 2048, 3072, 4096, 6144 and 8192 bit modular exponential groups.	(C 510) Vendor affirmed. Key establishment methodology provides between 112 and 200 bits of encryption strength.
EC Diffie-Hellman	Key Agreement; NIST P-224, NIST P-256, NIST P-384, NIST P-521	(C 510) Vendor affirmed. Key establishment methodology provides between 112 and 256 bits of encryption strength.
MD5	Message Digest; This function is only allowed as a part of an approved key transport scheme (e.g. TLS 1.0 or TLS 1.1).	
NDRNG; RDSEED / RDRAND, getrandom, /dev/random, /dev/urandom	Non-Approved RBGs	Entropy sources for NIST SP 800-90A Rev1 DRBG.

The SafeZone FIPS Cryptographic Module is intended for products where FIPS 140-2 approved algorithms are used. Rambus also provides solutions for customers that need various non-approved algorithms.

5.2 Non-FIPS mode of operation

In the end of 2013, some of algorithms previously allowed by the NIST were disallowed. This was because 80-bits of security was considered no longer sufficient.

See document NIST SP 800-131A for details. The SafeZone FIPS Cryptographic Module implements additional key lengths for some of these algorithms (RSA, DSA, ECDSA) for compatibility with applications previously using these key sizes. In addition, some upcoming algorithms (such as X25519) are supported by the module, but they are not allowed in FIPS Mode Operation. These and some other algorithms provided by the module are no longer allowed in approved mode of operation.

The non-FIPS validated algorithms and key sizes supported by the module are:

Algorithm	Implementation Details	Reason for algorithm being not allowed in FIPS mode.
RSA FIPS 186-2 Signature Generation	1024, 1536, 2048, 3072, and 4096 bit keys; PKCS #1 v1.5 and PSS	Transition from FIPS 186-2 to 186-4.
RSA FIPS 186-4 Signature Generation Key Pair Generation	1024 bit keys; PKCS #1 v1.5 and PSS	Key length used provides less than 112 bits of encryption strength
DSA FIPS 186-4 Signature Generation Domain Parameter Generation Key Pair Generation	P=1024/N=160	Key length used provides less than 112 bits of encryption strength
ECDSA FIPS 186-2/4 Signature Generation Key Pair Generation	NIST P-192 curve	Key length used provides less than 112 bits of encryption strength
ECDSA FIPS 186-2 Signature Generation	NIST P-224, P-256, P-384 and P-521 curves	Transition from FIPS 186-2 to 186-4.
HMAC FIPS 198-1	80-104 bit keys; SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	Key length used provides less than 112 bits of encryption strength.
KTS (KEM NIST SP 800-56B)	1024, 1536, bit keys; RSA-KEM-KWS-basic; key-wrapping	Key establishment methodology provides less than 112 bits of encryption strength
KTS (OAEP NIST SP 800-56B)	1024, 1536 bit keys; RSA-OAEP; key-wrapping	Key establishment methodology provides less than 112 bits of encryption strength

Algorithm	Implementation Details	Reason for algorithm being not allowed in FIPS mode.
KDF NIST SP 800-108	80-104 bit keys; SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, AES-CMAC; counter, feedback and double pipeline modes	Key derivation methodology provides less than 112 bits of encryption strength.
FFC Diffie-Hellman primitive;	Key Agreement Primitives; 1024 bit modular Diffie-Hellman groups	Key establishment methodology provides less than 112 bits of encryption strength.
ECC CDH primitive; A part of NIST SP 800-56A	Key Agreement Primitives; NIST P-192 curve	Key establishment methodology provides less than 112 bits of encryption strength.
ECC Diffie-Hellman; NIST SP 800-56A without KDF	Key Agreement Scheme; NIST P-192 curve	Key establishment methodology provides less than 112 bits of encryption strength.
RSA Encryption (PKCS #1 v1.5)	Key Transport; 1024, 1536 bit keys	Key establishment methodology provides less than 112 bits of encryption strength.
AES-KEY WRAP	Key Wrapping; AES-128, AES-192, AES-256	Based on old NIST specification key-wrap.pdf which has been superseded by SP 800-38F.
ChaCha20-Poly1305	Symmetric encryption and decryption	
X25519	Key Agreement Scheme	

5.3 List of services

The SafeZone FIPS provides services as specified in FIPS 140-2. The most of services are intended for FIPS 140-2 allowed mode of operation in CryptoOfficer or user role. The services correspond with function or functions with FL or FLS prefix. Functions with FLS prefix take additional *struct fls_instance* * parameter, which is

used to select which Asset Store instance is used. This API difference is not FIPS 140-2 relevant, and therefore both FL_ prefix functions and FLS_ functions are considered equivalent from FIPS 140-2 perspective.

SafeZone FIPS Cryptographic Module Service List			
Service	**FL API**	**FLS API**	**Notes**
Services which are provided in any mode			
Show Status	FL_LibStatus	FLS_LibStatus	Any mode
Show Version	FL_LibVersion	FLS_LibVersion	Any mode
Test DRBG	FL_RbgTestVector	FLS_RbgTestVector	Any mode
Check Free Space in Key Store	FL_AssetStoreStatus	FLS_AssetStoreStatus	Any mode
Obtain information on module build arguments	FL_StaticConfig		Any mode
Get/set information on current module configuration		FLS_RuntimeConfigSetProperty, FLS_RuntimeConfigGetProperty	Any mode
Check module id	FL_IntactID		Any mode
Zeroize memory area	FL_Erase	FLS_Erase	Any mode
Zeroize asset	FL_EraseAsset	FLS_EraseAsset	Any mode
Services which change mode/role			
Module Initialization (Invoked automatically upon loading the module)	FL_LibInit	FLS_LibInit	Any mode
Zeroize module state	FL_LibUnInit	FLS_LibUnInit	CO/U
Enter User Role	FL_LibEnterUserRole	FLS_LibEnterUserRole	CO
Services for Crypto-Officer and User roles			
Create Key (Setup key policy, allocate memory for key, and load key value)	FL_AssetAllocate, FL_AssetAllocateBasic, FL_AssetAllocateSamePolicy, FL_AssetAllocateAndAssociate-KeyExtra, FL_AssetLoadValue, FL_AssetLoadMultipart, FL_AssetLoadMultipartConvert-BigInt, FL_AssetPoke, FL_LocalAllocate, FL_LocalAllocateEx	FLS_AssetAllocate, FLS_AssetAllocateBasic, FLS_AssetAllocateSamePolicy, FLS_AssetAllocateAndAssociate-KeyExtra, FLS_AssetLoadValue, FLS_AssetLoadMultipart, FLS_AssetLoadMultipartConvert-BigInt, FLS_AssetPoke	CO/U
Copy Key	FL_AssetCopyValue	FLS_AssetCopyValue	CO/U
Delete Key	FL_AssetFree, FL_LocalFree	FLS_AssetFree	CO/U
Examine Key (get size or check)	FL_AssetShow, FL_AssetCheck	FLS_AssetShow, FLS_AssetCheck	CO/U
Peek Key (get key value)	FL_AssetPeek	FLS_AssetPeek	CO/U
Generate Key	FL_AssetLoadRandom	FLS_AssetLoadRandom	CO/U
Bulk Encryption/Decryption	FL_CipherInit, FL_CipherContinue, FL_CipherFinish	FLS_CipherInit, FLS_CipherContinue, FLS_CipherFinish	CO/U

SafeZone FIPS Cryptographic Module

Service List

Service	FL API	FLS API	Notes
Authenticated Encryption/Decryption with Associated Data	FL_EncryptAuthInitRandom, FL_EncryptAuthInitDeterministic, FL_CryptAuthInit, FL_CryptGcmAadContinue, FL_CryptGcmAadFinish, FL_CryptAuthContinue, FL_EncryptAuthFinish, FL_EncryptAuthPacketFinish, FL_DecryptAuthFinish	FLS_EncryptAuthInitRandom, FLS_EncryptAuthInit-Deterministic, FLS_CryptAuthInit, FLS_CryptGcmAadContinue, FLS_CryptGcmAadFinish, FLS_CryptAuthContinue, FLS_EncryptAuthFinish, FLS_EncryptAuthPacketFinish, FLS_DecryptAuthFinish	CO/U
Authenticated Encryption/Decryption with Associated Data for TLS v1.3	FL_CryptAuthInitTls13, FL_CryptAuthContinue, FL_EncryptAuthFinishTls13, FL_DecryptAuthFinishTls13, FL_EncryptAuthTls13, FL_DecryptAuthTls13	FLS_CryptAuthInitTls13, FLS_CryptAuthContinue, FLS_EncryptAuthFinishTls13, FLS_DecryptAuthFinishTls13, FLS_EncryptAuthTls13, FLS_DecryptAuthTls13	CO/U
Authenticated Encryption/Decryption with Associated Data for SRTP	FL_EncryptAuthSrtp, FL_DecryptAuthSrtp, FL_EncryptAuthSrtcp, FL_DecryptAuthSrtcp	FLS_EncryptAuthSrtp, FLS_DecryptAuthSrtp, FLS_EncryptAuthSrtcp, FLS_DecryptAuthSrtcp	CO/U
MAC Generation	FL_MacGenerateInit, FL_MacGenerateContinue, FL_MacGenerateFinish	FLS_MacGenerateInit, FLS_MacGenerateContinue, FLS_MacGenerateFinish	CO/U
MAC Verification	FL_MacVerifyInit, FL_MacVerifyContinue, FL_MacVerifyFinish	FLS_MacVerifyInit, FLS_MacVerifyContinue, FLS_MacVerifyFinish	CO/U
DRBG Random Number Generation	FL_RbgGenerateRandom	FLS_RbgGenerateRandom	CO/U
DRBG Reseeding	FL_RbgReseed	FLS_RbgReseed	CO/U
Key Derivation	FL_KeyDeriveKdk	FLS_KeyDeriveKdk	CO/U
TLS-PRF Key Derivation	FL_KeyDeriveKdk, FL_DeriveTlsPrf	FLS_KeyDeriveKdk, FLS_DeriveTlsPrf	CO/U
IKEv1 Key Derivation	FL_IKEv1ExtractSKEYID_DSA, FL_IKEv1ExtractSKEYID_PSK, FL_IKEv1ExtractSKEYID_PKE, FL_IKEv1DeriveKeyingMaterial	FLS_IKEv1ExtractSKEYID_DSA, FLS_IKEv1ExtractSKEYID_PSK, FLS_IKEv1ExtractSKEYID_PKE, FLS_IKEv1DeriveKeyingMaterial	CO/U
IKEv2 Key Derivation	FL_IKEv2ExtractSKEYSEED, FL_IKEv2ExtractSKEYSEED-rekey, FL_IKEv2DeriveDKM	FLS_IKEv2ExtractSKEYSEED, FLS_IKEv2ExtractSKEYSEED-rekey, FLS_IKEv2DeriveDKM	CO/U
IKEv1/IKEv2 KDF (Common)	FL_IkePrfExtract	FLS_IkePrfExtract	CO/U
NIST SP 800-56C Rev1	FL_HkdfExtract, FL_HkdfExpandAsset, FL_HkdfExpand	FLS_HkdfExtract, FLS_HkdfExpandAsset, FLS_HkdfExpand	CO/U
HKDF Key Derivation / TLS 1.3 PRF	FL_Hkdf	FLS_Hkdf	CO/U
SRTP Key Derivation	FL_SrtpKeyDerive	FLS_SrtpKeyDerive	CO/U
AES Key Wrapping	FL_AssetsWrapAes38F	FLS_AssetsWrapAes38F	CO/U
AES Key Unwrapping	FL_AssetsUnwrapAes38F	FLS_AssetsUnwrapAes38F	CO/U
AES Data Wrapping	FL_CryptKw	FLS_CryptKw	CO/U
AES Data Unwrapping	FL_CryptKw	FLS_CryptKw	CO/U
Trusted Root Key Derivation	FL_TrustedKdkDerive, FL_TrustedKekdkDerive	FLS_TrustedKdkDerive, FLS_TrustedKekdkDerive	CO/U
Trusted KDK Key Derivation	FL_TrustedKeyDerive	FLS_TrustedKeyDerive	CO/U

SafeZone FIPS Cryptographic Module

Service List

Service	FL API	FLS API	Notes
Trusted Key Wrapping	FL_AssetWrapTrusted	FLS_AssetWrapTrusted	CO/U
Trusted Key Unwrapping	FL_AssetUnwrapTrusted	FLS_AssetUnwrapTrusted	CO/U
PBKDF2 Key Derivation	FL_KeyDerivePbkdf2	FLS_KeyDerivePbkdf2	CO/U
DSA/Diffie-Hellman Domain Parameter and Key Pair Generation	FL_AssetGenerateKeyPair	FLS_AssetGenerateKeyPair	CO/U
Signature Generation	FL_HashSignFips186, FL_HashSignPkcs1, FL_HashSignPkcs1Pss	FLS_HashSignFips186, FLS_HashSignPkcs1, FLS_HashSignPkcs1Pss	CO/U
RSA Signature Generation, Primitive Only	FL_Pkcs1RSASP1	FLS_Pkcs1RSASP1	CO/U
RSA Decryption Primitive Only	FL_Pkcs1RSADP	FLS_Pkcs1RSADP	CO/U
RSA-KEM Key Unwrapping	FL_AssetsUnwrapRsaKem	FLS_AssetsUnwrapRsaKem	CO/U
RSA-OAEP Key Unwrapping	FL_AssetsUnwrapRsaOaep	FLS_AssetsUnwrapRsaOaep	CO/U
RSA-PKCS#1v1.5 Key Unwrapping	FL_AssetsUnwrapPkcs1v15	FLS_AssetsUnwrapPkcs1v15	CO/U
Diffie-Hellman Key Agreement	FL_DeriveDh	FLS_DeriveDh	CO/U
Elliptic Curve Diffie-Hellman Key Agreement	FL_DeriveDh	FLS_DeriveDh	CO/U
Diffie-Hellman Key Generation Appendix D	FL_DH_KeyGen		CO/U
Diffie-Hellman Derivation Appendix D	FL_DH_Derive		CO/U
Public Key Validation	FL_AssetCheck	FLS_AssetCheck	CO/U
DSA/Diffie-Hellman Domain Parameter Verification	FL_AssetCheck	FLS_AssetCheck	CO/U
Signature Verification	FL_HashVerifyFips186, FL_HashVerifyPkcs1, FL_HashVerifyRecoverPkcs1, FL_HashVerifyPkcs1Pss	FLS_HashVerifyFips186, FLS_HashVerifyPkcs1, FLS_HashVerifyRecoverPkcs1, FLS_HashVerifyPkcs1Pss	CO/U
RSA Signature Verification / Primitive Only	FL_Pkcs1RSAVP1	FLS_Pkcs1RSAVP1	CO/U
RSA-KEM Key Wrapping	FL_AssetsWrapRsaKem	FLS_AssetsWrapRsaKem	CO/U
RSA-OAEP Key Wrapping	FL_AssetsWrapRsaOaep	FLS_AssetsWrapRsaOaep	CO/U
RSA PKCS#1v1.5 Key Wrapping	FL_AssetsWrapPkcs1v15	FLS_AssetsWrapPkcs1v15	CO/U
RSA Encryption Primitive Only	FL_Pkcs1RSAEP	FLS_Pkcs1RSAEP	CO/U
On-demand selftest	FL_LibSelfTest	FLS_LibSelfTest	CO/U
Digest Computation	FL_HashInit, FL_HashContinue, FL_HashFinish, FL_HashFinishKeep, FL_HashSingle	FLS_HashInit, FLS_HashContinue, FLS_HashFinish, FLS_HashFinishKeep, FLS_HashSingle	CO/U
Load Precomputed Digest	FL_LoadFinishedHashStateAlgo	FLS_LoadFinishedHashStateAlgo	CO/U
Create new FLS operating context	FL_New	FLS_New	CO/U
Delete FLS operating context	FL_Free	FLS_Free	CO/U
Services requiring Crypto-Officer role			
Entropy Source Installation	FL_RbgInstallEntropySource, FL_RbgRequestSecurityStrength, FL_RbgUseNonblockingEntropy-Source	FLS_RbgInstallEntropySource, FLS_RbgRequestSecurityStrength, FLS_RbgUseNonblocking-EntropySource	CO
Create Trusted Root Key	FL_RootKeyAllocateAndLoad-Value	FLS_RootKeyAllocateAndLoad-Value	CO

SafeZone FIPS Cryptographic Module

Service List

Service	FL API	FLS API	Notes
Services only for non-FIPS mode of operation			
AES-KEY WRAP wrapping	FL_AssetsWrapAes	FLS_AssetsWrapAes	Non-FIPS
AES-KEY WRAP unwrapping	FL_AssetsUnwrapAes	FLS_AssetsUnwrapAes	Non-FIPS
CHACHA20-POLY1305 authenticated encryption	FL_Chacha20Poly1305IetfInit, FL_Chacha20Poly1305IetfClear, FL_Chacha20Poly1305Ietf-EncryptDetached, FL_Chacha20Poly1305Ietf-DecryptDetached, FL_Chacha20Poly1305Ietf-Encrypt, FL_Chacha20Poly1305Ietf-Decrypt		Non-FIPS
X25519 key agreement	FL_X25519_KeyGen, FL_X25519_Derive		Non-FIPS

6 Cryptographic Keys, CSPs, and SRDIs

While operating in a FIPS-compliant manner, the asset store within the module may contain the following security relevant data items (depending on which keys will be used by the user):

ID	Algorithm	Size	Description	Origin	Storage	Zeroization Method
General Keys/CSPs						
AES Encryption Key	AES including modes ECB, CBC, and CTR	128, 192, 256 bits	Key created for the purposes of encrypting and/or decrypting data using AES algorithm	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
AES CCM Encryption Key	AES CCM	128, 192, 256 bits	Key created for the purposes of authenticated encryption and/or decryption of data using AES and CCM algorithms	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
AES GCM Encryption Key (or AES GMAC Key)	AES GCM, AES GMAC	128, 192, 256 bits	Key created for the purposes of authenticated encryption and/or decryption of data using AES and GCM/GMAC algorithms	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
XTS-AES Encryption Key	XTS-AES	256, 512 bits	Key created for the purposes of encrypting and/or decrypting data using AES algorithm in XTS mode	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
Triple-DES Encryption Key	Triple-DES	192 bits	Key created for the purposes of encrypting and/or decrypting data using Triple-DES algorithm	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
CMAC Key	CMAC + AES	128, 192, 256 bits	Key created for the purposes of generating and verifying CMAC authentication codes	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
CMAC Verify Key	CMAC + AES	128, 192, 256 bits	Key created for the purpose of verifying CMAC authentication codes	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset

ID	Algorithm	Size	Description	Origin	Storage	Zeroization Method
KDF Key Derivation key	NIST SP 800-108 + HMAC or CMAC	112-512 bits	Key created for the purpose of deriving other keys as specified in NIST SP 800-108 or IKEv1/IKEv2 key derivation specified in NIST SP 800-135.	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
TLS-PRF Key Derivation Key	NIST SP 800-135	112-512 bits	Key created for the purpose of key derivation using TLS1.0/TLS1.2 key derivation function presented in NIST SP 800-135.	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
HMAC Key	HMAC + SHS	112-512 bits	Key created for the purposes of generating and verifying HMAC authentication codes	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
HMAC Verify Key	HMAC + SHS	112-512 bits	Key created for the purpose of verifying HMAC authentication codes	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
RSA Signing Key	RSA Private Key (CRT)	2048, 3072, 4096 bits (modulus size)	Private key for the purpose of signing data using RSA with PKCS #1v1.5 or PSS padding.	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
DSA Signing Key	DSA Private Key	P=2048/N=224, P=2048/N=256, P=3072/N=256	Private key for the purpose of signing data using DSA algorithm. Includes associated domain parameters.	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
ECDSA Signing Key	ECDSA Private Key	P-224, P-256, P-384, P-521	Private key for the purpose of signing data using ECDSA algorithm	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
AES Key-Wrapping Key	AES	128, 192, 256 bits	Key created for the purposes of data or key wrapping and unwrapping using NIST SP 800-38F algorithm	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset

ID	Algorithm	Size	Description	Origin	Storage	Zeroization Method
Diffie-Hellman Private Value	Diffie-Hellman	P=2048/N=224, P=2048/N=256, P=3072/N=256, MODP or ffdhe 2048, 3072, 4096, 6144, or 8192	Private value for the purpose of key agreement using Diffie-Hellman algorithm. Includes associated domain parameters.	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
EC Diffie-Hellman Private Value	EC Diffie-Hellman	P-224, P-256, P-384, P-521	Private value for the purpose of key agreement using Elliptic Curve Diffie-Hellman algorithm.	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
KTS (KEM) Unwrapping Key	RSA Private Key (CRT)	2048, 3072, 4096 bits	Private key for the purpose of transporting keys using RSA with KEM as specified in NIST SP 800-56B	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
KTS (OAEP) Unwrapping Key	RSA Private Key (CRT)	2048, 3072, 4096 bits	Private key for the purpose of transporting keys using RSA with OAEP as specified in NIST SP 800-56B	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
KTS (PKCS #1 v1.5) RSA Unwrapping Key	RSA Private Key (CRT)	2048, 3072, 4096 bits	Private key for the purpose of transporting keys using RSA with PKCS #1 v1.5 padding (also known as RSA Encryption)	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
Trusted Keys						
Trusted Root Key	NIST SP 800-108 KDF	256 bits	Key used for deriving other keys as per NIST SP 800-108. Can only derive 'Trusted KDK' and 'Trusted KEKDK' keys.	Crypto Officer	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
Trusted KDK	NIST SP 800-108 KDF	256 bits	Key used for deriving other keys as per NIST SP 800-108.	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset

ID	Algorithm	Size	Description	Origin	Storage	Zeroization Method
Trusted KEKDK	NIST SP 800-108 KDF + AES (Key Wrap)	256 bits	Key used for wrapping keys with combination of NIST SP800-108 KDF and AES Key Wrap.	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
Other CSPs						
DRBG CTR-128 entropy input string and the seed	CTR_DRBG 128-bits	256 bits	Entropy input materials	Entropy source	Plaintext in RAM	Power Off, FL_LibUnInit
DRBG CTR-128 state: Key	CTR_DRBG 128-bits	128 bits	Key for DRBG used for random number and key/key pair generation purposes.	Entropy source	Plaintext in RAM	Power Off, FL_LibUnInit
DRBG CTR-128 state: V	CTR_DRBG 128-bits	128 bits	V value for DRBG used for random number and key/key pair generation purposes.	Entropy source	Plaintext in RAM	Power Off, FL_LibUnInit
DRBG CTR-256 entropy input string and the seed	CTR_DRBG 256-bits with derivation function	256-1024 bits	Entropy input materials	Entropy source	Plaintext in RAM	Power Off, FL_LibUnInit
DRBG CTR-256 state: Key	CTR_DRBG 256-bits with derivation function	256 bits	Key for DRBG used for random number and key/key pair generation purposes.	Entropy source	Plaintext in RAM	Power Off, FL_LibUnInit
DRBG CTR-256 state: V	CTR_DRBG 256-bits with derivation function	128 bits	V value for DRBG used for random number and key/key pair generation purposes.	Entropy source	Plaintext in RAM	Power Off, FL_LibUnInit
PBKDF password	NIST SP 800-132	varies (at least 12 characters)	password or passphrase	Crypto Officer, User	Plaintext in RAM	Power Off, FL_Erase

ID	Algorithm	Size	Description	Origin	Storage	Zeroization Method
Public Keys						
Software Integrity Public Key	ECDSA / Verify	NIST P-224	Public key used by Power-on Software Integrity to ensure the integrity of the Cryptographic Module.	Embedded in the software	Plaintext in persistent storage	none
RSA Verification Key	RSA Public Key	1024, 2048, 3072, 4096 bits modulus size	Public key for the purpose of verifying signed data using RSA with PKCS #1 v1.5 or PSS padding. Not considered sensitive or CSP.	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
DSA Verification Key	DSA Public Key	P=1024/N=160, P=2048/N=224, P=2048/N=256, P=3072/N=256	Public key for the purpose of verifying signed data using DSA algorithm. Includes associated domain parameters. Not considered sensitive or CSP.	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
ECDSA Verification Key	ECDSA Public Key	P-192, P-224, P-256, P-384, P-521	Public key for the purpose of verifying signed data using ECDSA algorithm. Not considered sensitive or CSP.	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
Diffie-Hellman Public Value	Diffie-Hellman	P=2048/N=224, P=2048/N=256, P=3072/N=256, MODP or ffdhe 2048, 3072, 4096, 6144, or 8192	Public value for the purpose of key agreement using the Diffie-Hellman algorithm. Includes associated domain parameters. Not considered sensitive or CSP.	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
EC Diffie-Hellman Public Value	EC Diffie-Hellman	P-224, P-256, P-384, P-521	Public value for the purpose of key agreement using the Elliptic Curve Diffie-Hellman algorithm. Not considered sensitive or CSP.	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset

ID	Algorithm	Size	Description	Origin	Storage	Zeroization Method
KTS (KEM) Wrapping Key	RSA Public Key	2048, 3072, 4096 bits	Public key for the purpose of transporting keys using RSA with KEM as specified in NIST SP 800-56B. Not considered sensitive or CSP.	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
KTS (OAEP) Wrapping Key	RSA Public Key	2048, 3072, 4096 bits	Public key for the purpose of transporting keys using RSA with OAEP as specified in NIST SP 800-56B. Not considered sensitive or CSP.	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset
KTS (PKCS #1 v1.5) RSA Wrapping Key	RSA Public Key	2048, 3072, 4096 bits	Public key for the purpose of transporting keys using RSA with PKCS #1 v1.5 padding (also known as RSA Encryption). Not considered sensitive or CSP.	Crypto Officer, User	Plaintext in RAM	Power-off, FL_AssetFree, FL_LibUnInit, FL_EraseAsset

All the cryptographic keys and other security relevant materials handled by the module can be zeroized by using the cryptographic module, with the exception of the Software Integrity Public Key that is used in the self-test to validate the module.

There are three ways to zeroize a key: individual keys can be explicitly zeroized using the `FL_AssetFree` function call, all keys (except dynamically or preallocated allocated assets) are zeroized once the module is uninitialized (`FL_LibUnInit`) or encounters error state, and (as all the keys handled by the module except the Software Integrity Public key are stored in RAM memory), the keys can also be zeroized by turning the power off. Further keys and memory may be cleared with zeroization convenience functions (`FL_Erase, FL_EraseAsset`).

The main difference between normal and Trusted Keys is that Trusted Keys do not allow the User role to pick the key material to use, but the keys can only be derived from the trusted root key provided by the Crypto Officer role. The primary use of trusted keys is wrapping and unwrapping other keys for purposes of persistent storage outside the SafeZone FIPS Cryptographic Module. Trusted Keys do not provide any additional security for FIPS purposes. They merely are identifiers for the keys derived from the trusted root key.

6.1 Access Control Policy

The module allows controlled access to the SRDIs contained within it. The following table defines the access that an operator or an application has to each SRDI while operating the SafeZone FIPS Cryptographic Module in a given role performing a specific service (command). The permissions are categorized as a set of four separate permissions: read [R] (the SRDI can be read by this operation), write [W] (the SRDI can be written by this operation), execute [X] (the SRDI can be used in this operation), and delete [D] (the SRDI will be zeroized by this operation). If no permission is listed, then an operator outside the SafeZone FIPS Cryptographic Module has no access to the SRDI.

The operations are presented in the following tables: for secret keys, private keys, public keys, and none (operations which do not affect any of SRDI). The operations which are not appropriate for a specific key type have been omitted.

SafeZone FIPS Cryptographic Module SRDI/Role/Service Access Policy **Secret Keys**	Security Relevant Data Item	AES Encryption Key	AES CCM Encryption Key	AES GCM/GMAC Key	XTS-AES Encryption Key	Triple-DES Encryption Key	CMAC Key	CMAC Verify Key	KDF Key Derivation key	TLS-PRF Key Derivation key	HMAC Key	HMAC Verify Key	AES Key-Wrapping Key	Trusted Root Key	Trusted KDK	Trusted KEKDK	DRBG state: Key / V	DRBG entropy input / seed	PBKDF password
Role/Service																			
User role or Crypto Officer Role																			
Zeroize (FL_LibUnInit, FL_Erase, FL_EraseAsset)		D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Create Key (FL_AssetAllocate, FL_AssetAllocateBasic, FL_AssetAllocateSamePolicy, FL_AssetAllocateAndAssociateKeyExtra, FL_AssetLoadValue, FL_AssetLoadMultipart, FL_AssetLoadMultipartConvertBigInt, FL_AssetPoke, FL_LocalAllocate, FL_LocalAllocateEx)		W	W	W	W	W	W	W	W	W	W	W	W						
Copy Key (FL_AssetCopyValue)		W	W	W	W	W	W	W	W	W	W	W	W						
Delete Key (FL_AssetFree, FL_LocalFree)		D	D	D	D	D	D	D	D	D	D	D	D		D	D			
Examine Key (FL_AssetShow, FL_AssetCheck)																			
Peek Key (FL_AssetPeek)		R	R	R	R	R	R	R	R	R	R	R	R						
Generate Key (FL_AssetLoadRandom)		W	W	W	W	W	W	W	W	W	W	W	W				XW	XW	
Bulk Encryption/Decryption (FL_CipherInit, FL_CipherContinue, FL_CipherFinish)		X			X	X													

SafeZone FIPS Cryptographic Module SRDI/Role/Service Access Policy **Secret Keys**	Security Relevant Data Item	AES Encryption Key	AES CCM Encryption Key	AES GCM/GMAC Key	XTS-AES Encryption Key	Triple-DES Encryption Key	CMAC Key	CMAC Verify Key	KDF Key Derivation key	TLS-PRF Key Derivation key	HMAC Key	HMAC Verify Key	AES Key-Wrapping Key	Trusted Root Key	Trusted KDK	Trusted KEKDK	DRBG state: Key / V	DRBG entropy input / seed	PBKDF password
Role/Service																			
Authenticated Encryption/Decryption with Associated Data (FL_EncryptAuthInitRandom, FL_EncryptAuthInitDeterministic, FL_CryptAuthInit[1], FL_CryptGcmAadContinue, FL_CryptGcmAadFinish, FL_CryptAuthContinue, FL_EncryptAuthFinish, FL_EncryptAuthPacketFinish, FL_DecryptAuthFinish)			X	X															
Authenticated Encryption/Decryption with Associated Data for TLS v1.3 (FL_CryptAuthInitTls13, FL_CryptAuthContinue, FL_EncryptAuthFinishTls13, FL_DecryptAuthFinishTls13, FL_EncryptAuthTls13, FL_DecryptAuthTls13)			X	X															
Authenticated Encryption/Decryption with Associated Data for SRTP (FL_EncryptAuthSrtp, FL_DecryptAuthSrtp, FL_EncryptAuthSrtcp, FL_DecryptAuthSrtcp)			X	X															
MAC Generation (FL_MacGenerateInit, FL_MacGenerateContinue, FL_MacGenerateFinish)							X				X								
MAC Verification (FL_MacVerifyInit, FL_MacVerifyContinue, FL_MacVerifyFinish)							X	X			X	X							
DRBG Random Number Generation (FL_RbgGenerateRandom)																	XW	XW	
DRBG Reseeding (FL_RbgReseed)																	XW	XW	
Key Derivation (FL_KeyDeriveKdk)		W	W	W	W	W	W	W	XW	W	W	W	W						
TLS-PRF Key Derivation (FL_KeyDeriveKdk, FL_DeriveTlsPrf)		W	W	W	W	W	W	W	W	XW	W	W	W						
IKEv1 Key Derivation (FL_IKEv1ExtractSKEYID_DSA, FL_IKEv1ExtractSKEYID_PSK, FL_IKEv1ExtractSKEYID_PKE, FL_IKEv1DeriveKeyingMaterial)		W	W	W	W	W	W	W	XW	W	W	W	W						
IKEv2 Key Derivation (FL_IKEv2ExtractSKEYSEED, FL_IKEv2ExtractSKEYSEEDrekey, FL_IKEv2DeriveDKM)		W	W	W	W	W	W	W	XW	W	W	W	W						

[1] Function may only be used to begin AES-CCM encryption operation or to continue multipacket operation with deterministic IV. In particular, the function shall not be used to initialize AES-GCM encryption, except when IV has been constructed in FIPS 140-2 Implementation Guidance A.5 approved manner by an FIPS 140-2 approved module.

SafeZone FIPS Cryptographic Module SRDI/Role/Service Access Policy **Secret Keys**	Security Relevant Data Item	AES Encryption Key	AES CCM Encryption Key	AES GCM/GMAC Key	XTS-AES Encryption Key	Triple-DES Encryption Key	CMAC Key	CMAC Verify Key	KDF Key Derivation key	TLS-PRF Key Derivation key	HMAC Key	HMAC Verify Key	AES Key-Wrapping Key	Trusted Root Key	Trusted KDK	Trusted KEKDK	DRBG state: Key / V	DRBG entropy input / seed	PBKDF password
Role/Service																			
IKEv1/IKEv2 KDF (Common) (FL_IkePrfExtract)										X									
NIST SP 800-56C Rev1 (FL_HkdfExtract, FL_HkdfExpandAsset, FL_HkdfExpand)		W	W	W	W	W	W	W	XW	W	W	W	W						
HKDF Key Derivation / TLS 1.3 PRF (FL_Hkdf)		W	W	W	W	W	W	W	XW	W	W	W	W						
SRTP Key Derivation (FL_SrtpKeyDerive)		X																	
AES Key Wrapping (FL_AssetsWrapAes38F)		R	R	R	R	R	R	R	R	R	R	R	XR						
AES Key Unwrapping (FL_AssetsUnwrapAes38F)		W	W	W	W	W	W	W	W	W	W	W	XW						
AES Data Wrapping (FL_CryptKw)													X						
AES Data Unwrapping (FL_CryptKw)													X						
Trusted Root Key Derivation (FL_TrustedKdkDerive, FL_TrustedKekdkDerive)														X	W	W			
Trusted KDK Key Derivation (FL_TrustedKeyDerive)		W	W	W	W	W	W	W	W	W	W	W	W		X				
Trusted Key Wrapping (FL_AssetWrapTrusted)		R	R	R	R	R	R	R	R	R	R	R	R			X			
Trusted Key Unwrapping (FL_AssetUnwrapTrusted)		W	W	W	W	W	W	W	W	W	W	W	W			X			
PBKDF2 Key Derivation (FL_KeyDerivePbkdf2)		W	W	W	W	W	W	W	W	W	W	W	W						RW
Crypto-officer Role																			
Entropy Source Installation (FL_RbgInstallEntropySource, FL_RbgRequestSecurityStrength, FL_RbgUseNonblockingEntropySource)																		W	
Create Trusted Root Key (FL_RootKeyAllocateAndLoadValue)														W					
Enter User Role (FL_LibEnterUserRole)																			

SafeZone FIPS Cryptographic Module

SRDI/Role/Service Access Policy

Private Keys

Role/Service	Security Relevant Data Item	RSA Signing Key	DSA Signing Key	ECDSA Signing Key	Diffie-Hellman Private Value	EC Diffie-Hellman Private Value	KTS (KEM) Unwrapping Key	KTS (OAEP) Unwrapping Key	KTS (PKCS #1 v1.5, other) RSA Unwrapping Key	DRBG state: Key / V	DRBG entropy input / seed
User role or Crypto Officer Role											
Zeroize (FL_LibUnInit, FL_Erase, FL_EraseAsset)		D	D	D	D	D	D	D	D	D	D
Create Key (FL_AssetAllocate, FL_AssetAllocateBasic, FL_AssetAllocateSamePolicy, FL_AssetAllocateAndAssociateKeyExtra, FL_AssetLoadValue, FL_AssetLoadMultipart, FL_AssetLoadMultipartConvertBigInt, FL_AssetPoke, FL_LocalAllocate, FL_LocalAllocateEx)		W	W	W	W	W	W	W	W		
Copy Key (FL_AssetCopyValue)		W	W	W	W	W	W	W	W		
Delete Key (FL_AssetFree, FL_LocalFree)		D	D	D	D	D	D	D	D		
Examine Key (FL_AssetShow, FL_AssetCheck)											
Peek Key (FL_AssetPeek)		R	R	R	R	R	R	R	R		
Generate Key (FL_AssetLoadRandom)										XW	XW
Generate Key Pair (FL_AssetGenerateKeyPair)		W	W	W	W	W	W	W	W	XW	XW
DSA/Diffie-Hellman Domain Parameter and Key Pair Generation (FL_AssetGenerateKeyPair)			W		W					XW	XW
Signature Generation (FL_HashSignFips186, FL_HashSignPkcs1, FL_HashSignPkcs1Pss)		X	X	X						XW	XW
RSA Signature Generation, Primitive Only (FL_Pkcs1RSASP1)		X									
RSA Decryption Primitive Only (FL_Pkcs1RSADP)									X		
AES Key Wrapping (FL_AssetsWrapAes38F)		R	R	R	R	R	R	R	R		
AES Key Unwrapping (FL_AssetsUnwrapAes38F)		W	W	W	W	W	W	W	W		
RSA Key Wrapping (FL_AssetsWrapPkcs1v15, FL_AssetsWrapRsaKem, FL_AssetsWrapRsaOaep)		R	R	R	R	R	R	R	R		
RSA-KEM Key Unwrapping (FL_AssetsUnwrapRsaKem)		W	W	W	W	W	WX	W	W		
RSA-OAEP Key Unwrapping (FL_AssetsUnwrapRsaOaep)		W	W	W	W	W	W	WX	W		
RSA-PKCS#1v1.5 Key Unwrapping (FL_AssetsUnwrapPkcs1v15)		W	W	W	W	W	W	W	WX		
Trusted Key Wrapping (FL_AssetWrapTrusted)		R	R	R	R	R	R	R	R		
Trusted Key Unwrapping (FL_AssetUnwrapTrusted)		W	W	W	W	W	W	W	W		
Diffie-Hellman Key Agreement (FL_DeriveDh)					X						
Elliptic Curve Diffie-Hellman Key Agreement (FL_DeriveDh)						X					
Diffie-Hellman Key Generation Appendix D (FL_DH_KeyGen)					X						
Diffie-Hellman Derivation Appendix D (FL_DH_Derive)					X						

SafeZone FIPS Cryptographic Module SRDI/Role/Service Access Policy **Public Keys**	Security Relevant Data Item	Software Integrity Public Key	RSA Verification Key	DSA Verification Key	ECDSA Verification Key	Diffie-Hellman Public Value	EC Diffie-Hellman Public Value	KTS (KEM) Wrapping Key	KTS (OAEP) Wrapping Key	KTS (PKCS #1 v1.5, other) RSA Wrapping Key	DRBG state: Key / V	DRBG entropy input / seed
Role/Service												
User role or Crypto-Officer Role												
Zeroize (FL_LibUnInit, FL_Erase, FL_EraseAsset)			D	D	D	D	D	D	D	D	D	D
On-demand self-test (FL_LibSelfTest)		X										
Create Key (FL_AssetAllocate, FL_AssetAllocateBasic, FL_AssetAllocateSamePolicy, FL_AssetAllocateAndAssociateKeyExtra, FL_AssetLoadValue, FL_AssetLoadMultipart, FL_AssetLoadMultipartConvertBigInt, FL_AssetPoke, FL_LocalAllocate, FL_LocalAllocateEx)			W	W	W	W	W	W	W	W		
Copy Key (FL_AssetCopyValue)			W	W	W	W	W	W	W	W		
Delete Key (FL_AssetFree, FL_LocalFree)			D	D	D	D	D	D	D	D		
Examine Key (FL_AssetShow, FL_AssetCheck)			RX	RX	RX	RX	RX	RX	RX	RX		
Peek Key (FL_AssetPeek)			R	R	R	R	R	R	R	R		
Generate Key Pair (FL_AssetGenerateKeyPair)			W	W	W	W	W	W	W	W	XW	XW
DSA/Diffie-Hellman Domain Parameter and Key Pair Generation (FL_AssetGenerateKeyPair)				W		W					XW	XW
Public Key Validation (FL_AssetCheck)			X	X	X	X	X	X	X	X		
DSA/Diffie-Hellman Domain Parameter Verification (FL_AssetCheck)				X		X						
Signature Verification (FL_HashVerifyFips186, FL_HashVerifyPkcs1, FL_HashVerifyRecoverPkcs1, FL_HashVerifyPkcs1Pss)			X	X	X							
RSA Signature Verification / Primitive Only (FL_Pkcs1RSAVP1)			X									
RSA-KEM Key Wrapping (FL_AssetsWrapRsaKem)			R	R	R	R	R	RX	R	R		
RSA-OAEP Key Wrapping (FL_AssetsWrapRsaOaep)			R	R	R	R	R	R	RX	R		
RSA PKCS#1v1.5 Key Wrapping (FL_AssetsWrapPkcs1v15)			R	R	R	R	R	R	R	RX		
RSA Encryption Primitive Only (FL_Pkcs1RSAEP)										X		
Diffie-Hellman Key Agreement (FL_DeriveDh)						X						
Diffie-Hellman Ephemeral Key Generation Appendix D (FL_DH_KeyGen)						X						
Diffie-Hellman Key Agreement Appendix D (FL_DH_Derive)						X						
Elliptic Curve Diffie-Hellman Key Agreement (FL_DeriveDh)							X					
Crypto-officer Role												
Module Initialization (FL_LibInit/FLS_LibInit) (This function is automatically invoked upon loading the module)		X									XW	XW

SafeZone FIPS Cryptographic Module

SRDI/Role/Service Access Policy

Services not using any SRDI

Role/Service
User role or Crypto Officer Role
Show Status (FL_LibStatus)
Digest Computation (FL_HashInit, FL_HashContinue, FL_HashFinish, FL_HashFinishKeep, FL_HashSingle)
Load Precomputed Digest (FL_LoadFinishedHashStateAlgo)
Create new FLS operating context / delete FLS operating context (FL_New, FL_Free)
Zeroize memory area (FL_Erase)

SafeZone FIPS Cryptographic Module

SRDI/Role/Service Access Policy

Services allowed in any state (including error state)

Role/Service
Services which are provided in any state
Show Version (FL_LibVersion)
Test DRBG (FL_RbgTestVector)
Check Free Space in Key Store (FL_AssetStoreStatus)
Obtain information on module build arguments (FL_StaticConfig)
Get/set information on current module configuration (FLS_RuntimeConfigSetProperty, FLS_RuntimeConfigGetProperty)
Check module id (FL_IntactID)
Zeroize memory area (FL_Erase)
Zeroize asset (FL_EraseAsset)
Zeroize module state (FL_LibUnInit)

SafeZone FIPS Cryptographic Module

SRDI/Role/Service Access Policy

Non-FIPS 140-2 Services

Role/Service
Additional services which can be used only in non-FIPS mode of operation
AES-KEY WRAP wrapping (FL_AssetsWrapAes)
AES-KEY WRAP unwrapping (FL_AssetsUnwrapAes)
CHACHA20-POLY1305 authenticated encryption (FL_Chacha20Poly1305IetfInit, FL_Chacha20Poly1305IetfClear, FL_Chacha20Poly1305IetfEncryptDetached, FL_Chacha20Poly1305IetfDecryptDetached, FL_Chacha20Poly1305IetfEncrypt, FL_Chacha20Poly1305IetfDecrypt)
X25519 key agreement (FL_X25519_KeyGen, FL_X25519_Derive)

Most of the functions in the SafeZone FIPS Cryptographic module have alternative invocation. The alternative invocation uses FLS prefix instead of FL.

6.2 User Guide

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

6.2.1 NIST SP 800-108: Key Derivation Functions

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

6.2.2 NIST SP 800-56C Rev1: Key-Derivation Methods in Key-Establishment Schemes

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

There are following rules the user of the functions for NIST SP 800-56C Key Derivation functions shall observe:

- Key derived using NIST SP 800-56C rev 1 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 requires a key. This key corresponds to salt in NIST SP 800-56C rev 1. If salt is to be omitted, use all-zero byte key at is 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 Rev1 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 (with FL_AssetFree or FLS_AssetFree).

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 K_{DK}	The length of optional K_{DK} (in bits)	Security strength *s* supported (in bits)
(HMAC-)SHA-1	up-to 512	HMAC-SHA-1	160	$112 \leq s \leq 160$
(HMAC-)SHA-224	up-to 512	HMAC-SHA-224	224	$112 \leq s \leq 224$
(HMAC-)SHA-256	up-to 512	HMAC-SHA-256	256	$112 \leq s \leq 256$
(HMAC-)SHA-384	up-to 1024	HMAC-SHA-384	384	$112 \leq s \leq 384$
(HMAC-)SHA-512	up-to 1024	HMAC-SHA-512	512	$112 \leq s \leq 512$
AES-128-CMAC	128	AES-128-CMAC	128	$112 \leq s \leq 128$
AES-192-CMAC	192	AES-128-CMAC	128	$112 \leq s \leq 128$
AES-256-CMAC	256	AES-128-CMAC	128	$112 \leq s \leq 128$

Two-Step Key Derivation will make use of both salt and K_{DK}.

6.2.3 HMAC-based Extract-and-Expand Key Derivation Function (HKDF) for TLS v1.3

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

This is a non-FIPS approved function. The function can be used in FIPS mode of operation for compatibility with TLS v1.3 protocol and NIST SP 800-52 Rev2 only.

6.2.4 NIST SP 800-132: Password-Based Key Derivation Function

The key derived using NIST SP 800-132 shall only be used for storage purposes.

Both options presented in NIST SP 800-132 for deriving the Data Protection Key from the Master Key are supported.

The SafeZone FIPS Lib does not limit the length of the passphrase 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. Thus, for security over 64 bits, the passwords must generally be longer than 12 characters.

Minimum requirements and limits for NIST SP 800-132:

- There is no maximum for length of salt used, but at least 128 bits (16 bytes) of salt value must be randomly generated.
- Iteration count shall be as large as possible. Iteration count used must be at least 1000 to meet 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 (Master key) can be used directly as the Data protection key, or as input to KDF or as a decryption key for Encrypted Data protection key.

6.2.5 NIST SP 800-38D: Galois/Counter Mode

The FIPS 140-2 Implementation Guidance A.5 applies to AES-GCM usage with this module.

Item 1 in IG A.5 forbids using external IV for encryption via the `FL_CryptAuthInit` function. However, the `FL_CryptAuthInit` function is still used for decryption and the `FL_CryptAuthInit` function is used for subsequent encryption operations for operation sequences started with the `FL_EncryptAuthInitDeterministic` function.

`FL_CryptAuthInit` function can also be used by the user to initialize AES-GCM operation, when IV has been generated in manner compatible with AES-GCM Implementation Guidance A.5, such as when internal IV generation has been accomplished by other FIPS 140-2 validated module.

The operator must use the `FL_EncryptAuthInitRandom` function if random IV generation (IG A.5 item 2) is required, or in case of deterministic IV generation (IG A.5 item 3), the `FL_EncryptAuthInitDeterministic` function.

The module supports AES-GCM with IPSec, TLS v1.2, TLS v1.3 and SRTP protocols. The IPSec and TLS v1.2 protocols will use the `FL_EncryptAuthInitDeterministic` function. For TLS v1.3, functions `FL_CryptAuthInitTls13`, `FL_EncryptAuthFinishTls13`, `FL_EncryptAuthTls13` and have been introduced. These functions provide equivalent functionality than `FL_EncryptAuthInitDeterministic`, but work with TLS v1.3 protocol values. TLS v1.3 uses 64-bit counter, but instead of counting from 0, counter is masked (XOR) with 96-bit value (known as Encryption Salt) derived at the same time than key. This value also offers equivalent of ModuleName used by IG A.5 item 3. Therefore, it can be concluded that the scheme offers security equivalent to IG A.5 item 3. Also decryption functions, `FL_DecryptAuthFinishTls13`, `FL_DecryptAuthTls13` functions are offered for consistency. 64-bit counter incremented once per TCP record will never overflow in practice. However, TLS 1.3 RFC 8446 suggests much smaller limits for key usage: encryption of up-to $2^{24.5}$ (around 24 million) full-size records is the recommended maximum use for a key. It is responsibility of user to decide limit for key usage (based on e.g. record size they use) and rekey or terminate connection after the chosen limit has been met.

The module supports AES-GCM with SRTP (RFC 7714). For SRTP, functions `FL_EncryptAuthSrtp`, `FL_EncryptAuthSrtcp` have been introduced. These functions provide equivalent functionality than `FL_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 A.5 item 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 responsibility of the user to rekey upon counter overflow. In addition, SRTP uses 96-bit Encryption Salt that is XORred with other fields. For control purposes, SRTP has 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 FIPS 140-2 Implementation Guidance A.5: Item B3 applies: In case a module's power is lost and then restored, the key used for the AES GCM encryption/decryption must be re-distributed.

6.2.6 NIST SP 800-38E: XTS Mode

The module supports XTS Mode for Confidentiality on Storage Devices. Both XTS-AES-128 (256 bit key) and XTS-AES-256 (512 bit key) are supported. The

XTS-AES key is parsed as concatenation of two AES keys *Key_1* and *Key_2*. As is explained in FIPS 140-2 Implementation Guidance A.9, it is required that *Key_1* ≠ *Key_2*. If *Key_1* = *Key_2*, attempt to perform XTS-AES encryption or decryption will fail. The XTS Mode is only approved for usage in storage applications.

6.2.7 NIST SP 800-67 Rev 2: Triple-DES Encryption

The module support Triple-DES encryption algorithm. The algorithm has tight restrictions for maximum number of encryptions with a single key. It is allowed to perform at most 2^{20} (IETF protocols) or 2^{16} (non-IETF protocols) data block encryptions with the same Triple-DES key.

The module does not enforce these limits, instead the user of the module is responsible for ensuring their use of the module meets these restrictions.

According to NIST SP 800-131A Rev 2, Triple-DES Encryption is deprecated algorithm and is becoming disallowed after 2023. It is recommended that the users of move to AES algorithm for symmetric encryption needs.

6.2.8 NIST SP 800-90A Rev1: Deterministic Random Bit Generator

The module generates cryptographic keys whose strengths are modified by available entropy. No assurance of the minimum strength of the generated keys is given by the module. Depending on the platform, the module provides access to different entropy sources.

By default, the SafeZone FIPS Cryptographic Module DRBG uses x86/x86-64 architecture specific RDSEED or RDRAND instructions to obtain entropy. If the instructions are not available, the module will use *getrandom* Linux system call. If the system call does not exist, it will use operating system and platform independent operating system device files as entropy sources. Primarily, the module uses `/dev/random` as the entropy source on platforms that provide such an entropy device. This entropy generation path is merely a convenience default. The quality of entropy coming from `/dev/random` is not measured by the SafeZone FIPS Cryptographic Module.

It is possible to use function `FL_RbgUseNonblockingEntropySource` to configure `/dev/urandom` as the entropy source. The difference between `/dev/random` and `/dev/urandom` is that when the entropy source does not know if there is sufficient entropy available, `/dev/random` will block and `/dev/urandom` will generate pseudo-random values based on available entropy. The quality of entropy coming from `/dev/urandom` is not measured by the SafeZone FIPS Cryptographic Module.

If Crypto Officer uses `/dev/random` or `/dev/urandom` as entropy source, it is up to Crypto Officer to configure it suitably to provide reasonable security. Crypto Officer can provide an entropy function which overrides the default entropy source.

6.2.9 NIST SP 800-133: Key Generation

The module allows key generation. Key generation will use one of random number generators, NIST SP 800-90A Rev1 DRBG-CTR AES-256 or DRBG-CTR AES-128. AES-256 based DRBG is used to generate symmetric keys and many of asymmetric keys. AES-128 DRBG is used in generation of some asymmetric keys of up-to 128 bit equivalent security strength (such as RSA-2048 and RSA-3072 keys).

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.

6.2.10 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 `FL_MacGenerateFinish` or `FL_MacVerifyFinish` is 4.

6.2.11 Support of Industry Protocols

The module is intended to interoperate with established Industry Protocols.

The module implements wide support of primitives for building applications with IKE, SRTP and/or TLS support. The key derivation functions from NIST SP 800-135 revision 1, various key agreement schemes, encryption and authentication functions are implemented. In addition, key extract-and-expand scheme from NIST SP 800-56C Rev1 is implemented for use with TLS v1.3.

Key Agreement groups supported for IKEv2 (IPsec) include 2048-bit MODP group (ID=14), 3072-bit MODP group (ID=15), 4096-bit MODP group (ID=16), 6144-bit MODP group (ID=17), 8192-bit MODP group (ID=18), 2048-bit MODP group with 224-bit prime order subgroup (ID=23), 2048-bit MODP group with 256-bit prime order subgroup (ID=24), NIST P-224 (ID=26), NIST P-256 (ID=19), NIST P-384 (ID=20), and NIST P-521 (ID=21).

Key Agreement groups supported for TLS v1.0-v1.2 include ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, P-224, P-256, P-384, and P-521. In addition, the module may process generic Diffie-Hellman parameters (with p between 2048 and 8192 bits), and the module can support RSA Key Transport with PKCS #1 v1.5.

This module supports implementation of IKEv1, IKEv2 and TLS v1.0/v1.1/v1.2 protocols. No parts of the protocols, other than KDF, have been tested by the CAVP and CMVP. The module also supports implementation of TLS v1.3 protocol, but as there is no CMVP guidance for TLS v1.3, the protocol has not been tested by CAVP or CMVP.

6.2.12 Processor Algorithmic Acceleration

This module supports Processor Algorithm Acceleration (PAA) functions.

These functions allow fast and power efficient execution of common cryptographic functions on processors.

The supported PAA implementations are:

- AES-NI (Intel and AMD processors)
    - Accelerator sub-functions for AES implementations (including GCM)
- ARMv8-a Cryptography Extensions
    - Accelerator sub-functions for AES and SHA implementations

These capabilities are automatically enabled on processors with appropriate support. Processors without PAA support can be used, but AES or SHA performance will be lower than for processors with PAA support. For compliance with requirements of FIPS 140-2 Implementation Guidance 1.21, the same module is able to run with and without Processor Algorithmic Acceleration functions.

6.3 Porting maintaining validation

SafeZone FIPS 140-2 product is typically delivered in binary format. The product in binary format is validated for platforms mentioned in the validation certificate.

There is also another product available from the Rambus: source code product. This package can be recompiled by the user for their target platform. Recompilation without any changes to the module is allowed by FIPS 140-2 Implementation Guidance G.5. If changes to the module are required, they need to be processed as a revalidation (see FIPS 140-2 Implementation Guidance G.8 for details). This section provides guidance for user post-validation porting of the module.

A user may perform post-validation porting of a module and affirm the modules continued validation compliance provided that the following is maintained: A software, firmware or hybrid cryptographic module will remain compliant with the FIPS 140-2 validation when operating on any general purpose computer (GPC) or platform provided that the GPC for the software module, or software controlling portion of the hybrid module, uses the specified single user operating system/mode specified on the validation certificate, or another compatible single user operating system, or that the GPC or platform for the firmware module or firmware controlling portion of the hybrid module, uses the specified operating system on the validation certificate.

Rambus can provide unmodified source code of the cryptographic module for purpose of post-validation porting. The package is only available within a commercial licensing agreement. The module can be compiled for target system(s) with commands:

```
tar zxf FIPSLib1.2-src.tgz
cd FIPSLib1.2-src
make PLATFORM=target
```

The *target* can currently be one of x86_64-linux-gnu, i686-linux-gnu, arm-linux-gnueabi, arm-linux-gnueabihf or aarch64-linux-gnu.

If the unmodified module does not compile with these commands, the target platform cannot be supported without revalidation.

7 Self Tests

7.1 Power-Up Self-Tests

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

- Software Integrity Test (using ECDSA Verify with NIST P-224)
- KAT test for SHA-1
- KAT test for SHA-512
- KAT test for SHA3-224
- KAT test for HMAC SHA-256
- KAT test for AES encryption (CBC, 128-bit key)
- KAT test for AES decryption (CBC, 128-bit key)
- KAT test for AES encryption (CCM, 128-bit key)
- KAT test for AES decryption (CCM, 128-bit key)
- KAT test for AES encryption (GCM, 128-bit key)
- KAT test for AES decryption (GCM, 128-bit key)
- KAT test for AES encryption (XTS, 128-bit key strength)
- KAT test for AES decryption (XTS, 128-bit key strength)
- KAT test for CMAC, 192-bit key
- KAT test for Triple-DES encryption (CBC, 192-bit key)
- KAT test for Triple-DES decryption (CBC, 192-bit key)
- KAT for RSA 2048-bit (signing and verification; PKCS #1 v1.5)
- KAT for DSA (signing P=2048/N=256; verification P=1024/N=160)
- KAT for ECDSA Signing (NIST P-224)
- KAT for KTS: RSA Key Wrapping 2048-bit (RSA-OAEP; encryption and decryption)
- KAT for Diffie-Hellman
- KAT for EC Diffie-Hellman
- KAT for KBKDF (Counter Mode)
- KAT for 800-56C (includes KBKDF Feedback Mode)
- KAT for KBKDF (Double Pipeline Mode)
- KAT and health test for AES-CTR-256 DRBG
- KAT and health test for AES-CTR-128 DRBG

The self-tests are invoked automatically upon loading the SafeZone FIPS Cryptographic Module. The initialization function `FL_LibInit`/`FLS_LibInit` is executed via DEP (default entry point) as specified in FIPS 140-2 Implementation Guidance 9.10.

Any error during the power-up self tests will result in a module transition to the error state. There are two possible ways to recover from the error state:

- Reinitializing the module with the API function sequences `FL_LibUnInit`/`FLS_LibUnInit` and `FL_LibInit`/`FLS_LibInit`.

- Power-cycling the device and reinitialize the module with the API function `FL_LibInit`/`FLS_LibInit`. (Invoked automatically via DEP.)

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.

7.2 Conditional Self tests

The SafeZone FIPS Cryptographic Module contains the following conditional self-tests:

- Pair-wise consistency check for key pairs created for digital signature purposes (DSA, FIPS 186-4)
- Pair-wise consistency check for key pairs created for digital signature purposes (ECDSA, FIPS 186-4)
- Pair-wise consistency check for RSA key pairs created for digital signature (FIPS 186-4) or key transport (NIST SP 800-56B) purposes.
- Conditional health test for AES-CTR-256 DRBG (upon reseeding).
- Continuous random number generator test for DRBG.
- Continuous random number generator test for non-Approved NDRNG.

The conditional self-tests for manual key entry and software/firmware load or bypass are not provided, as these are not applicable.

Any error during the conditional self tests will result in a module transition to the error state. The ways to recover from the error state are listed in section 7.1.

8 Mitigation of Other Attacks

The module contains an implementation of the RSA algorithm with data independent processing time for signing and decryption operations. This makes it harder to attack the RSA implementation via timing attacks.

The module does not mitigate other attacks outside the scope of FIPS 140-2.