Summit Linux FIPS Core Crypto Module

Software Version 7.1

FIPS 140-2 Non-Proprietary Security Policy

Document Version 1.2

Last update: 2024-04-10

Prepared by:

atsec information security corporation

4516 Seton Center Pkwy, Suite 250

Austin, TX 78759

www.atsec.com

Table of Contents

List of Tables

List of Figures

1 Introduction

This document is the non-proprietary FIPS 140-2 Security Policy for the Summit Linux FIPS Core Crypto Module, software version 7.1. It contains the security rules under which the module must be operated and describes how this module meets the requirements as specified in FIPS 140-2 (Federal Information Processing Standards Publication 140-2) for a Security Level 1 module.

This Security Policy contains non-proprietary information. All other documentation submitted for FIPS 140-2 conformance testing and validation is proprietary and is releasable only under appropriate non-disclosure agreements.

1.1 Purpose of the Security Policy

There are three major reasons that a security policy is needed:

- It is required for FIPS 140-2 validation,
- It allows individuals and organizations to determine whether a cryptographic module, as implemented, satisfies the stated security policy, and
- It describes the capabilities, protection and access rights provided by the cryptographic module, allowing individuals and organizations to determine whether it will meet their security requirements.

1.2 Target Audience

This document is part of the package of documents that are submitted for FIPS 140 2 conformance validation of the module. It is intended for the following audience:

- Developers.
- FIPS 140-2 testing lab.
- The Cryptographic Module Validation Program (CMVP).
- Customers using or considering integration of the Summit Linux FIPS Core Crypto Module.

1.3 How this Security Policy was Prepared

The vendor has provided the non-proprietary Security Policy of the cryptographic module, which was further consolidated into this document by atsec information security together with other vendor-supplied documentation as guided by FIPS 140-2 IG G.9. In preparing the Security Policy document, the laboratory formatted the vendor-supplied documentation for consolidation without altering the technical statements therein contained. The further refining of the Security Policy document was conducted iteratively throughout the conformance testing, wherein the Security Policy was submitted to the vendor, who would then edit, modify, and add technical contents. The vendor would also supply additional documentation, which the laboratory formatted into the existing Security Policy, and resubmitted to the vendor for their final editing.

2 Cryptographic Module Specification

2.1 Module Overview

The Summit Linux FIPS Core Crypto Module (hereafter referred to as the “module”) is a Software module supporting FIPS 140-2 Approved cryptographic algorithms. The module is composed by software components comprised of a kernel and OpenSSL libraries. These libraries provide a C language application program interface (API) for use by other processes that require cryptographic functionality.

The module offers approved cryptographic functions in the FIPS mode for, among other uses:

- Algorithms for use in the Wi-Fi protocols CCMP and GCMP.
- Algorithms for use in the TLS protocol.
- Encryption and decryption for data at rest.

2.2 FIPS 140-2 Validation Scope

Table 1 shows the security level claimed for each of the eleven sections that comprise the FIPS 140-2 standard.

Table 1: FIPS 140-2 Security Requirements.

Security Requirements Section		Level
1	Cryptographic Module Specification	1
2	Cryptographic Module Ports and Interfaces	1
3	Roles and Services and Authentication	1
4	Finite State Machine Model	1
5	Physical Security	N/A
6	Operational Environment	1
7	Cryptographic Key Management	1
8	EMI/EMC	1
9	Self-Tests	1
10	Design Assurance	1
11	Mitigation of Other Attacks	N/A
Overall Level		**1**

2.3 Definition of the Cryptographic Module and its Cryptographic Boundaries

The Summit Linux FIPS Core Crypto Module is defined as a Software, Multi-chip Standalone module per the requirements within FIPS 140-2. The logical cryptographic boundary of the module consists of the software component files (kernel, SummitSSL, and the fipscheck integrity test tool) and the integrity test HMAC files.

Figure 1 depicts the physical representation of the tested platform: the top view of the circuit board, and a bottom view of the same circuit board. The tested environment is further described in Section 2.4.

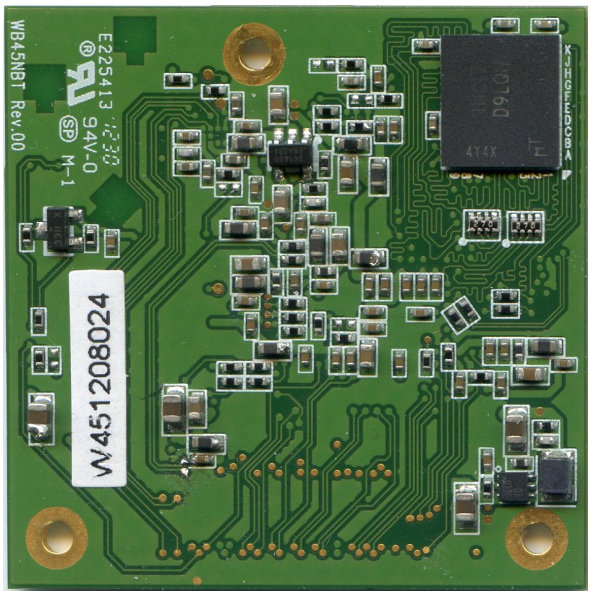

Figure 1: Physical configurations of the tested platform (top and bottom in order).

Table 2 lists the components that integrate the module. The software components of the module are of version 7.1.

Table 2: Components of the cryptographic module

Component	Description
SummitSSL library	Software shared library (based on OpenSSL version 1.0.2u), containing cryptographic algorithms. Filename: /usr/lib/libcrypto.so.1.0.0 HMAC: 964e4c5ce0ca2149804c6142bf4af937533f76eeebd492acda51174aee720c04
Summit Linux kernel	Kernel (based on Linux kernel version 4.19), containing cryptographic algorithms. The kernel functions as the interface to the hardware. Filename: /boot/Image.lzma HMAC: 28ade14713c70a00af3b5c40cb9b97bf698344ef56a1a76f9322f77e1d8f4c5d
HMAC integrity files	Files containing HMAC values for integrity test of software components. Files: /usr/lib/fipscheck/Image.lzma.hmac

Component	Description
	/usr/lib/fipscheck/libcrypto.so.1.0.0.hmac /usr/lib/fipscheck/fipscheck.hmac /usr/lib/fipscheck/libfipscheck.so.1.hmac
fipscheck integrity test tool	Software tool that performs integrity test of the software components of the module. Files: /usr/bin/fipscheck (executable) /usr/lib/fipscheck/libfipscheck.so.1 (library) Executable HMAC: cfeff5a2ab3cbed28464f9d0f37ce7ae11cde0cd94b9376807564fd1537b4ef3 Library HMAC: 22ba7737252fa6a25d5c93b80684ee59e34806cd84a59cfea89f6bfa4aeed6a8

Figure 2 shows the block diagram of the module. The logical cryptographic boundary is indicated with orange blocks, distributed among the software components. Blocks of another color do not belong to the logical boundary. Users of the module interact through the software API that are the logical interfaces mapping to the FIPS interfaces (data input and output, control input, status output). A dotted line encompasses the module's components that interface through the API.

In Figure 2, users of the module are exemplified by applications. These applications may reside within the NAND Flash memory or may reside outside (but still within the physical boundary), always interacting with the module's API.

The physical cryptographic boundary of the module is defined as the perimeter of the circuit board on which the module is installed (Section 2.4). The filesystem and operating system reside on NAND Flash memory within the physical boundary.

No components are excluded from the requirements of FIPS 140-2.

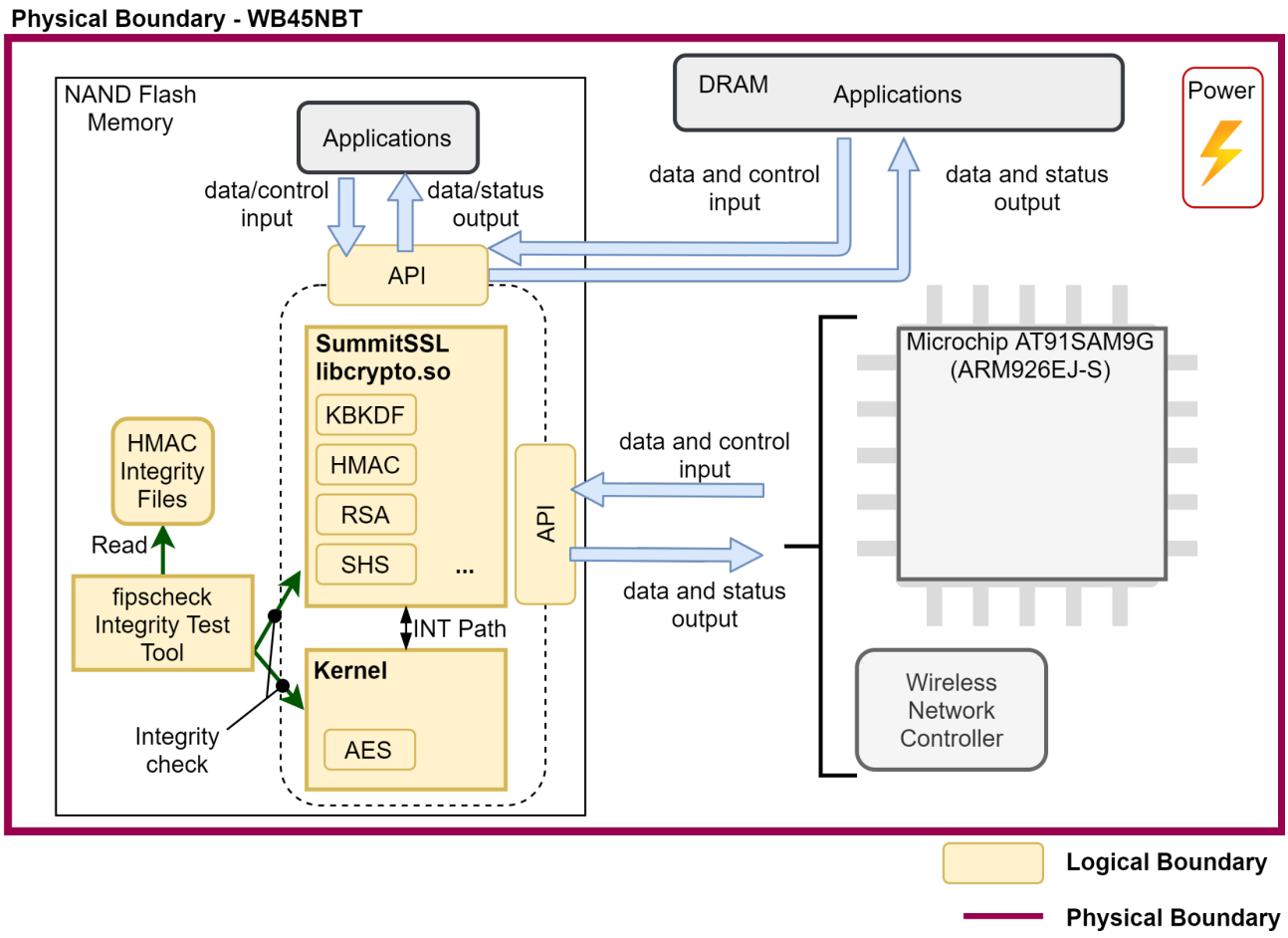

Figure 2: Block diagram with logical and physical cryptographic boundaries.

2.4 Tested Operational Environment

The module was tested on the operational environment listed in Table 3. These environment are composed of the WB45NBT wireless bridge device with Microchip AT91SAM9G (ARM926EJ-S) microprocessor.

Table 3: Tested operational environment.

Operating System	Microprocessor	Hardware
Summit Linux 7.1	Microchip AT91SAM9G (ARM926EJ-S), ARMv5-based	Ezurio WB45NBT wireless bridge

2.5 Modes of Operation

The module supports two modes of operation.

- In "**FIPS mode**" (the Approved mode of operation), only approved or allowed security functions, with sufficient security strength, are offered by the module.
- In "**non-FIPS mode**" (the non-Approved mode of operation), non-approved security functions are offered by the module.

The module enters the operational mode after Power-On Self-Tests (POST) succeed. Once the module is operational, the mode of operation is implicitly assumed depending on the selected

setting for the function FIPS_mode_set() invoked prior to the service, the security function invoked, and the security strength[1] of the cryptographic keys/curves chosen for the function or service.

In more detail, the module assumes the mode of operation in the following manner.

- Kernel Component:
  - The mode is implicitly assumed depending on the security function invoked and the security strength of the cryptographic keys/curves chosen (see Table 5 and Table 6).
- SummitSSL Component:
  - If FIPS_mode_set(0) is called before the invocation of the service, the module implicitly assumes the non-FIPS mode of operation for any service.
  - If FIPS_mode_set(1) is called before the invocation of the service:
    - If the service is strictly listed in Table 5 and is using strictly algorithms, keys/curves listed in Table 7 and Table 8, then the module assumes the FIPS-approved mode of operation.
    - Otherwise, the module assumes the non-FIPS mode of operation.

If the POST or the Conditional Tests fail (Section 9), the module goes into the error state. The status of the module can be determined by the availability of the module. If the module is available, then it has passed all self-tests. If the module is unavailable, it is because any self-test failed, and the module has transitioned to the error state.

The module does not share keys and other Critical Security Parameters (CSPs) that are used or stored in FIPS mode with functions in the non- FIPS mode, and vice versa.

[1] See Section 5.6.1 in SP800-57 for a definition of “security strength”.

3 Module Ports and Interfaces

As a software-only module, the module does not have physical ports. For the purposes of FIPS 140-2 definition, the physical ports are those of the hardware platform on which the software module runs.

The module provides cryptographic services and an application program interface (API) for user applications. The logical interfaces are represented by the API through which applications request services and obtain responses. The API represents the logical interfaces with the software components of the module.

Table 4 summarizes the FIPS logical interfaces and their mappings to the software interfaces.

Table 4: Ports and interfaces.

Logical Interface	Description
Data Input	API input parameters for data
Data Output	API output parameters for data
Control Input	API function calls, API input parameters for control.
Status Output	API return codes, API output parameters for status, and log messages.
Power Input	The power input is not applicable for the software components.

The Power Input is not applicable for the software components.

4 Roles, Services and Authentication

4.1 Roles

The module supports the following roles:

- **User role**: performs all services (in both FIPS mode and non-FIPS mode of operation), except module installation and configuration.
- **Crypto Officer role**: performs module installation and configuration.

The User and Crypto Officer roles are implicitly assumed by the entity accessing the module's services. In other words, by invoking a specific service offered by the module, the role is implicitly assumed by the entity according to the service that was invoked by that entity, as the service is defined to one or the other role.

4.2 Services

The module provides services to entities who assume one of the available roles. Table 5 and Table 6 depict all services that are described in more detail in the developer documentation. The tables also list the roles allowed to invoke each service, and the keys and Critical Security Parameters (CSPs) involved and how these keys and CSPs are accessed.

The module does not implement the GCMP, CCMP, or TLS protocols, but rather the cryptographic algorithms (such as the TLS KDF and SP800-108 KDF) that can be used to implement those protocols by applications.

The tables use the following convention when specifying the access permissions that the service has for each CSP or key. The applicable abbreviations are shown within parentheses.

- **Create (C):** the user entity can create a new CSP.
- **Read (R):** the user entity can read the CSP.
- **Update (U):** the user entity can write a new value to the CSP.
- **Zeroize (Z):** the user entity can zeroize the CSP.
- **N/A:** the user entity does not access any CSP or key during its operation.

For the "Role" column, U indicates the User role, and CO indicates the Crypto Officer role. An X marks which role has access to that service.

4.2.1 Services in the FIPS-Approved Mode of Operation

Table 5 provides a full description of FIPS Approved services and the non-Approved but Allowed services provided by the module in the FIPS-approved mode of operation.

Table 5: Services in the FIPS-approved mode of operation.

Service	Service Description and Algorithms	Role		Keys and CSPs	Access Types
		U	CO		
SummitSSL Component: Services with FIPS_mode_set(1)					
Symmetric Encryption/Decryption	Encrypts or decrypts a block of data using AES.	X		AES Key	R
RSA Key Generation	Generate RSA asymmetric keys using DRBG.	X		RSA public/private keys	C, R

Service	Service Description and Algorithms	Role		Keys and CSPs	Access Types
		U	CO		
DSA Key Generation	Generate DSA asymmetric keys using DRBG.	X		DSA public/private keys	C, R
ECDSA Key Generation	Generate ECDSA asymmetric keys using DRBG.	X		ECDSA public/private keys	C, R
RSA Digital Signature Generation and Verification	Sign and verify signature operations for RSA PCKS#1v1.5, RSA-PSS and X9.31.	X		RSA public/private keys	R
DSA Digital Signature Generation and Verification	Sign and verify signature operations for DSA.	X		DSA public/private keys	R
ECDSA Digital Signature Generation and Verification	Sign and verify signature for ECDSA.	X		ECDSA public/private keys	R
TLS Key Derivation	Derive keying material for use in the TLS protocol. KDF in TLS v1.0/1.1, TLS v1.2 (SP800-135).	X		Pre-master secret, master secret, derived key (AES, HMAC), KDF internal state	C, R, U
Key-Based Key Derivation (KBKDF)	SP800-108 KDF in Counter mode. Derive keys for establishment of secure communication channels.	X		Key derivation key and derived keys (AES, HMAC), 802.11 pre-shared key (PSK), 802.11 pairwise master key (PMK), 802.11 KDF internal state, 802.11 Temporal Keys, 802.11 MIC keys (KCK), 802.11 Key Encryption Key (KEK), EAP-TLS MSK, EAP-TTLS MSK, EAP-PEAP MSK	C, R, U
Diffie-Hellman Shared Secret Computation	Establish a shared secret. KAS-FFC-SSC	X		Diffie-Hellman public/private keys, shared secret, pre-master secret	C, R
EC Diffie-Hellman Shared Secret Computation	Establish a shared secret. KAS-ECC-SSC	X		EC Diffie-Hellman public/private keys, shared secret, pre-master secret	C, R
Key Wrapping with RSA	Encapsulates and decapsulates a key using RSA encrypt/decrypt primitives	X		RSA public/private keys.	C, R, U

Service	Service Description and Algorithms	Role		Keys and CSPs	Access Types
		U	CO		
				Wrapped key[2].	
Key Wrapping with Symmetric Algorithms	Wrap and unwrap keys with AES-KW, AES-KWP	X		AES keys (key wrapping key). Wrapped key.	C, R, U
Certificate Management	Management of key properties within certificates.	X		RSA, DSA, and ECDSA public/private keys associated to an X.509 certificate	R, U
Message Authentication Code (MAC)	Authenticate and verify authentication of data using HMAC-SHA-1, HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512	X		HMAC Key	R
	Authenticate and verify authentication of data using CMAC and GMAC with AES-128, AES-192, AES-256.	X		AES Key	R
Message Digest	Hash a block of data with SHS. SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	X		None	N/A
Random Number Generation	Generate random numbers based on the SP800-90A DRBG.	X		Entropy input string, internal state, seed	C, R, U
Kernel Component					
Symmetric Encryption/Decryption	Encrypts or decrypts a block of data using AES.	X		AES Key	R
Other FIPS-related Services					
Show Status	Show status of the module state	X		None	N/A
Self-Test	Initiate power-on self-tests	X		None	N/A
Zeroize	Zeroize all critical security parameters	X		All keys and CSPs	Z
Module Installation	Installation of the module		X	None	N/A

[2] The “Wrapped key” is the data output from the wrapping algorithm, or the input if the algorithm is being used to unwrap a key. This wrapped key can be of any type, and this type is transparent to the service.

Service	Service Description and Algorithms	Role		Keys and CSPs	Access Types
		U	CO		
Module Configuration	Configuration of the module		X	None	N/A

4.2.2 Services in the Non-FIPS-Approved Mode of Operation

Table 6 presents the services only available in non-FIPS-approved mode of operation.

Table 6: Services in the non-FIPS approved mode of operation.

Service	Service Description and Algorithms	Role		Keys	Access Types
		U	CO		
SummitSSL Component: Services with FIPS_mode_set(0)					
Symmetric Encryption/Decryption	Encrypts or decrypts a block of data using AES.	X		AES Key	R
RSA Key Generation	Generate RSA asymmetric keys using DRBG.	X		RSA public/private keys	C, R
DSA Key Generation	Generate DSA asymmetric keys using DRBG.	X		DSA public/private keys	C, R
ECDSA Key Generation	Generate ECDSA asymmetric keys using DRBG.	X		ECDSA public/private keys	C, R
RSA Digital Signature Generation and Verification	Sign and verify signature operations for RSA PCKS#1v1.5, RSA-PSS and X9.31.	X		RSA public/private keys	R
DSA Digital Signature Generation and Verification	Sign and verify signature operations for DSA.	X		DSA public/private	R
ECDSA Digital Signature Generation and Verification	Sign and verify signature for ECDSA.	X		ECDSA public/private keys	R
TLS Key Derivation	Derive keying material for use in the TLS protocol. KDF in TLS v1.0/1.1, TLS v1.2 (SP800-135).	X		Pre-master secret, master secret, derived key (AES, HMAC), KDF internal state	C, R, U
Key-Based Key Derivation (KBKDF)	SP800-108 KDF in Counter mode. Derive keys for establishment of	X		Key derivation key and derived keys (AES, HMAC), 802.11 pre-shared key (PSK), 802.11 pairwise	C, R, U

Service	Service Description and Algorithms	Role		Keys	Access Types
		U	CO		
	secure communication channels.			master key (PMK), 802.11 KDF internal state, 802.11 Temporal Keys, 802.11 MIC keys (KCK), 802.11 Key Encryption Key (KEK), EAP-TLS MSK, EAP-TTLS MSK, EAP-PEAP MSK	
Diffie-Hellman Shared Secret Computation	Establish a shared secret. KAS-FFC-SSC	X		Diffie-Hellman public/private keys, shared secret, pre-master secret	C, R
EC Diffie-Hellman Shared Secret Computation	Establish a shared secret. KAS-ECC-SSC	X		EC Diffie-Hellman public/private keys, shared secret, pre-master secret	C, R
Key Wrapping with RSA	Encapsulates and decapsulates a key using RSA encrypt/decrypt primitives	X		RSA public/private keys. Wrapped key	C, R, U
Key Wrapping with Symmetric Algorithms	Wrap and unwrap keys with AES-KW, AES-KWP	X		AES keys (key wrapping key). Wrapped key	C, R, U
Message Authentication Code (MAC)	Authenticate and verify authentication of data using HMAC-SHA-1, HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512	X		HMAC Key	R
	Authenticate and verify authentication of data using CMAC and GMAC with AES-128, AES-192, AES-256	X		AES Key	R
Random Number Generation	Generate random numbers based on the DRBG.	X		Entropy input string, internal state, seed	C, R, U
SummitSSL Component: Services with FIPS_mode_set(0) or FIPS_mode_set(1)					
Symmetric Encryption/Decryption	Encrypts or decrypts using non-Approved algorithms or key sizes not listed in Table 7	X		Triple-DES, Camellia, CAST, DES, IDEA, RC2, RC4, RC5 keys	R
RSA, DSA, ECDSA Key Generation	Generation of non-Approved RSA, DSA and ECDSA keys	X		RSA key < 2048 bits DSA keys or ECDSA curves not listed in Table 7	C, R

Service	Service Description and Algorithms	Role		Keys	Access Types
		U	CO		
Digital Signature Generation and Verification	Sign or verify operations with non-Approved RSA, DSA, or ECDSA key lengths or curves	X		RSA key < 2048 DSA keys or ECDSA curves not listed in Table 7 Signature Generation with SHA-1	R
TLS Key Derivation	Derive keying material for cryptographic function outside of a TLS protocol context, or using SHS and HMAC algorithms not listed in Table 7 or not in conformance with SP800-135.	X		Pre-master secret, master secret, derived key (AES, HMAC).	C, R, U
Key Wrapping with RSA	Encrypts or decrypts using non-Approved RSA key size	X		RSA key pair Wrapped key	C, R, U
Diffie-Hellman Shared Secret Computation	Establish a shared secret. KAS-FFC-SSC	X		Diffie-Hellman key < 2048 bits, shared secret	C, R
EC Diffie-Hellman Shared Secret Computation	Establish a shared secret. KAS-ECC-SSC	X		EC Diffie-Hellman CSPSs with curves not listed in Table 7 or Table 8, shared secret	C, R
Random Number Generation	Generation of random numbers using the ANSI X9.31 PRNG	X		seed, seed key, internal state	C, R, U
Message Digest	Hashing using non-Approved hash functions that include MD2, MD4, MD5, MDC2, RIPEMD, Whirlpool	X		None	N/A
J-PAKE Key Agreement	Password authenticated key agreement using J-PAKE	X		J-PAKE key pair	C, R
Kernel Component					
Symmetric Encryption/Decryption	Encrypts or decrypts using non-Approved algorithms	X		DES, RC4, Triple-DES keys	R
Digital Signature Generation and Verification	Sign or verify operations using RSA	X		RSA key pair	R

Service	Service Description and Algorithms	Role		Keys	Access Types
		U	CO		
Key Wrapping with RSA	Encrypts or decrypts using RSA	X		RSA key pair Wrapped key	C, R, U
Diffie-Hellman Shared Secret Computation	Establish a shared secret. KAS-FFC-SSC	X		Diffie-Hellman public/private keys, shared secret	C, R
EC Diffie-Hellman Shared Secret Computation	Establish a shared secret. KAS-ECC-SSC	X		EC Diffie-Hellman public/private keys, shared secret	C, R
Message Authentication Code (MAC)	Authenticate and verify authentication of data using HMAC-SHA-1, HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512, HMAC-SHA3-224, HMAC-SHA3-256, HMAC-SHA3-384, HMAC-SHA3-512	X		HMAC Key	R
	Authenticate and verify authentication of data using CMAC with Triple-DES	X		Triple-DES Key	R
Message Digest	Hashing using SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512, MD5 hash functions	X		None	N/A
Random Number Generation	Generate random numbers based on the DRBG.	X		Entropy input string, internal state, seed	C, R, U

4.3 Algorithms

The module implements cryptographic algorithms that are used by the services provided by the module. The cryptographic algorithms that are approved to be used in the FIPS mode of operation are tested and validated by the CAVP.

Table 7, Table 8 and Table 9 present the cryptographic algorithms in specific modes of operation. These tables include the CAVP certificates for different implementations, the algorithm name, respective standards, the available modes, key sizes, or curves wherein applicable, and usage. Information from certain columns may be applicable to more than one row.

The kernel component of the module provides multiple implementations of algorithms. Different implementations can be invoked by using the unique algorithm driver names. Among the implementations, the kernel component of the module supports generic C for all algorithms; generic C non-optimized AES for AES-CBC-CS and AES-GCM, assembler AES implementation with generic C block modes.

Section 13 brings a list of the names contained in the CAVP algorithm certificates that refer to the specific algorithm implementations, along with their description.

4.3.1 Approved Algorithms

Table 7 lists the cryptographic algorithms that are approved to be used in the FIPS mode of operation in this module. For the kernel component, the entries include the algorithm driver name that is approved for the respective algorithm.

Note that the algorithm certificates may include algorithms that are not available as approved in this module.

Table 7: FIPS-approved cryptographic algorithms.

Algorithm	Standard	Mode/Method	Key Lengths, Curves, Moduli (bits)	Use	CAVP Cert#
SummitSSL Component					
AES	FIPS197 SP800-38A	CBC, CFB1, CFB8, CFB128, CTR, ECB, OFB	128, 192 and 256 bits	Data Encryption and Decryption	A1488 (AES_C)
	FIPS197 SP800-38B	CMAC	128, 192 and 256 bits	MAC Generation and Verification	
	FIPS197 SP800-38C	CCM	128, 192 and 256 bits	Data Encryption and Decryption	
	FIPS197 SP800-38E	XTS	128, 256 bits	Data Encryption and Decryption	
	FIPS197 SP800-38F	KW, KWP	128, 192 and 256 bits	Key Wrapping and Unwrapping	
	FIPS197 SP800-38D	GCM with internal IV (Mode 8.2.1)	128, 192 and 256 bits	Data Encryption and Decryption	A1489 (AES_C_GCM)
	FIPS197 SP800-38D	GCM with external IV	128, 192 and 256 bits	Data Decryption[3]	
	FIPS197 SP800-38D	GMAC with external IV	128, 192 and 256 bits	MAC Generation and Verification	
DSA	FIPS 186-4	n/a	L=2048, N=224; L=2048, N=256; L=3072, N=256	Key Generation	A1490 (SHA_ASM)

[3] This algorithm was tested for both encryption and decryption. However, only the decryption operation is approved for this module.

Algorithm	Standard	Mode/Method	Key Lengths, Curves, Moduli (bits)	Use	CAVP Cert#
		P/Q Probable, G Unverifiable SHA2-224, SHA2-256, SHA2-384, SHA2-512	L=2048, N=224	Domain Parameter Generation	A1490 (SHA_ASM)
		P/Q Probable, G Unverifiable SHA2-256, SHA2-384, SHA2-512	L=2048, N=256; L=3072, N=256	Domain Parameter Generation	A1490 (SHA_ASM)
		P/Q Probable, G Unverifiable SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	L=1024, N=160	Domain Parameter Verification	A1490 (SHA_ASM)
		P/Q Probable, G Unverifiable SHA2-224, SHA2-256, SHA2-384, SHA2-512	L=2048, N=224	Domain Parameter Verification	A1490 (SHA_ASM)
		P/Q Probable, G Unverifiable SHA2-256, SHA2-384, SHA2-512	L=2048, N=256; L=3072, N=256	Domain Parameter Verification	A1490 (SHA_ASM)
		SHA2-224, SHA2-256, SHA2-384, SHA2-512	L=2048, N=224; L=2048, N=256; L=3072, N=256	Signature Generation	A1490 (SHA_ASM)
		SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	L=1024, N=160; L=2048, N=224; L=2048, N=256; L=3072, N=256	Signature Verification	A1490 (SHA_ASM)
DRBG	SP800-90A	CTR_DRBG AES128, AES192, AES256 with/without DF, with/without PR	n/a	Random Number Generation	A1488 (AES_C)
ECDSA	FIPS186-4	Testing Candidates	B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521	Key Generation	A1490 (SHA_ASM)

Algorithm	Standard	Mode/Method	Key Lengths, Curves, Moduli (bits)	Use	CAVP Cert#
		n/a	B-233, B-283, B-409, B-571, K-163, K-233, K-283, K-409, K-571, P-192, P-224, P-256, P-384, P-521	Key Verification	
		SHA2-224, SHA2-256, SHA2-384, SHA2-512	B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521	Signature Generation	
		SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	B-233, B-283, B-409, B-571, K-163, K-233, K-283, K-409, K-571, P-192, P-224, P-256, P-384, P-521	Signature Verification	
KAS-FFC-SSC SP800-56Ar3	SP800-56Ar3	dhEphem scheme	ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	Diffie-Hellman Shared Secret Computation	A1487 (FFC_DH)
KAS-ECC-SSC SP800-56Ar3	SP800-56Ar3	EphemeralUnified scheme	B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521	EC Diffie-Hellman Shared Secret Computation	A1490 (SHA_ASM)
Safe Primes Key Generation	SP800-56Ar3	n/a	ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	Key Generation	A1487 (FFC_DH)
Safe Primes Key Verification	SP800-56Ar3	n/a	ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	Key Verification	A1487 (FFC_DH)

Algorithm	Standard	Mode/Method	Key Lengths, Curves, Moduli (bits)	Use	CAVP Cert#
HMAC	FIPS198-1	SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	112 bits or greater	Message Authentication Code	A1490 (SHA_ASM)
KDF TLS v1.0/1.1, v1.2	SP800-135	TLS v1.0/1.1: SHA-1 TLS v1.2: SHA2-256, SHA2-384, SHA2-512	n/a	Key Derivation	A1490 (SHA_ASM)
KBKDF SP800-108	SP800-108	Counter Mode HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512	n/a	Key Derivation	A1486 (KBKDF)
RSA	FIPS186-4	B.3.3 Random Probably Primes Random Public Exponent	2048, 3072, 4096 bits	Key Generation	A1490 (SHA_ASM)
		X9.31 with SHA2-256, SHA2-384, SHA2-512	2048, 3072, 4096 bits	Digital Signature Generation	A1490 (SHA_ASM)
		PKCS#1v1.5 with SHA2-224, SHA2-256, SHA2-384, SHA2-512	2048, 3072, 4096 bits	Digital Signature Generation	A1490 (SHA_ASM)
		PSS with SHA2-224, SHA2-256, SHA2-384, SHA2-512	2048, 3072, 4096 bits	Digital Signature Generation	A1490 (SHA_ASM)
		X9.31 with SHA-1, SHA2-256, SHA2-384, SHA2-512	1024, 2048, 3072, 4096 bits	Digital Signature Verification	A1490 (SHA_ASM)
		PKCS#1v1.5 and PSS with SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	1024, 2048, 3072, 4096 bits	Digital Signature Verification	A1490 (SHA_ASM)
SHS	FIPS180-4	SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	N/A	Message Digest	A1490 (SHA_ASM)

Algorithm	Standard	Mode/Method	Key Lengths, Curves, Moduli (bits)	Use	CAVP Cert#
KTS	SP800-38F	AES-GCM	128, 256 bits	Key Wrapping within TLS	A1489 (AES_C_GCM)
	SP800-38F	AES-KW, AES-KWP	128, 192, 256 bits	Key Wrapping	A1488 (AES_C)
CKG	IG D.12 SP800-133	Asymmetric keys (Section 7.2)	N/A	N/A	Vendor Affirmed
Kernel Component					
AES	FIPS197 SP800-38A	CBC, CTR, ECB *Drivers:* *cbc(aes-generic)* *ctr(aes-generic)* *ecb(aes-generic)* *cbc(aes-arm)* *ctr(aes-arm)* *ecb(aes-arm)*	128, 192 and 256 bits	Data Encryption and Decryption	A1480 (C_C) A1483 (ARMASM_C)
	FIPS197 SP800-38B	CMAC *Drivers:* *cmac(aes-generic)* *cmac(aes-arm)*	128, 192 and 256 bits	MAC Generation and Verification	
	FIPS197 SP800-38C	CCM *Drivers:* *ccm_base(ctr(aes-generic),cbcmac(aes-generic))* *ccm_base(ctr(aes-arm),cbcmac(aes-arm))*	128, 192 and 256 bits	Data Encryption and Decryption	
	FIPS197 SP800-38E	XTS *Drivers:* *xts(aes-generic)* *xts(aes-arm)*	128, 256 bits	Data Encryption and Decryption	
	SP800-38A Addendum	CBC-CS3 *Drivers:* *cts(cbc(aes-generic))* *cts(cbc(aes-arm))*	128, 192 and 256 bits	Data Encryption and Decryption	A1481 (CTS_C_C) A1485 (CTS_ARMASM_C)

Algorithm	Standard	Mode/Method	Key Lengths, Curves, Moduli (bits)	Use	CAVP Cert#
	SP800-38F	GCM with external IV *Drivers:* *gcmp(gcm_base(ctr(aes-generic),ghash-generic))* *gcmp(gcm_base(ctr(aes-arm),ghash-arm))* *Decryption only:* *gcm_base(ctr(aes-generic),ghash-generic))* *gcmp(gcm_base(ctr(aes-arm),ghash-arm))*	128, 192, and 256 bits	Data Encryption and Decryption	A1480 (C_C) A1483 (ARMASM_C)
KTS	SP800-38F	AES-GCM *Drivers:* *gcmp(gcm_base(ctr(aes-generic),ghash-generic))* *gcmp(gcm_base(ctr(aes-arm),ghash-arm))* *Decryption only:* *gcm_base(ctr(aes-generic),ghash-generic))* *gcmp(gcm_base(ctr(aes-arm),ghash-arm)).*	128, 256 bits	Key Wrapping within GCMP	A1480 (C_C) A1483 (ARMASM_C)
Entropy Source					
ENT (NP)	SP800-90B	N/A	N/A	N/A	N/A

4.3.2 Non-Approved-but-Allowed Algorithms

Table 8 lists the non-Approved-but-Allowed cryptographic algorithms provided by the module that are allowed to be used in the FIPS mode of operation in this module.

Table 8: Non-Approved-but-allowed cryptographic algorithms.

Algorithm	Usage
RSA PKCS 1.5 Key Wrapping with key size between 2048 bits and 15360 bits (or more)	Key wrapping, key establishment methodology provides between 112 and 256 bits of encryption strength. Allowed by IG D.9.

4.3.3 Non-Approved Algorithms

Table 9 lists the cryptographic algorithms that are not allowed to be used in the FIPS mode of operation in this module. Use of any of these algorithms (and corresponding services in Table 6) will implicitly switch the module to the non-Approved mode.

Table 9: Non-FIPS approved cryptographic algorithms.

Algorithm	Usage
SummitSSL Component	
ANSI X9.31 RNG	Random number generation
Camellia	Encryption/decryption
CAST	Encryption/decryption
DES	Encryption/decryption
Diffie-Hellman	Shared secret computation using keys of length not listed in Table 7
DSA	Parameter/key generation/signature generation and verification with keys not listed in Table 7
EC Diffie-Hellman	Shared secret computation using curves not listed in Table 7
ECDSA	Key generation/signature generation and verification with curves not listed in Table 7
IDEA	Encryption/decryption
J-PAKE	Password Authenticated Key Exchange
MD2	Message digest
MD4	Message digest
MD5	Message digest
MDC2	Message digest
RC2	Encryption/decryption
RC4	Encryption/decryption
RC5	Encryption/decryption
RIPEMD	Message digest

Algorithm	Usage
RSA	Key generation/signature generation/signature verification, and key wrapping with keys of length not listed in Table 7
SHA-1	In signature generation
Triple-DES	Encryption/decryption
Whirlpool	Message digest
Kernel Component	
Triple-DES	Encryption/decryption
DES	
RC4	
RSA	Encryption/decryption
	Signature Generation/Verification
Diffie-Hellman	Shared secret computation
EC Diffie-Hellman	Shared secret computation
HMAC-SHA-1, HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512, HMAC-SHA3-224, HMAC-SHA3-256, HMAC-SHA3-384, HMAC-SHA3-512	Keyed-hash message authentication code
Triple-DES-CMAC	
SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512,	Message digest
MD5	
DRBG	Random Number Generation
GCM	RFC4106 implementations and those not listed in Table 7

4.4 Operator Authentication

The module does not support operator authentication mechanisms. The role of the operator is implicitly assumed based on the service requested.

5 Physical Security

The module is a software module, and thus the physical security requirements do not apply.

6 Operational Environment

6.1 Applicability

The module operates in a modifiable operational environment per FIPS 140-2 Security Level 1 specifications. The module runs on the Summit Linux operating system executing on the hardware specified in Section 2.4.

6.2 Policy

The operating system is restricted to a single operator mode of operation (i.e., concurrent operators are explicitly excluded by the operating system, which provides context and memory space separation for distinct processes).

The application that makes calls to the module is the single user of the module, even when the application is serving multiple clients.

7 Cryptographic Key Management

This section describes the cryptographic keys and CSPs managed by the module, and how this management is performed during the keys and CSPs life cycle.

Table 10 summarizes the keys and other CSPs that are used by the cryptographic services implemented in the module. The table lists the use of each key/CSP and, as applicable, how they are generated or established, and their method of entry and output of the module. For all keys and CSPs, the storage is in RAM in plaintext. The zeroization method is described in Section 7.5.

Table 10: Lifecycle of keys and other Critical Security Parameters (CSPs).

Name	Use	Generation/Establishment	Entry and Output	Type
AES Key	Encryption, decryption. MAC generation and verification for CMAC. Key wrapping.	Provided by the user entity.	Entered via API input parameter. No output.	AES key, all modes per Table 7. Length: 128, 192 and 256 bits for all modes except XTS: XTS accepts lengths of 128 and 256 bits.
AES Derived Key	Encryption, decryption. MAC generation and verification for CMAC. Key wrapping.	Derived during 802.11 authentication using 802.11 SP800-108 KDF. Derived by SP800-135 TLS KDF.	No entry. Output via API output parameters in plaintext.	AES key (CBC, CCM, GCM) with length 128 and 256 bits (defined by TLS ciphersuite, CCMP, or GCMP).
HMAC Key	MAC generation and verification	Provided by the user entity.	Entered via API input parameter. No output.	HMAC keys of length > 112 bits.
HMAC Derived Key	MAC generation and verification	Derived during 802.11 authentication using 802.11 SP800-108 KDF. Derived by SP800-135 TLS KDF.	No entry. Output via API output parameters in plaintext.	HMAC key of length defined by ciphersuite, CCMP, or GCMP.
RSA public and private key	RSA signature generation and verification. Key wrapping.	Keys are generated using FIPS 186-4 and the random value used in the key generation is obtained from SP800- 90A DRBG.	Entered via API input parameter or generated by module. Output via API output parameters in plaintext.	RSA keys of length 1024, 2048, 3072, 4096 bits (or more as allowed for key wrapping)
DSA public and private key	DSA signature generation and verification.	Keys are generated using FIPS 186-4 and	Entered via API input parameter	DSA keys of length 1024, 2048, 3072 bits

Name	Use	Generation/Establishment	Entry and Output	Type
		the random value used in the key generation is obtained from SP800- 90A DRBG.	or generated by module. Output via API output parameters in plaintext.	
ECDSA public and private key	ECDSA signature generation and verification.	Keys are generated using FIPS 186-4 and the random value used in the key generation is obtained from SP800- 90A DRBG.	Entered via API input parameter or generated by module. Output via API output parameters in plaintext.	ECDSA keys for all supported curves in Table 7.
Diffie-Hellman public and private key	Shared secret computation.	Keys are generated using SP800-56Arev3 methods and the random value used in the key generation is obtained from SP800- 90A DRBG.	No entry. Output via API output parameters in plaintext.	Key lengths from supported safe prime groups ffdhe/MODP in Table 7.
EC Diffie-Hellman public and private key	Shared secret computation.	Keys are generated using SP800-56Arev3 methods and the random value used in the key generation is obtained from SP800-90A DRBG.	No entry. Output via API output parameters in plaintext.	All supported curves in Table 7.
Pre-master secret	Establishment of encrypted session.	Generated during the shared secret computation when using Diffie-Hellman or EC Diffie-Hellman key exchange. Generated by TLS client as output from DRBG when using RSA key exchange.	Entry: if received by module as TLS server, wrapped with server’s public RSA key; otherwise, no entry. Output: if generated by module as TLS client, wrapped with server’s public RSA key; otherwise, no output.	Length defined per user application ciphersuite.

Name	Use	Generation/Establishment	Entry and Output	Type
Master secret	Establishment of encrypted session.	Derived from pre-master secret (SP800-135 TLS KDF).	N/A	384 bits.
Entropy input string	Entropy input strings used to compose the seed to the DRBG.	Obtained from entropy source.	N/A	384 bits.
DRBG Internal state (V, Key) and seed	Used to generate random bits.	During DRBG initialization and reseed.	N/A	Internal state and seed.
RSA, ECDSA, DSA private key associated to an X.509 certificate, and X.509 (public) certificates	Client and server authentication during TLS exchange.	Provided by the user entity.	Entered via API parameters. The certificate can exit the module via TLS protocol.	RSA, DSA, ECDSA keys and certificates.
802.11 Pre-shared key (PSK)	Used for pre-shared key authentication and session key establishment, as well as for 802.11 KDF	N/A	Manually distributed, electronically entered in plaintext. No exit.	Up to 256 bits of length.
802.11 Pairwise Master Key (PMK)	Used for pre-shared key authentication and session key establishment, as well as for 802.11 KDF	N/A	Manually distributed, electronically entered in plaintext, or derived from the PSK or EAP parameters (using any of the module's KDF functions). No exit.	256 or 384 bits.
802.11 KDF Internal State	Used for SP800-108 KDF to calculate the WPA2 session keys	SP800-108 KDF	N/A	Internal state of the KDF.
802.11 Temporal Keys	AES-CCM or AES-GCM keys used for session	SP800-108 KDF	N/A	AES-CCM, AES-GCM of 128 or 256 bits.

Name	Use	Generation/Establishment	Entry and Output	Type
	encryption/ decryption			
802.11 MIC keys (KCK)	Key confirmation keys (KCK) used for message authentication during session establishment	SP800-108 KDF	N/A	128 or 192 bits.
802.11 Key Encryption Key (KEK)	Used for AES Key Wrapping of the 802.11 Group Temporal Key (GTK)	SP800-108 KDF	N/A	128 or 256 bits.
802.11 Group Temporal Key (GTK)	802.11 session key for broadcast communications	Established by key transport: wrapped with 802.11 KEK (IG D.9).	Entered via key transport: wrapped with 802.11 KEK. No exit.	128, 256 bits.
TLS KDF Internal State	Values of the TLS KDF internal state used in EAP methods (Table 5).	SP800-135 TLS KDF	N/A	Internal state of the KDF.
EAP-TLS MSK	Establishment of encrypted session.	Derived from pre-master secret (SP800-135 TLS KDF).	N/A	At least 512 bits.
EAP-TTLS MSK	Establishment of encrypted session.	Derived from pre-master secret (SP800-135 TLS KDF).	N/A	At least 512 bits.
EAP-PEAP MSK	Establishment of encrypted session.	Derived from pre-master secret (SP800-135 TLS KDF).	N/A	At least 512 bits.
Shared Secret	Computation of shared secret for further key derivation	Generated during the shared secret computation when using Diffie-Hellman or EC Diffie-Hellman key exchange.	No entry. Output via API output parameters in plaintext.	Length defined per user application.

7.1 Random Number Generation

The module provides a DRBG compliant with SP800-90A for random number generation and the creation of key components of asymmetric keys. The DRBG implements a CTR_DRBG mechanism with AES-128, AES-192 or AES-256, with selectable enabling of derivation function and prediction

resistance. The DRBG is initialized during module initialization and seeded from the entropy source directly from /dev/hwrng. The entropy input part of the seed has a length of 413 bits.

The entropy source is provided by the kernel component, thus inside of the logical boundary. The entropy source is Non-Physical in nature, compliant with SP800-90B and marked as ENT (NP) on the certificate. This entropy source provides at least 256 bits of entropy in the entropy input part of the seed to the DRBG. The DRBG is then capable of supporting the full encryption strength of all its mechanisms during seeding and reseeding.

The module performs the health tests for the SP800-90A DRBG as defined per Section 11.3 of SP800-90A. The entropy source performs the health tests on the noise source per the requirements of SP800-90B, ensuring that the entropy source continues to operate as expected.

If any health test fails, the entropy source will not provide any data output. The entropy collector instance will remain in error state. To recover from the failure, the entropy collector instance MUST be deallocated and a fresh instance must be allocated.

7.2 Key Generation

For generating RSA, DSA, ECDSA, Diffie-Hellman and EC Diffie-Hellman keys, the SummitSSL component of the module implements asymmetric key generation services compliant with FIPS186-4 or SP800-56Arev3 as applicable and using a DRBG compliant with SP800-90A. The random value used in asymmetric key generation is obtained from the DRBG. In accordance with FIPS 140-2 IG D.12, the cryptographic module performs Cryptographic Key Generation (CKG) for asymmetric keys as per SP800-133 (vendor affirmed).

Symmetric keys are derived from the shared secret established by Diffie-Hellman and EC Diffie-Hellman in a manner that is compliant to NIST SP800-135 for TLS KDF. Symmetric keys can also be derived by means of the IEEE 802.11 protocols CCMP and GCMP, compliant to NIST SP800-108 KDF.

7.3 Key Entry/Output

The module does not support manual key entry or intermediate key generation output. The module does not produce key output outside its physical boundary.

The keys are entered to the module in plaintext form via API parameters, and output from the module via API output parameters in plaintext. Both these operations occur between the module and the calling application only.

7.4 Key/CSP Storage

Public and private keys are provided to the module by the calling process and are destroyed when released by the appropriate API function calls. The module does not perform persistent storage of keys. The only exception is the HMAC key used for integrity test, which is stored in the module's file system. The HMAC key is used solely for the integrity check and cannot be exported from the module or read by user APIs.

7.5 Key/CSP Zeroization

For the SummitSSL component, a general RAM zeroization API is provided: `sl_DeviceSet(SL_DEVICE_FIPS, SL_DEVICE_FIPS_ZEROIZATION, 0, NULL)`. The API call zeroizes all the RAM, and thus zeroizes all the keys and CSPs.

The kernel component provides two zeroization APIs: `crypto_free_cipher()`, and `crypto_free_aead()`. Both these functions invoke `crypto_free_tfm()`, which will zeroize the context and free the cipher handle.

Zeroization of all keys and CSPs in RAM can also be obtained by powering off the module, and then powering the module back on (power cycle).

The application is responsible for calling the appropriate destruction functions from the SummitSSL API and kernel API. The destruction functions then overwrite the memory occupied by keys with

zeros and deallocates the memory with the free() call. In case of abnormal termination, the keys in physical memory are overwritten by the kernel before the physical memory is allocated to another process.

7.6 Key Establishment

The module provides Diffie-Hellman and EC Diffie-Hellman shared secret computation compliant with SP800-56Arev3 in accordance with Scenario X1 (1) of IG D.8. The module provides support for Diffie-Hellman an EC Diffie-Hellman key agreement schemes compliant with SP800-56Arev3 by offering separate services for shared secret computation and the key derivation using the SP800-135 TLS KDF (for use within the TLS protocol), so that the user application can implement the key agreement. In addition, the module offers AES key wrapping per SP800-38F and RSA key wrapping (encapsulation) using public key encryption and private key decryption primitives as allowed by IG D.9.

The module provides approved key transport methods according to IG D.9. Even though the module does not implement the GCMP and TLS protocols, the module claims compliance for AES-GCM under IG A.5 under the context of TLS and GCMP usage (Section 10.2.2). As such, there is a scenario under which the AES-GCM algorithm can be used as key transport for an application that implements the GCMP and TLS protocols. Therefore, the module implements the key transport methods by:

- Using an approved key wrapping algorithm (AES-KW, AES-KWP).
- Use of the approved AES-GCM authenticated encryption mode under the context of an application establishing a connection using the wireless protocol GCMP or using a TLS ciphertext with the AES-GCM authenticated encryption mode. Note that in a GCMP connection, the AES-GCM encryption is used.

Table 7 and Table 8 specify the key sizes allowed in the FIPS mode of operation. According to Table 2 in SP800-57, the key sizes of key wrapping, transport, and shared secret computation (using the respective symmetric algorithm, RSA, Diffie-Hellman, and EC Diffie-Hellman) provide the following security strengths:

- AES key wrapping provides between 128 and 256 bits of encryption strength.
- RSA PKCS 1.5 key wrapping provides between 112 and 256 bits of encryption strength.
- Diffie-Hellman shared secret computation provides between 112 and 200 bits of encryption strength.
- EC Diffie-Hellman shared secret computation provides between 112 and 256 bits of encryption strength.
- Use of approved authenticated encryption mode (AES-GCM) within GCMP and TLS. In these contexts, the key establishment methodology provides 128 or 256 bits of encryption strength.

No parts of the TLS protocol, other than the KDF, have been tested by the CAVP and CMVP.

8 Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)

The test platforms listed in Table 3 have been tested and found to conform to the EMI/EMC requirements specified by 47 Code of Federal Regulations, FCC PART 15, Subpart B, Unintentional Radiators, Digital Devices, Class B (i.e., Home use). These devices are designed to provide reasonable protection against harmful interference when the devices are operated in a commercial environment.

9 Self-Tests

9.1 Power-Up Self-Tests

The module performs power-up self-tests (POSTs) automatically when the module is powered on. These POSTs ensure that the module is not corrupted and that the cryptographic algorithms work as expected. No operator intervention is necessary to run the POSTs.

While the module is executing the POSTs, services are not available, and input and output are inhibited. The module is not available for use until successful completion of the POSTs.

The integrity of the module's software components (the kernel and the SummitSSL components) is individually verified by the fipscheck integrity test tool using an HMAC-SHA2-256. The HMAC value of each software component is computed at build time and stored in the .hmac file for each component. The value is recalculated at runtime for the image of the kernel and for the SummitSSL binary, and then compared against the stored value in the file. If the comparison succeeds, then the remaining POSTs (consisting of the algorithm-specific Known Answer Tests) for SummitSSL are performed. The kernel component executes its algorithm-specific Known Answer Tests before the fipscheck integrity test tool executes to verify the integrity of the kernel.

The kernel component triggers the execution of the fipscheck tool during boot without operator assistance, as part of the kernel boot routine. The SummitSSL component triggers the execution of the fipscheck tool upon loading through a DEP (Default Entry Point) mechanism, again without operator assistance.

On successful completion of all the power-up tests, the module becomes operational and cryptographic services are then available. If any of the tests fails, the module transitions to the error state and subsequent calls to the module will fail. The status of the module can be determined by the availability of the module. If the module is available, then it has passed all self-tests. If the module is unavailable, it is because the POST procedure failed, and the module has transitioned to the error state. Thus, in the error state, no further cryptographic operations will be possible.

Table 11 details the self-tests that are performed on the FIPS-approved cryptographic algorithms supported in the FIPS-approved mode of operation, using the Known-Answer Tests (KATs) and Pairwise Consistency Tests (PCTs).

Table 11: Self-tests.

Algorithm	Test
Kernel Component	
AES	• KAT AES(CCM) with 256-bit key, encryption • KAT AES(ECB) with 128-bit key, encryption and decryption
SummitSSL Component	
AES	• KAT AES(GCM) with 256-bit key, encryption • KAT AES(ECB) with 128-bit key, decryption
DSA	• PCT DSA signature generation and verification with L=2048, N=224 and SHA2-256
RSA	• KAT RSA PSS signature generation and verification with 2048-bit key SHA2-256
ECDSA	• PCT ECDSA signature generation and verification with P-224 and K-233, both with SHA2-512

Algorithm	Test
KAS-FFC-SSC (Diffie-Hellman)	• Primitive "Z" Computation KAT with 2048-bit key
KAS-ECC-SSC (EC Diffie-Hellman)	• Primitive "Z" Computation KAT with P-224 curve
DRBG	• KAT CTR_DRBG using AES-256 with and without DF, with and without PR, and health tests per Section 11.3 of SP800-90A
KBKDF	• KAT with HMAC-SHA2-256
KDF in TLS v1.0/1.1	• KAT with HMAC-SHA-1
KDF in TLS v1.2	• KAT with HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512
HMAC	• KAT HMAC-SHA-1 • KAT HMAC-SHA2-256 • KAT HMAC-SHA2-512
SHS	• KAT SHA-1
Module Integrity	• HMAC-SHA2-256

9.2 Conditional Self-Tests

Conditional tests are performed during operational state of the module when the respective cryptographic functions are used. If any of the conditional tests fails, the module transitions to the error state.

Table 12 lists the conditional self-tests performed by the functions.

Table 12: Conditional self-tests.

Algorithm	Test
DSA Key generation	PCT using SHA2-256, signature generation and verification
ECDSA Key generation	PCT using SHA2-256, signature generation and verification
RSA Key generation	PCT using SHA2-256, signature generation and verification, and for encryption and decryption
Entropy source	Continuous health tests (Repetition Count Test and Adaptive Proportion Test)

9.3 On-Demand Self-tests

The module provides the Self-Test service to perform self-tests on demand. On demand self-tests can be invoked by powering-off and powering-on the module. This service performs the same cryptographic algorithm tests executed during power-up. During the execution of the on-demand self-tests, cryptographic services are not available, and no data output or input is possible.

10 Guidance

This section provides guidance for the Crypto Officer and the User to maintain proper use of the module per FIPS 140-2 requirements.

10.1 Crypto-Officer Guidance

Before deploying the module for usage, the Crypto Officer shall employ the following steps:

1. Verify the HMAC values of each component of the module as listed in Table 2.
2. Verify that the kernel component command line is configured to run `fipsInit.sh` before any user mode application or init system.
3. Verify that `'fips=1'` parameter is present on the kernel command line for FIPS mode operation.

10.2 User Guidance

As specified in Section 2.5, the mode of operation of this module is implicitly selected depending upon which security functions or services and key sizes or curves are being used, and the invocation of FIPS_mode_set(1) or FIPS_mode_set(0) command prior to the cryptographic service.

To run the module in FIPS mode, the user shall follow the rules detailed in Section 2.5 and only use the FIPS approved or allowed services listed in Table 5, or the validated or allowed cryptographic algorithms and security functions listed in Table 7 and Table 8.

10.2.1 Random Number Generator

The SummitSSL API call of RAND_cleanup must not be used. This call will clean up the internal DRBG state. This call also replaces the DRBG instance with the non-FIPS Approved libcrypto Deterministic Random Number Generator when using the RAND* API calls.

10.2.2 AES GCM IV

AES-GCM encryption and decryption are used in the context of the TLS protocol version 1.2 using the SummitSSL component (corresponding to Scenario 1 of IG A.5), and in the context of IEEE 802.11 GCMP using the kernel component (corresponding to Scenario 5 of IG A.5).

10.2.2.1 TLS version 1.2

For TLS v1.2, the module uses the context of Scenario 1 of IG A.5. The module is compliant with SP800-52 and the mechanism for IV generation is compliant with RFC5288. For this compliance, the module's implementation of the AES-GCM shall be used together with an application that negotiates the protocol session's keys and the 32-bit nonce value of the IV. The nonce is considered the "name" field in Scenario 3 of IG A.5. The setting of the counter portion of the IV is performed within the cryptographic boundary.

The nonce explicit part of the IV does not exhaust the maximum number of possible values for a given session key. This condition is implicitly ensured by the design of the TLS protocol, in which the nonce_explicit is denied exhaustion by the control exerted by the protocol's management logic (wherein the nonce_explicit is incremented per each TLS record). This management logic also implies that the probability of an exhaustion of all $2^{64} - 1$ values of the nonce_explicit for the same TLS session in a realistic time frame is not significant.

No parts of the TLS protocol, other than the KDF, have been tested by the CAVP and CMVP.

10.2.2.2 IEEE 802.11 GCMP

For IEEE 802.11 GCMP, the module complies with Scenario 5 of IG A.5 for use of the AES-GCM algorithm within the context of GCMP. The module implements an internal production unit logic that constructs the IV deterministically upon the initialization of a GCMP connection, and therefore

the initialization of a GCM encryption context. The 96-bit IV is divided into a 48-bit Transmitter Address field, and a 48-bit Packet Number (PN) field.

The module obtains the Transmitter Address field from the network interface (the wireless network adapter) that is part of the operational environment of the module. This Address field is typically an IEEE 802 Medium Access Control (MAC) address that uniquely identifies the module during the GCMP connection and remains the same value throughout the lifetime of the connection. (In Scenario 3 of FIPS 140-2 IG A.5, this Address field corresponds to the “name” field.)

The 48-bit Packet Number field is deterministically constructed by the module as a counter field, starting at 1 and strictly incrementing by 1 at each invocation of the GCMP encryption in the context of the GCMP connection. The counter is never allowed to repeat for the same context. If the maximum number of values for the counter is exhausted, the module refuses to offer the GCM encryption service, and thus the GCMP context is aborted. (This Packet Number field corresponds to the deterministic non-repetitive counter in Scenario 3 of FIPS 140-2 IG A.5.) The combined length from the 48-bit Address field and 48-bit Packet Number forms the GCM IV of 96 bits.

The module also receives the GCM IV from the GCMP layer outside of the boundary of the module and verifies whether the IV as computed by the GCMP layer matches the IV as constructed internally by the module. If there is a mismatch, the module does not allow the GCM encryption service to be provided and the GCMP context is aborted.

To invoke the compliant GCM in the kernel component, the driver names for encrypt and decrypt are indicated in Table 7.

In case the module’s power is lost and then restored, the key used for AES GCM encryption or decryption shall be re-distributed.

10.2.3 AES-XTS

The AES algorithm in XTS mode can be only used for the cryptographic protection of data on storage devices, as specified in SP800-38E. In addition, the length of a single data unit encrypted with the XTS-AES shall not exceed 2^{20} AES blocks, that is, 16 MiB of data.

In addition, to meet the requirement in IG A.9, the module implements a check to ensure that the two AES keys used in XTS-AES algorithm are not identical.

10.2.4 Key Usage and Management

In general, a single key shall be used for only one purpose (e.g., encryption, integrity, authentication, key wrapping, random bit generation, or digital signatures) and be disjoint between the modes of operations of the module. Thus, if the module is switched between its FIPS mode and non-FIPS mode or vice versa, the following procedures shall be observed:

- The DRBG engine shall be reseeded.
- CSPs and keys shall not be shared between security functions of the two different modes.

10.3 Handling Self-Test Errors

The module transitions to the error state when any of the self-tests or conditional tests fails in any of the components of the module. In such a case, the module, while in the error state, inhibits output and makes no cryptographic service available. After logging the error, the module then automatically reboots in an attempt to recover from the errors.

Following are the error messages specific to self-test failure as reported by the SummitSSL component.

- FIPS_R_FINGERPRINT_DOES_NOT_MATCH: The integrity verification check failed.
- FIPS_R_SELFTEST_FAILED: a known answer test failed.
- FIPS_R_TEST_FAILURE: a known answer test failed (RSA); pairwise consistency test failed (DSA).

- FIPS_R_PAIRWISE_TEST_FAILED: a pairwise consistency test failed during DSA, ECDSA or RSA key generation.

These errors are reported through the regular ERR interface of the module.

The kernel component reports self-test errors through the dmesg logs, indicating the timestamp and the algorithm that failed the self-test. Two examples are provided below.

```
[    0.197191] alg: skcipher: Test 1 failed (invalid result) on encryption for
ctr(aes-arm)

[    2.990814] alg: aead: Test 1 failed on encryption for ccm_base(ctr(aes-
arm),cbcmac(aes-arm))
```

The fipscheck tool reports errors found during integrity test. Examples are provided below.

```
fipscheck: Hmac mismatch on file
{'/tmp/Image.lzma'|'/usr/bin/fipscheck'|'/lib/libfipscheck.so.1'|'/usr/lib/libcrypt
o.so.1.0.0'}

fipscheck: Cannot open hmac file
{'/tmp/Image.lzma'|'/usr/bin/fipscheck'|'/lib/libfipscheck.so.1'|'/usr/lib/libcrypt
o.so.1.0.0'}

FIPS Integrity check Failed: fipscheck error:
reboot: Restarting system
```

The only way to recover from the error state is through rebooting the module and passing the power-on self-tests (which will be triggered upon the automatic reboot when the module transitions to the error state). If failures persist, the module shall be reinstalled. If, after reinstallation, the module still accuses failures, then the module shall be decommissioned.

11 Mitigation of Other Attacks

The vendor does not claim any mitigation of other attacks for this module.

12 Acronyms, Terms and Abbreviations

Term	Definition
AES	Advanced Encryption Standard
CAVP	Cryptographic Algorithm Validation Program
CMVP	Cryptographic Module Validation Program
CSP	Critical Security Parameter
DH	Diffie-Hellman
DHE	Diffie-Hellman Ephemeral
DRBG	Deterministic Random Bit Generator
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
HMAC	(Keyed) Hash Message Authentication Code
KAT	Known Answer Test
KDF	Key Derivation Function
NIST	National Institute of Standards and Technology
POST	Power-On Self Test
PR	Prediction Resistance
PSS	Probabilistic Signature Scheme
PUB	Publication
SHA2	Secure Hash Algorithm
TLS	Transport Layer Security

13 Algorithm Implementations

Table 13 describes the names utilized in the algorithm certificates and the implementations to which they refer for the test platforms.

Table 13: Algorithm implementations and their names in the CAVP certificates.

Name	Description
AES_C_GCM	Generic C AES-GCM implementation
AES_C	Generic C AES implementation
C_C	Cipher in C, block modes in C
ARMASM_C	Assembler AES implementation and generic C block mode
KBKDF	Generic C non-optimized KBKDF implementation
FFC_DH	Generic C non-optimized DH implementation
SHA_ASM	Assembler SHA implementation (when SHA is used)
CTS_ARMASM_C	CBC-CS generic C non-optimized block mode using assembler AES implementation
CTS_C_C	AES-CBC-CS with AES using C, block mode using C

14 References

Barker, E., & Roginsky, A. (2015, 11). SP 800-131A Revision 1. Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths. National Institute of Standards & Technology. Retrieved from http://dx.doi.org/10.6028/NIST.SP.800-131Ar1

IEEE802.11, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. (2016, 8). IEEE.

National Institute of Standards and Technology. (2000, January 27). *FIPS PUB 186-2: Digital Signature Standard (DSS).* Retrieved April 01, 2019, from https://csrc.nist.gov/csrc/media/publications/fips/186/2/archive/2000-01-27/documents/fips186-2.pdf

National Institute of Standards Technology. (2001, 5 25). FIPS PUB 140-2. Security Requirements for Cryptographic Modules. Retrieved from https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf

National Institute of Standards Technology. (2008, 7). FIPS PUB 198-1. The Keyed-Hash Message Authentication Code (HMAC). Retrieved from http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf

National Institute of Standards Technology. (2012, 3). FIPS PUB 180-4. Secure Hash Standard (SHS). Gaithersburg, MD 20899-8900: National Institute of Standards & Technology. Retrieved from http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf

National Institute of Standards Technology. (2013, July). FIPS PUB 186-4. Digital Signature Standard (DSS). https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.186-4.pdf.

National Institute of Standards Technology. (2016, 12 21). Annex B: Approved Protection Profiles for FIPS PUB 140-2, Security Requirements for Cryptographic Modules. Retrieved from https://csrc.nist.gov/CSRC/media/Publications/fips/140/2/final/documents/fips1402annexa.pdf

National Institute of Standards Technology. (2019, August 16). *Implementation Guidance for FIPS 140-2 and the Cryptographic Module Validation Program.* Retrieved August 27, 2019, from https://csrc.nist.gov/csrc/media/projects/cryptographic-module-validation-program/documents/fips140-2/fips1402ig.pdf