Aviat Networks

WTM 4000 module

FIPS 140-3 Non-Proprietary Security Policy

Document Version 1.0
23 May 2025

AVIAT NETWORKS
200C Parker Drive, Suite 100A
Austin, TX 78728
aviatnetworks.com
+1 (512) 265-3680

Table of Contents

List of Tables

List of Figures

Author(s)	Title	Date	Version	Description
Ruth French	Senior Manager, Regulatory Compliance	23 May 2025	1.0	Initial Release

Table 1: This Document History

1 – General

1.1 Overview

This document defines the Security Policy for AVIAT NETWORKS WTM 4000 module, hereafter denoted the Module. The Module meets FIPS 140-3 overall Level 1 requirements, with security levels as described in section 1.2 below.

1.2 Security Levels

Section	Title	Security Level
1	General	1
2	Cryptographic module specification	1
3	Cryptographic module interfaces	1
4	Roles, services, and authentication	1
5	Software/Firmware security	1
6	Operational environment	1
7	Physical security	N/A
8	Non-invasive security	N/A
9	Sensitive security parameter management	1
10	Self-tests	1
11	Life-cycle assurance	1
12	Mitigation of other attacks	N/A
	Overall Level	1

Table 2: Security Levels

1.3 Additional Information

In accordance with AS02.05, [ISO19790] §7.7 Physical Security is optional and does not apply to the Module.

Term/Ref	Description
[140-3]	FIPS 140-3, *Security Requirements for Cryptographic Modules*
[OE]	The “Operating Environment”
[186-4]	FIPS 186-4, *Digital Signature Standard (DSS)*
[90Arev1]	NIST SP 800-90A Rev. 1, *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*
[56Arev3]	NIST SP 800-56A Rev. 3, *Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography*
[56Crev2]	NIST SP 800-56C Rev. 2, *Recommendation for Key-Derivation Methods in Key-Establishment Schemes*
[135rev1]	NIST SP 800-135 Rev. 1, *Recommendation for Existing Application-Specific Key Derivation Functions*

Term/Ref	Description
[140Drev2]	NIST SP 800-140D revision 2, *CMVP Approved Sensitive Parameter Generation and Establishment Methods: CMVP Validation Authority Updates to ISO/IEC 24759*
[UG]	AVIAT NETWORKS FIPS 140-3 User Guide (sometimes referred to as the “Cryptographic Officer Guidance Manual” in documentation not produced by this vendor)
[COGM]	Cryptographic Officer Guidance Manual (Another term for [UG] recognized by some in the Industry. Same meaning as [UG]
[140-3 IG]	FIPS 140-3, Implementation Guidance
[131Arev2]	NIST SP 800-131A Rev. 2, *Transitioning the Use of Cryptographic Algorithms and Key Lengths*
[56Brev2]	NIST SP 800-56B Rev. 2, *Recommendation for Pair-Wise Key-Establishment Using Integer Factorization Cryptography*

Table 3: Legend of Terms and references that appear in this document

2 – Cryptographic Module Specification

2.1 Description

TOEPP: The platform(s) used for testing are documented in Table 6: Tested Operational Environments - Software, Firmware, Hybrid. If the onboard CPU of a tested platform supported a known PAA [FIPS 140-3 IG 2.3.C] and was desirable for FIPS use, then in accordance with [FIPS 140-3 IG2.3.C] that platform was tested both with and without PAA unless an identical or similar platform had already been tested. When an identical or similar platform was already tested, the new platform was tested only with PAA. This is reflected by the column PAA/PAI in table 6 as marked with a Yes or No entry. The Intel and AMD AESNI (AES New Instructions) are known PAA(s).

Purpose and Use:

The Module is a cryptography software library. The Module is a Multi-Chip Stand Alone embodiment. The Module is intended for use by U.S. and Canadian Federal agencies in addition to any other markets that require FIPS 140-3 validated cryptographic functionality. The Module was originally designed with embedded and IoT in mind. As a side effect of this design, it also scales exceptionally well on larger desktop and server systems allowing more connections per box than similar competing solutions.

The Module version under validation is Software Version v5.2.1.

Module Type: Software

Module Embodiment: MultiChipStand

Module Characteristics:

Cryptographic Boundary:

Figure 1 depicts the Module operational environment, with the software module cryptographic boundary highlighted in red inclusive of all Module entry points (API calls). The Module is defined as a *Software module* per AS02.03. No components are excluded from [140-3] requirements. The pre-operational approved integrity test is performed over all components of the cryptographic boundary. Updates to the Module are provided as a complete replacement in accordance with AS04.27 – AS04.35.

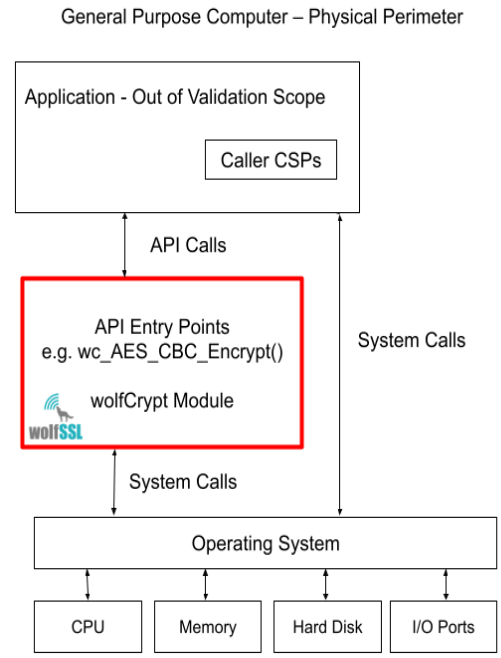

Figure 1: Module Block Diagram

Source File Name	Description
aes.c	AES algorithm
aes_asm.s	AES assembler optimizations (Linux)
aes_asm.asm	AES assembler optimizations (Windows 10)
cmac.c	CMAC algorithm
dh.c	Diffie-Hellman
ecc.c	Elliptic curve cryptography
fips.c	Pre-operational entry point and API wrappers
fips_test.c	Power on self-tests
hmac.c	HMAC algorithm
kdf.c	TLS v1.2, v1.3 and SSH v2 KDFs
random.c	DRBG algorithm
rsa.c	RSA algorithm
sha.c	SHA algorithm
sha256.c	SHA-256 algorithm
sha256_asm.s	SHA-256 assembler optimizations (Linux)
sha512_asm.s	SHA-512 assembler optimizations (Linux)
sha3.c	SHA-3 algorithm
sha512.c	SHA-512 algorithm
wolfcrypt_first.c	First function and Read Only address marking start of cryptographic boundary
wolfcrypt_last.c	Last function and Read Only address marking end of cryptographic boundary

Table 4: Source Files

The source code files listed in Table “Source Files” result in the corresponding object files that comprise the WTM 4000 module boundary on each supported operating environment; the extensions of the object file can differ across environments.

2.2 Tested and Vendor Affirmed Module Version and Identification

Tested Module Identification – Hardware:

N/A for this module.

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

Package or File Name	Software/ Firmware Version	Features	Integrity Test
wolfssl-5.6.3-commercial-fips-linuxv5.2.1.7z	v5.2.1	FIPS 140-3 module and SSL/TLS library	HMAC-SHA256

Table 5: Tested Module Identification – Software, Firmware, Hybrid (Executable Code Sets)

Tested Module Identification – Hybrid Disjoint Hardware:

N/A for this module.

Tested Operational Environments - Software, Firmware, Hybrid:

Operating System	Hardware Platform	Processors	PAA/PAI	Hypervisor or Host OS	Version(s)
Linux 5.4	WTM 4100	Broadcom BCM56260B0IFSBG - Saber2	No		v5.2.1

Table 6: Tested Operational Environments - Software, Firmware, Hybrid

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

N/A for this module.

The Module conforms to [140-3 IG] 2.3.C *Processor Algorithm Accelerators (PAA) and Processor Algorithm Implementation (PAI).* The Intel Processor AES-NI functions are identified by [140-3 IG] 2.3.C as a known PAA.

No vendor affirmed operational environments are claimed for this validation of the module.

2.3 Excluded Components

N/A the module does not support excluded components.

2.4 Modes of Operation

Modes List and Description:

Mode Name	Description	Type	Status Indicator
Approved mode of operation	The Module supports an Approved mode of operation. In this mode all services are available.	Approved	FIPS_MODE_NORMAL (1)
Degraded mode of operation	The Module implements a Degraded Mode of operation: when a CAST fails, that CAST is marked as failed and the module will inhibit use of algorithms governed by that CAST	Approved	FIPS_MODE_DEGRADED (2)

Table 7: Modes List and Description

Mode Change Instructions and Status:

Each time the module is power cycled or reloaded all CAST status are initialized to FIPS_CAST_STATE_INIT.

Each algorithm invocation includes a check of the algorithms CAST status; if the CAST status is FIPS_CAST_STATE_INIT the module will automatically run the CAST and that algorithms CAST status will be updated to either FIPS_CAST_STATE_SUCCESS (if it passes) or FIPS_CAST_STATE_FAILURE (if it fails). See degraded mode for when a CAST status fails.

To check the modules overall status at any time the cryptographic officer may use the Show Status service by calling wolfCrypt_GetMode_fips() this will return either:

FIPS_MODE_INIT (0) - Module is currently running its' pre-operational self-test in another thread (multi-threaded)
FIPS_MODE_NORMAL (1) - Module in normal mode of operation without errors
FIPS_MODE_DEGRADED (2) - Module in degraded mode of operation with some errors
FIPS_MODE_FAILED (3) - Module failed the integrity check and is not usable

To check the CAST state of any algorithm the cryptographic officer may use the Show Status Service by calling wc_GetCastStatus_fips(<algorithm type>) where algorithm type can be any of the following:

- FIPS_CAST_AES_CBC
- FIPS_CAST_AES_GCM
- FIPS_CAST_HMAC_SHA1
- FIPS_CAST_HMAC_SHA2_256
- FIPS_CAST_HMAC_SHA2_512
- FIPS_CAST_HMAC_SHA3_256
- FIPS_CAST_DRBG
- FIPS_CAST_RSA_SIGN_PKCS1v15
- FIPS_CAST_ECC_CDH
- FIPS_CAST_ECC_PRIMITIVE_Z
- FIPS_CAST_DH_PRIMITIVE_Z
- FIPS_CAST_ECDSA
- FIPS_CAST_KDF_TLS12
- FIPS_CAST_KDF_TLS13
- FIPS_CAST_KDF_SSH

The returned status indicator of wc_GetCastStatus_fips(<algorithm type>) may be checked against any of the following states:

- FIPS_CAST_STATE_INIT (0) - CAST hasn't run yet
- FIPS_CAST_STATE_PROCESSING (1) - CAST is running
- FIPS_CAST_STATE_SUCCESS (2) - CAST has passed previously
- FIPS_CAST_STATE_FAILURE (3) - CAST has failed

Degraded Mode Description:

The Module implements a degraded mode of operation: when a CAST fails, the module enters an error state. The algorithm CAST status is set to FIPS_CAST_STATE_FAILED and the module runs all CASTS prior to the first operational use of any algorithm, regardless of the CAST having passed previously. Before exiting the error state, the module status (reported in the Show Status service) is set to FIPS_MODE_DEGRADED. Upon exiting the error state, the module enters the degraded mode of operation. This sequence of events is in accordance with AS02.26. The algorithm that failed its' CAST initially triggering the error state will no longer be available for use in degraded mode of operation and any algorithms that depend on that algorithm will also be unavailable for use. See Table 16: Conditional Self-Tests in section 10.2, column Conditions to see if a CAST failure will affect use of another algorithm. To recover from degraded mode of operation CO *shall* power cycle or reload the module (equivalent to a power cycle).

2.5 Algorithms

Approved Algorithms:

Algorithm	CAVP Cert	Properties	Reference
AES-CBC	A4308	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-CCM	A4308	Key Length - 128, 192, 256	SP 800-38C
AES-CMAC	A4308	Direction - Generation, Verification Key Length - 128, 192, 256	SP 800-38B
AES-CTR	A4308	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-ECB	A4308	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-GCM	A4308	Direction - Decrypt, Encrypt IV Generation - External, Internal IV Generation Mode - 8.2.1, 8.2.2 Key Length - 128, 192, 256	SP 800-38D
AES-GMAC	A4308	Direction - Decrypt, Encrypt IV Generation - External, Internal IV Generation Mode - 8.2.1, 8.2.2 Key Length - 128, 192, 256	SP 800-38D
AES-OFB	A4308	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
DSA KeyGen (FIPS186-4)	A4308	L - 2048 N - 256	FIPS 186-4
ECDSA KeyGen (FIPS186-4)	A4308	Curve - P-224, P-256, P-384, P-521 Secret Generation Mode - Extra Bits	FIPS 186-4
ECDSA KeyVer (FIPS186-4)	A4308	Curve - P-192, P-224, P-256, P-384, P-521	FIPS 186-4

Algorithm	CAVP Cert	Properties	Reference
ECDSA SigGen (FIPS186-4)	A4308	Component - No Curve - P-224, P-256, P-384, P-521 Hash Algorithm - SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512	FIPS 186-4
ECDSA SigVer (FIPS186-4)	A4308	Component - No Curve - P-192, P-224, P-256, P-384, P-521 Hash Algorithm - SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512	FIPS 186-4
Hash DRBG	A4308	Prediction Resistance - No Mode - SHA2-256	SP 800-90A Rev. 1
HMAC-SHA-1	A4308	Key Length - Key Length: 112-1024 Increment 8	FIPS 198-1
HMAC-SHA2-224	A4308	Key Length - Key Length: 112-1024 Increment 8	FIPS 198-1
HMAC-SHA2-256	A4308	Key Length - Key Length: 112-1024 Increment 8	FIPS 198-1
HMAC-SHA2-384	A4308	Key Length - Key Length: 112-1024 Increment 8	FIPS 198-1
HMAC-SHA2-512	A4308	Key Length - Key Length: 112-1024 Increment 8	FIPS 198-1
HMAC-SHA3-224	A4308	Key Length - Key Length: 112-1024 Increment 8	FIPS 198-1
HMAC-SHA3-256	A4308	Key Length - Key Length: 112-1024 Increment 8	FIPS 198-1
HMAC-SHA3-384	A4308	Key Length - Key Length: 112-1024 Increment 8	FIPS 198-1
HMAC-SHA3-512	A4308	Key Length - Key Length: 112-1024 Increment 8	FIPS 198-1
KAS-ECC-SSC Sp800-56Ar3	A4308	Domain Parameter Generation Methods - P-256, P-384, P-521 Scheme - ephemeralUnified - KAS Role - initiator, responder	SP 800-56A Rev. 3
KAS-FFC-SSC Sp800-56Ar3	A4308	Domain Parameter Generation Methods - ffdhe2048 Scheme - dhEphem - KAS Role - initiator, responder	SP 800-56A Rev. 3
KDF SSH (CVL)	A4308	Cipher - AES-128, AES-192, AES-256 Hash Algorithm - SHA-1, SHA2-256, SHA2-384, SHA2-512	SP 800-135 Rev. 1
KDF TLS (CVL)	A4308	TLS Version - v1.2 Hash Algorithm - SHA2-256, SHA2-384, SHA2-512	SP 800-135 Rev. 1

Algorithm	CAVP Cert	Properties	Reference
RSA KeyGen (FIPS186-4)	A4308	Key Generation Mode - B.3.3 Modulo - 2048, 3072, 4096 Primality Tests - Table C.2 Private Key Format - Standard	FIPS 186-4
RSA SigGen (FIPS186-4)	A4308	Signature Type - PKCS 1.5, PKCSPSS Modulo - 2048, 3072, 4096	FIPS 186-4
RSA SigVer (FIPS186-4)	A4308	Signature Type - PKCS 1.5, PKCSPSS Modulo - 1024, 2048, 3072, 4096	FIPS 186-4
SHA-1	A4308	Message Length - Message Length: 0-65536 Increment 8	FIPS 180-4
SHA2-224	A4308	Message Length - Message Length: 0-65536 Increment 8	FIPS 180-4
SHA2-256	A4308	Message Length - Message Length: 0-65536 Increment 8	FIPS 180-4
SHA2-384	A4308	Message Length - Message Length: 0-65536 Increment 8	FIPS 180-4
SHA2-512	A4308	Message Length - Message Length: 0-65536 Increment 8	FIPS 180-4
SHA3-224	A4308	Message Length - Message Length: 0-65536 Increment 8	FIPS 202
SHA3-256	A4308	Message Length - Message Length: 0-65536 Increment 8	FIPS 202
SHA3-384	A4308	Message Length - Message Length: 0-65536 Increment 8	FIPS 202
SHA3-512	A4308	Message Length - Message Length: 0-65536 Increment 8	FIPS 202
TLS v1.2 KDF RFC7627 (CVL)	A4308	Hash Algorithm - SHA2-256, SHA2-384, SHA2-512	SP 800-135 Rev. 1
TLS v1.3 KDF (CVL)	A4308	HMAC Algorithm - SHA2-256, SHA2-384 KDF Running Modes - DHE, PSK, PSK-DHE	SP 800-135 Rev. 1

Table 8: Approved Algorithms

NOTE: Only the algorithms specified in this section are supported by the module in approved mode of operation.

No operational use of an algorithm may be performed until the corresponding CAST has passed.

Vendor-Affirmed Algorithms:

Name	Properties	Implementation	Reference
CKG-1	Asymmetric:RSA Asymmetric:ECDSA	Linux 4.4 (Ubuntu 16.04 LTS) with an Intel Core i5-5300U CPU @2.30GHz x 4 with PAA; Linux 4.4 (Ubuntu 16.04	SP800-133r2 5.1 "Key Pairs for Digital Signature Schemes"

Name	Properties	Implementation	Reference
		LTS) with an Intel Core i5-5300U CPU @2.30GHz x 4 without PAA	
CKG-2	Asymmetric:ECC Asymmetric:FFC	Linux 4.4 (Ubuntu 16.04 LTS) with an Intel Core i5-5300U CPU @2.30GHz x 4 with PAA; Linux 4.4 (Ubuntu 16.04 LTS) with an Intel Core i5-5300U CPU @2.30GHz x 4 without PAA	SP800-133r2 5.2 "Key Pairs for Key Establishment"
CKG-3	Symmetric:AES Symmetric:HMAC	Linux 4.4 (Ubuntu 16.04 LTS) with an Intel Core i5-5300U CPU @2.30GHz x 4 with PAA; Linux 4.4 (Ubuntu 16.04 LTS) with an Intel Core i5-5300U CPU @2.30GHz x 4 without PAA	SP800-133r2 6.2 "Derivation of Symmetric Keys"

Table 9: Vendor-Affirmed Algorithms

Non-Approved, Allowed Algorithms:

N/A for this module.

The Module does not implement non-approved algorithms. The services listed in this Security Policy include all cryptographic and non-cryptographic functionality.

NOTE: For TLS 1.2 KDF Extended master-secret *shall* be used in approved mode of operation.

Non-Approved, Allowed Algorithms with No Security Claimed:

N/A for this module.

Non-Approved, Not Allowed Algorithms:

N/A for this module.

2.6 Security Function Implementations

Name	Type	Description	Properties	Algorithms
DRBG	DRBG	Deterministic Random Byte Generator		SHA2-256: (A4308) A4308: Hash DRBG: (A4308) A4308:
Message Authentication	MAC	Hash-Based Message Authentication Codes, Generation and Verification		HMAC-SHA-1: (A4308) A4308: HMAC-SHA2-224: (A4308) A4308: HMAC-SHA2-

Name	Type	Description	Properties	Algorithms
				256: (A4308) A4308: HMAC-SHA2-384: (A4308) A4308: HMAC-SHA2-512: (A4308) A4308: HMAC-SHA3-224: (A4308) A4308: HMAC-SHA3-256: (A4308) A4308: HMAC-SHA3-384: (A4308) A4308: HMAC-SHA3-512: (A4308) A4308:
Secure Hash	SHA	Secure Hash Function		SHA-1: (A4308) A4308: SHA2-224: (A4308) A4308: SHA2-256: (A4308) A4308: SHA2-384: (A4308) A4308: SHA2-512: (A4308) A4308: SHA3-224: (A4308) A4308: SHA3-256: (A4308) A4308: SHA3-384: (A4308) A4308: SHA3-512: (A4308) A4308:
TLS 1.3 Key Agreement	KAS-56CKDF	KDF: Extract then Expand (56C)		TLS v1.3 KDF: (A4308) A4308:

Name	Type	Description	Properties	Algorithms
				HMAC-SHA2-256: (A4308) A4308: HMAC-SHA2-384: (A4308) A4308: HMAC-SHA2-512: (A4308) A4308:
Primitive Key Agreement	KAS-KeyGen	DH: Key agreement primitives		KAS-FFC-SSC Sp800-56Ar3: (A4308) A4308:
KDF Derived Key Agreement	KAS-135KDF	KDF: Derive keying material from a shared secret (135);		KDF SSH: (A4308) A4308: KDF TLS: (A4308) A4308: TLS v1.2 KDF RFC7627: (A4308) A4308: SHA-1: (A4308) A4308: SHA2-256: (A4308) A4308: SHA2-384: (A4308) A4308: SHA2-512: (A4308) A4308:
KAS SSC Derived Key Agreement	KAS-SSC	Derived keying material from a shared secret		KAS-ECC-SSC Sp800-56Ar3: (A4308) A4308: KAS-FFC-SSC Sp800-56Ar3: (A4308) A4308:
133r2 5.1 Asymmetric Key Generation	CKG	SP800-133r2 5.1 "Key Pairs for Digital Signature Schemes"		RSA KeyGen (FIPS186-4): (A4308) A4308: ECDSA KeyGen (FIPS186-4): (A4308)

Name	Type	Description	Properties	Algorithms
				A4308: Hash DRBG: (A4308) A4308: CKG-1: ()
133r2 5.2 Asymmetric Key Generation	CKG	SP800-133r2 5.2 "Key Pairs for Key Establishment"		KAS-ECC-SSC Sp800-56Ar3: (A4308) A4308: KAS-FFC-SSC Sp800-56Ar3: (A4308) A4308: Hash DRBG: (A4308) A4308: CKG-2: ()
Symmetric Key Generation	CKG	SP800-133r2 6.2 "Derivation of Symmetric Keys"		AES-CBC: (A4308) A4308: AES-CCM: (A4308) A4308: AES-CMAC: (A4308) A4308: AES-CTR: (A4308) A4308: AES-ECB: (A4308) A4308: AES-GCM: (A4308) A4308: AES-GMAC: (A4308) A4308: AES-OFB: (A4308) A4308: HMAC-SHA-1: (A4308) A4308: HMAC-SHA2-224: (A4308) A4308: HMAC-SHA2-256: (A4308)

Name	Type	Description	Properties	Algorithms
				A4308: HMAC-SHA2-384: (A4308) A4308: HMAC-SHA2-512: (A4308) A4308: HMAC-SHA3-224: (A4308) A4308: HMAC-SHA3-256: (A4308) A4308: HMAC-SHA3-384: (A4308) A4308: HMAC-SHA3-512: (A4308) A4308: Hash DRBG: (A4308) A4308: CKG-3: ()
RSA Asymmetric Key-Pair Generation	AsymKeyPair-KeyGen	Generate an RSA Asymmetric Key Pair		RSA KeyGen (FIPS186-4): (A4308) A4308: Hash DRBG: (A4308) A4308:
DSA Asymmetric Key-Pair Generation	AsymKeyPair-KeyGen AsymKeyPair-PubKeyVal AsymKeyPair-DomPar	Generate a DSA Asymmetric Key Pair, Validate a Public DSA Key and KAS-FFC-SSC Domain Parameter Generation (SP800-56Ar3)		KAS-FFC-SSC Sp800-56Ar3: (A4308) A4308: DSA KeyGen (FIPS186-4): (A4308) A4308: Hash DRBG: (A4308) A4308:
ECC Asymmetric Key-Pair Generation	AsymKeyPair-KeyVer AsymKeyPair-KeyGen AsymKeyPair-DomPar	Generate an ECC Asymmetric Key Pair, ECC KeyVer and KAS-ECC-SSC Domain Parameter		KAS-ECC-SSC Sp800-56Ar3: (A4308) A4308: ECDSA KeyGen (FIPS186-4): (A4308) A4308:

Name	Type	Description	Properties	Algorithms
		Generation (SP800-56Ar3)		Hash DRBG: (A4308) A4308:
Digital Signature Generation	DigSig-SigGen	Digital Signature Generation		RSA SigGen (FIPS186-4): (A4308) A4308: ECDSA SigGen (FIPS186-4): (A4308) A4308: SHA2-224: (A4308) A4308: SHA2-256: (A4308) A4308: SHA2-384: (A4308) A4308: SHA2-512: (A4308) A4308: SHA3-224: (A4308) A4308: SHA3-256: (A4308) A4308: SHA3-384: (A4308) A4308: SHA3-512: (A4308) A4308: Hash DRBG: (A4308) A4308:
Digital Signature Verification	DigSig-SigVer	Digital Signature Verification	DigSig-SigVer:1024 (verification only) DigSig-SigVer:SHA-1 (verification only) DigSig-SigVer:P-192 (Signature and	RSA SigVer (FIPS186-4): (A4308) A4308: ECDSA SigVer (FIPS186-4): (A4308) A4308: ECDSA KeyVer (FIPS186-4): (A4308)

Name	Type	Description	Properties	Algorithms
			Key Verification only)	A4308: SHA-1: (A4308) A4308: SHA2-224: (A4308) A4308: SHA2-256: (A4308) A4308: SHA2-384: (A4308) A4308: SHA2-512: (A4308) A4308: SHA3-224: (A4308) A4308: SHA3-256: (A4308) A4308: SHA3-384: (A4308) A4308: SHA3-512: (A4308) A4308:
Auth Block Cipher	BC-Auth	Authenticated Block Ciphers		AES-GMAC: (A4308) A4308: AES-GCM: (A4308) A4308: AES-CMAC: (A4308) A4308: AES-CCM: (A4308) A4308:
UnAuth Block Cipher	BC-UnAuth	Unauthenticated Block Ciphers		AES-CBC: (A4308) A4308: AES-ECB: (A4308) A4308: AES-OFB: (A4308) A4308: AES-CTR:

Name	Type	Description	Properties	Algorithms
				(A4308) A4308:

Table 10: Security Function Implementations

2.7 Algorithm Specific Information

The conditions for using the Module in the Approved mode of operation are:

1. The Module is a cryptographic library and it is intended to be used with a calling application. The calling application is responsible for the usage of the primitives in the correct sequence including the IVs and sessions.
2. The keys used by the Module for cryptographic purposes are determined by the calling application. The calling application is required to provide keys in accordance with [140Drev2].
3. With the Module installed and configured in accordance with [UG] instructions, only the algorithms listed in the table in Section 2.5 are available. The module is in the Approved mode if the following conditions for algorithm use are met. NOTE: All conditions and restrictions below are met when the executable binary is built in accordance with the UG instructions. Applications that would be at risk of violating any restriction in this section will fail to build and link successfully against the compliant module binary executable.
    a. Adherence to [140-3 IG] C.H *Key/IV Pair Uniqueness Requirements from SP 800-38D*. The Module supports both internal IV generation (for use with the [56Arev3] compliant KAS API entry points) and external IV generation (for TLS KAS usage). For internal IV generation, the Module complies with C.H 2, users MUST specify an IV length of GCM_NONCE_MID_SZ or greater for internal IV generation otherwise specifying any length less than 96-bits is rejected by the module. For internal IV generation, C.H requires the calling application to use the modules internal approved DRBG to generate the random IV For external IV generation, the Module complies with C.H 1 (a), tested per option (ii) under C.H **TLS protocol IV generation**. The module performs a check for nonce_explicit rollover, returning an error if that condition is encountered.
    b. ECDSA and RSA signature generation must be used with a SHA-2 or SHA-3 hash function.
    c. RSA signature generation and encryption primitives must use RSA keys with k = 2048, 3072 or 4096 bits or greater.
    d. The calling process shall adhere to all current [131Arev2] algorithm usage restrictions.
4. Manual key entry is not supported.
5. Data output is inhibited during self-tests, zeroization, and error states.
6. RSA Decrypt Primitive (RSADP) with k=2048-bit is the only CAVP testable aspect of [56Brev2]. The module implements the RSA primitive operations only, there are no claims of key transport. The module implements ‘RSA Encrypt Primitive’ (RSAEP) and RSADP. The vendor affirms conformance to [56Brev2] for RSAEP and RSADP with other key sizes since no CAVP test is available for key sizes other than 2048-bit.

2.8 RBG and Entropy

N/A for this module.

N/A for this module.

2.9 Key Generation

2.10 Key Establishment

2.11 Industry Protocols

The Module conforms to [140-3 IG] D.C References to the Support of Industry Protocols: while the module provides [56A] conformant schemes and API entry points oriented to TLS and SSH usage, the Module does not contain the full implementation of TLS or SSH. The following statements are required per IG D.C case #2:

> *No parts of the TLS protocol other than the approved cryptographic algorithms and the KDFs, have been tested by the CAVP and CMVP.*
>
> *No parts of the SSH protocol other than the approved cryptographic algorithms and the KDFs, have been tested by the CAVP and CMVP.*

2.12 Additional Information

The Module design corresponds to the Module security rules. Security rules enforced by the Module are described in the appropriate context of this document.

3 Cryptographic Module Interfaces

3.1 Ports and Interfaces

Physical Port	Logical Interface(s)	Data That Passes
N/A: Internal (call stack)	Control Input	API entry point: stack frame including non-sensitive parameters
N/A: Internal (call stack)	Control Output	API call parameters passed by reference for structures allocated by wolfCrypt
N/A: Internal (call stack)	Data Input	API call parameters passed by reference or value for cryptographic service input
N/A: Internal (call stack)	Data Output	API call parameters passed by reference for cryptographic service output
N/A: Internal (call stack)	Status Output	API return value: enumerated status resulting from call execution

Table 11: Ports and Interfaces

Table 7 defines the Module's [140-3] logical interfaces; the Module does not interact with physical ports.

4 Roles, Services, and Authentication

4.1 Authentication Methods

N/A for this module.

4.2 Roles

Name	Type	Operator Type	Authentication Methods
CO	Role	CO	

Table 12: Roles

The Module supports the Cryptographic Officer (CO) operator role, and does not support multiple concurrent operators, a maintenance role or bypass capability. The cryptographic module does not provide an authentication or identification method of its own. The CO role is implicitly identified by the service requested.

4.3 Approved Services

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Digital Signature	Generate or verify digital signatures.	Successful completion of the service (status code >= 0)	Sign: Key Struct (DS_SGK); message; Verify: signature value; flags; sizes.	Sign: Status return; Signature value. Verify: Status return;	Digital Signature Generation Digital Signature Verification	CO - DS_SGK: W,E,Z - DS_SVK: W,E,Z
Generate Key Pair	Generate asymmetric key pairs.	Successful completion of the service (status code >= 0)	FFC, ECC: curve identifier; RSA: modulous size;	Status return; Key structure (GKP_Private)	ECC Asymmetric Key-Pair Generation DSA Asymmetric Key-Pair Generation RSA Asymmetric Key-Pair Generation 133r2 5.2	CO - GKP_Private: G,R,Z - GKP_Public: G,R,Z

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
					Asymmetric Key Generation 133r2 5.1 Asymmetric Key Generation	
Key Agreement	DH key agreement primitives.	Successful completion of the service (status code >= 0)	Key structures (KAS_Private and KAS_Public); flags;	Status return; KAS_SSC;	Primitive Key Agreement	CO - KAS_Private: W,E,Z - KAS_Public: W,E,Z - KAS_SSC: G,R,Z
Key Derivation	Derive keying material from a shared secret	Successful completion of the service (status code >= 0)	KAS_SSC; flags;	Status return; KD_DKM;	TLS 1.3 Key Agreement KDF Derived Key Agreement KAS SSC Derived Key Agreement	CO - KAS_SSC: R,E,Z
Keyed Hash	Generate or verify message integrity	Successful completion of the service (status code >= 0)	KH_Key	Status return; Tag value;	Message Authentication Auth Block Cipher	CO - KH_Key: W,E
Message Digest	Generate a message digest	Successful completion of the service (status code >= 0)	Message; flags;	Status return; Hash value;	Secure Hash	CO
Random	Generate random bits using the DRBG	Successful completion of the service (status	DRBG structure (Internal State containing secret(s) C	Status return; Random Value;	DRBG	CO - Seed: W,E,Z - Internal State: G,E - Secret C: G,E

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
		code >= 0)	and V); Seed			- Secret V: G,E - Entropy Input String: W,E,Z
Self-test	Perform the designated self-test.	Successful completion of the service (status code >= 0)	Flags	Status return	Message Authentication	CO - MOD_INT: G,Z - coreKey: E
Show Status	Provide Module status	Successful completion of the service (status code >= 0)	None	Status return		CO
Symmetric cipher	Encrypt or Decrypt data, including AEAD modes (CCM, GCM)	Successful completion of the service (status code >= 0)	SC_EDK; flags;	Status return. Plaintext or ciphertext data;	Auth Block Cipher UnAuth Block Cipher Symmetric Key Generation	CO - SC_EDK: E,W
Zeroise	FreeRng_fips destroys RNG CSPs. All functions zeroise CSPs using function ForceZero (overwriting with zeros) within the function scope after use. Caller stack cleanup is the duty of	Successful completion of the service (status code >= 0)	DRBG struct (RBG State) or other structures containing SSPs	Status return		CO - DS_SGK: Z - GKP_Private: Z - KAS_Private: Z - KAS_SSC: Z - KD_DKM: Z - KH_Key: Z - Seed: Z - Internal

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
	the application. Restarting the general-purpose computer clears all CSPs in RAM.					State: Z - Secret C: Z - Secret V: Z - SC_EDK: Z - Entropy Input String: Z
Show Version	Provide Module Version	Successful completion of the service (status code >= 0)	None	Plaintext containing the module version		CO

Table 13: Approved Services

All services implemented by the Module are listed in Table 16. The calling application may use the Show status service (wolfCrypt_GetStatus_fips call) to determine the status of the Module. A return code of FIPS_MODE_NORMAL means the Module is in a state without errors; Please see Section 2.4 for more information. In addition, as per [140-3 IG] 2.4.C the module supports an implicit indicator via the successful completion of a service, module does not support non-approved services.

See the AVIAT NETWORKS FIPS 140-3 User Guide [UG] for additional information on the cryptographic services listed in this section.

Note that the caller provides the KAS_Private and KAS_Public keys for shared secret computation; the caller's exchange and assurance of PSPs with the remote participant is outside the scope of the Module.

For services Generate Key Pair, Key Agreement and Key Derivation consistent with [140-3 IG] 9.5.A, available only if the *private_key_read_enable* property is set to TRUE

4.4 Non-Approved Services

N/A for this module.

5 Software/Firmware Security

5.1 Integrity Techniques

The Module uses HMAC-SHA2-256 with a 256-bit key (HMAC Cert. #A4308) as the approved integrity technique. Before the integrity technique is executed the module performs an HMAC-SHA2-256 KAT.

5.2 Initiate on Demand

The operator can initiate the integrity test on demand by reloading the Module or by calling the API wolfCrypt_IntegrityTest_fips() at any time after power on. (See Section 10.5 “Operator Initiation of Self-Tests” later in this document for details of proper use of this API in an application).

5.3 Open-Source Parameters

While the module is not “open source” since it is only shipped under a commercial license, open source practice of source code delivery with a commercial license is standard for the module. As such the module (while not required to do so) will abide by ISO/IEC 19790:2012 B.2.5. Please see details in the AVIAT NETWORKS FIPS 140-3 User Guide [UG] for the [OE] listed on the FIPS certificate. Details will include information about compiler, compiler configuration settings and methods to compile the source code into an executable form in a FIPS validated manner. See also section 11.1 Installation, Initialization, and Startup Procedures later in this document.

6 Operational Environment

6.1 Operational Environment Type and Requirements

Type of Operational Environment: Modifiable

6.2 Configuration Settings and Restrictions

Any setting that affects the module directly while compiling the executable binary *shall* not be used. If unsure contact AVIAT NETWORKS by sending email to “TACAM at aviatnet dot com”. An AVIAT engineer will review the setting for impact on the FIPS validated sources and determine if the setting is allowed or disallowed for an approved mode of operation. NOTE: The User Guide [UG] will contain an exact list of allowed settings. CO should refer to the [UG] first before contacting AVIAT support.

6.3 Additional Information

The operational environment for the Module is modifiable.

Table 6 lists the operational environments on which the Module was tested.

Specification of the security rules, settings or restrictions to the configuration of the operational environment are covered in the [UG]. The configure script provided with the package detects the environment and sets the required flags.

There are no specific restrictions to the configuration of the operational environment unless stated in the [UG].

7 Physical Security

7.1 Mechanisms and Actions Required

N/A for this module.

7.5 EFP/EFT Information

N/A for this module.

7.6 Hardness Testing Temperature Ranges

N/A for this module.

8 Non-Invasive Security

The Module does not implement non-invasive security mechanisms.

9 Sensitive Security Parameters Management

9.1 Storage Areas

Storage Area Name	Description	Persistence Type
S1	RAM (Memory)	Dynamic

Table 14: Storage Areas

9.2 SSP Input-Output Methods

Name	From	To	Format Type	Distribution Type	Entry Type	SFI or Algorithm
IE1	EXT: Call stack (API) input parameters	INT	Plaintext	Automated	Electronic	
IE2	INT: Call stack (API) output parameters	EXT	Plaintext	Automated	Electronic	
IE3	EXT: Loaded from external entropy source	INT	Plaintext	Automated	Electronic	

Table 15: SSP Input-Output Methods

9.3 SSP Zeroization Methods

Zeroization Method	Description	Rationale	Operator Initiation
Z1	cleared immediately after use	Module does not store SSPs persistently	Zeroise
Z2	Per ISO/IEC 19790:2012 section 7.9.7, parameters used solely for self-test purposes in 7.10 need not meet zeroisation requirements	FIPS 140-3 IG 9.7.B	

Table 16: SSP Zeroization Methods

The module supports an implicit Zeroisation indicator. The implicit indicator is a successful completion of the service call.

9.4 SSPs

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
DS_SGK	Digital Signature: Signature Generation using Private Key	RSA: 2048, 3072, 4096; ECDSA: 224, 256, 384, 521; - RSA: 112, 128; ECDSA: 112, 128, 192, 256;	Private - CSP			RSA SigGen (FIPS186-4) ECDSA SigGen (FIPS186-4)
DS_SVK	Digital Signature Verification using Public Key	RSA: 1024*, 2048, 3072, 4096; ECDSA: 192*, 224, 256, 384, 521; - RSA:	Public - PSP			RSA SigVer (FIPS186-4) ECDSA SigVer (FIPS186-4)

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
		80*, 112, 128; ECDSA: 80*, 112, 128, 192, 256;				
GKP_Private	Generated Key Pair (Private)	RSA: 2048, 3072, 4096; ECDSA: 224, 256, 384, 521; - RSA: 112, 128; ECDSA: 112, 128, 192, 256;	Private - CSP	RSA Asymmetric Key-Pair Generation ECC Asymmetric Key-Pair Generation		RSA KeyGen (FIPS186-4) ECDSA KeyGen (FIPS186-4)
GKP_Public	Generated Key Pair (Public)	RSA: 2048, 3072, 4096; ECDSA: 224, 256, 384, 521; - RSA: 112, 128; ECDSA: 112, 128, 192, 256;	Public - PSP	RSA Asymmetric Key-Pair Generation ECC Asymmetric Key-Pair Generation		RSA KeyGen (FIPS186-4) ECDSA KeyGen (FIPS186-4)
KAS_Private	Key pair component provided by the local	FFC: 2048; ECC: 224,	Private - CSP	ECC Asymmetric Key-Pair Generation		KAS-FFC-SSC Sp800-

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
	participant, used for Diffie-Hellman shared secret generation.	256, 384, 521; - FFC: 112; ECC: 112, 128, 192, 256;		DSA Asymmetric Key-Pair Generation Primitive Key Agreement		56Ar3 KAS-ECC-SSC Sp800-56Ar3
KAS_Public	Key pair component provided by the local participant, used for Diffie-Hellman shared secret generation.	FFC: 2048; ECC: 224, 256, 384, 521; - FFC: 112; ECC: 112, 128, 192, 256;	Public - PSP	ECC Asymmetric Key-Pair Generation DSA Asymmetric Key-Pair Generation Primitive Key Agreement		KAS-ECC-SSC Sp800-56Ar3 KAS-FFC-SSC Sp800-56Ar3
KAS_SSC	Shared secret calculation; z output value is expected to be used by a KDF	FFC: 2048; ECC: 224, 256, 384, 521; - FFC: 112; ECC: 112, 128, 192, 256;	Shared Secret - CSP		ECC Asymmetric Key-Pair Generation DSA Asymmetric Key-Pair Generation KAS SSC Derived Key Agreement	KAS-FFC-SSC Sp800-56Ar3 KAS-ECC-SSC Sp800-56Ar3 KDF TLS KDF SSH
KD_DKM	Key Derivation derived keying material	TLS KDF v1.2 RFC 7627: 1024; TLS KDF v1.3:	Derived Key Material - CSP		TLS 1.3 Key Agreement KDF Derived Key Agreement	TLS v1.2 KDF RFC7627 TLS v1.3 KDF KDF SSH

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
		256, 384; KDF SSH: 256, 384, 512 - 256-bit				
KH_Key	Keyed Hash key	CMAC: 128, 192, 256; GMAC: 128, 192, 256; HMAC: 160, 256, 512; - CMAC: 128, 192, 256; GMAC: 128, 192, 256; HMAC: 128, 256;	Symmetric Key - CSP			AES-CMAC AES-GMAC HMAC-SHA3-512 HMAC-SHA3-384 HMAC-SHA3-256 HMAC-SHA3-224 HMAC-SHA2-512 HMAC-SHA2-384 HMAC-SHA2-256 HMAC-SHA2-224 HMAC-SHA-1
Entropy Input String	Entropy input bit string loaded from the external entropy source	256-bit - 256-bit	Entropy - CSP			Hash DRBG
Seed	DRBG Seed_materi	384-bit - 256-bit	Entropy - CSP	DRBG		Hash DRBG

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
	al consisting of entropy input string (256-bit) concatenated with the nonce (128-bit)					
Secret C	Hash DRBG Internal State Secret C	440-bits - 256-bit	Entropy - CSP	DRBG		Hash DRBG
Secret V	Hash DRBG Internal State Secret V	440-bits - 256-bit	Entropy - CSP	DRBG		Hash DRBG
Internal State	Hash DRBG Internal State (SHA-256) with secret values V and C. V is 440-bits, C is 440-bits.	880-bit - 256-bit	Entropy - CSP	DRBG		Hash DRBG
SC_EDK	AES key used for symmetric encryption (including AES authenticated encryption). Modes: CBC, CCM, CTR, ECB, GCM, OFB	128, 192 or 256 bits - 128, 192 or 256 bits	Symmetric Key - CSP			AES-CBC AES-CCM AES-CTR AES-ECB AES-GCM AES-OFB
MOD_INT	Module Integrity Value Computed at Run Time	32-bytes - 256-bit	Message Authentication - CSP	Message Authentication		HMAC-SHA2-256
coreKey	HMAC key for in-core integrity	32-bytes - 256-bit	Message Authentication - CSP			HMAC-SHA2-256

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
	check self-test					

Table 17: SSP Table 1

Name	Input - Output	Storage	Storage Duration	Zeroization	Related SSPs
DS_SGK	IE1	S1:Plaintext	While in use	Z1	
DS_SVK	IE1	S1:Plaintext	While in use	Z1	
GKP_Private	IE2	S1:Plaintext	While in use	Z1	GKP_Public:Paired With
GKP_Public	IE2	S1:Plaintext	While in use	Z1	GKP_Private:Paired With
KAS_Private	IE1	S1:Plaintext	While in use	Z1	
KAS_Public	IE2	S1:Plaintext	While in use	Z1	
KAS_SSC		S1:Plaintext	While in use	Z1	KAS_Public:Derived From KAS_Private:Derived From
KD_DKM		S1:Plaintext	While in use	Z1	
KH_Key	IE1	S1:Plaintext	While in use	Z1	
Entropy Input String	IE3	S1:Plaintext	While in use	Z1	
Seed		S1:Plaintext	While in use	Z1	
Secret C		S1:Plaintext	While in use	Z1	Seed:Derived From
Secret V		S1:Plaintext	While in use	Z1	Seed:Derived From
Internal State		S1:Plaintext	While in use	Z1	
SC_EDK	IE1	S1:Plaintext	While in use	Z1	
MOD_INT		S1:Plaintext	While in use	Z1	
coreKey		S1:Plaintext	While in use	Z2	

Table 18: SSP Table 2

* Per SP800-131Ar2 Section 3, Table 2, key sizes (1024-bit for RSA and 192-bit for ECC) are available for legacy use verification requirements when inter-oping with legacy systems. These key sizes shall not be used for signing operations.

10 Self-Tests

10.1 Pre-Operational Self-Tests

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details
HMAC-SHA2-256	hash type: SHA256, key length: 32-bytes. Please note this is the module integrity test	KAT	SW/FW Integrity	FIPS_MODE_NORMAL or FIPS_MODE_FAILED	MAC

Table 19: Pre-Operational Self-Tests

Each time the Module is powered on or loaded (equivalent to a power on) the integrity of the module is tested per ISO/IEC 19790:2012 Section 7.10.2.2. The very first step of the pre-operational self-test (POST) is to force every Conditional Algorithm Self-Test to be in the FIPS_CAST_STATE_INIT mode meaning the CAST for a given algorithm has not run since power on and the CAST must run and pass prior to operational use of the algorithm.

The integrity test uses HMAC-SHA2-256 to ensure the modules integrity therefore per AS 10.20 HMAC CAST is triggered prior to the integrity check. The HMAC CAST uses a known answer test per ISO/IEC 19790-2012 Section 7.10.3.2. The POST executes outside user control as the module is powering on or being loaded.

10.2 Conditional Self-Tests

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
AES-CBC	key length: 32-bytes	KAT	CAST	FIPS_CAST_STATE_SUCCESS or FIPS_CAST_STATE_FAILURE	Encrypt	Before first use of algorithm(s) AES-ECB, AES-CBC, AES-CTR, AES-OFB, AES-GCM, AES-

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
						GMAC, AES-CCM or AES-CMAC
AES-CBC	key length: 32-bytes	KAT	CAST	FIPS_CAST_STATE_SUCCESS or FIPS_CAST_STATE_FAILURE	Decrypt	Before first use of algorithm(s) AES-ECB, AES-CBC, AES-CTR, AES-OFB, AES-GCM, AES-GMAC, AES-CCM or AES-CMAC
AES-GCM	key length: 32-bytes	KAT	CAST	FIPS_CAST_STATE_SUCCESS or FIPS_CAST_STATE_FAILURE	Decrypt	Before first use of algorithm(s) AES-GCM or AES-GMAC
AES-GCM	key length: 32-bytes	KAT	CAST	FIPS_CAST_STATE_SUCCESS or FIPS_CAST_STATE_FAILURE	Encrypt	Before first use of algorithm(s) AES-GCM or AES-GMAC
HMAC-SHA1	hash type: SHA1; key length: 20-bytes	KAT	CAST	FIPS_CAST_STATE_SUCCESS or FIPS_CAST_STATE_FAILURE	MAC	Before first use of algorithm(s) SHA1

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
						or HMAC-SHA1
HMAC-SHA2-256	hash type: SHA256; key length: 20-bytes	KAT	CAST	FIPS_CAST_STATE_SUCCESS or FIPS_CAST_STATE_FAILURE	MAC	Before first use of algorithm(s) SHA224, SHA256, HMAC-SHA224 or HMAC-SHA256
HMAC-SHA2-512	hash type: SHA2-512, key length: 20-bytes	KAT	CAST	FIPS_CAST_STATE_SUCCESS or FIPS_CAST_STATE_FAILURE	MAC	Before first use of algorithm(s) SHA384, SHA512, HMAC-SHA384 or HMAC-SHA512
HMAC-SHA3-256	hash type: SHA3-256, key length: 64-bytes	KAT	CAST	FIPS_CAST_STATE_SUCCESS or FIPS_CAST_STATE_FAILURE	MAC	Before first use of algorithm(s) SHA3-224, SHA3-256, SHA3-384 or SHA3-512, HMAC-SHA3-224, HMAC-SHA3-256, HMAC-SHA3-384 or HMAC-

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
						SHA3-512
RSA-PKCSv1.5	hash type: SHA256; key length: 2048-bits	KAT	CAST	FIPS_CAST_STATE_SUCCESS or FIPS_CAST_STATE_FAILURE	Sign	Before first use of algorithm(s) RSA (PKCSv1.5) or RSA (PSS)
RSA-PKCSv1.5	hash type: SHA256, key length: 2048-bits	KAT	CAST	FIPS_CAST_STATE_SUCCESS or FIPS_CAST_STATE_FAILURE	Verify	Before first use of algorithm(s) RSA (PKCSv1.5) or RSA (PSS)
ECC Diffie-Hellman	hashType: SHA2-256; curve: P-256	KAT	CAST	FIPS_CAST_STATE_SUCCESS or FIPS_CAST_STATE_FAILURE	Computation Shared Secret Z	Before first use of algorithm(s) ECC for shared secret generation
FFC Diffie-Hellman	hashType: SHA2-256; keySize: 2048-bit;	KAT	CAST	FIPS_CAST_STATE_SUCCESS or FIPS_CAST_STATE_FAILURE	Computation Shared Secret Z	Before first use of algorithm(s) FFC for shared secret generation
ECDSA	curve: P256; hashType: SHA2-256	KAT	CAST	FIPS_CAST_STATE_SUCCESS or FIPS_CAST_STATE_FAILURE	Sign	Before first use of algorithm(s) ECDSA
ECDSA	curve: P256; hashType: SHA2-256	KAT	CAST	FIPS_CAST_STATE_SUCCESS or FIPS_CAST_STATE_FAILURE	Verify	Before first use of algorithm(s) ECDSA

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
TLSv1.2 KDF	HMAC-SHA2-256	KAT	CAST	FIPS_CAST_STATE_SUCCESS or FIPS_CAST_STATE_FAILURE	Derive Keying Material	Before first use of TLSv1.2 KDF
TLSv1.3 KDF	HMAC-SHA2-256	KAT	CAST	FIPS_CAST_STATE_SUCCESS or FIPS_CAST_STATE_FAILURE	Derive Keying Material	Before first use of TLSv1.3 KDF
KDF SSH	hashType: SHA2-256	KAT	CAST	FIPS_CAST_STATE_SUCCESS or FIPS_CAST_STATE_FAILURE	Derive Keying Material	Before first use of KDF SSH
RSA-PCT	key size: 2048,3072,4096	PCT	PCT	Service is successful or an error code RSA_KEY_PAIR_E	Sign/Verify	Invoked automatically during generate key pair service
ECC-PCT	curve size: 224, 256, 384, 521	PCT	PCT	Service is successful or an error code ECC_PCT_E	Sign/Verify	Invoked automatically during generate key pair service
DH-PCT	key size: 2048, 3072, 4096	PCT	PCT	Service is successful or an error code MP_CMP_E	Modulus Exponentiation	Invoked automatically during generate key pair service
DRBG	DRBG mode: SHA2-256	KAT	CAST	FIPS_CAST_STATE_SUCCESS or FIPS_CAST_STATE_FAILURE	Health-Test with sub-elements: Instantiate, Generate, Reseed	Before first use of algorithm(s) DBRG or Immediately upon registering an external entropy source

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
						with the module

Table 20: Conditional Self-Tests

Once the module is powered on and has passed the POST, calls to any cryptographic algorithm will trigger the CAST on first operational use of the algorithm. The POST and CASTS are available on demand after power on and can be executed by the cryptographic officer (CO) at any time. The CO may optionally invoke any CAST ahead of algorithm use at a more convenient time rather than letting it run automatically on first use. Regardless of the CAST running manually or automatically, once it has passed the CO may manually re-run any CAST at any time in a periodic fashion, a CAST will no longer run automatically after it has passed the first time.

10.3 Periodic Self-Test Information

Algorithm or Test	Test Method	Test Type	Period	Periodic Method
HMAC-SHA2-256	KAT	SW/FW Integrity	P2	Automatic or Manually

Table 21: Pre-Operational Periodic Information

Algorithm or Test	Test Method	Test Type	Period	Periodic Method
AES-CBC	KAT	CAST	P1	Manually
AES-CBC	KAT	CAST	P1	Manually
AES-GCM	KAT	CAST	P1	Manually
AES-GCM	KAT	CAST	P1	Manually
HMAC-SHA1	KAT	CAST	P1	Manually
HMAC-SHA2-256	KAT	CAST	P2	Automatic or Manually
HMAC-SHA2-512	KAT	CAST	P1	Manually
HMAC-SHA3-256	KAT	CAST	P1	Manually
RSA-PKCSv1.5	KAT	CAST	P1	Manually
RSA-PKCSv1.5	KAT	CAST	P1	Manually
ECC Diffie-Hellman	KAT	CAST	P1	Manually
FFC Diffie-Hellman	KAT	CAST	P1	Manually
ECDSA	KAT	CAST	P1	Manually
ECDSA	KAT	CAST	P1	Manually
TLSv1.2 KDF	KAT	CAST	P1	Manually
TLSv1.3 KDF	KAT	CAST	P1	Manually

Algorithm or Test	Test Method	Test Type	Period	Periodic Method
KDF SSH	KAT	CAST	P1	Manually
RSA-PCT	PCT	PCT	P3	Automatic
ECC-PCT	PCT	PCT	P3	Automatic
DH-PCT	PCT	PCT	P3	Automatic
DRBG	KAT	CAST	P4	Automatic or Manually

Table 22: Conditional Periodic Information

Name	Description
P1	Periodic method 1: Automatically by the module when algorithm is first invoked. CO may opt to invoke prior to first algorithm use to avoid delay at time of first operational use of an algorithm or at a later time manually.
P2	Periodic method 2: Automatically by the module during power on. CO may opt to invoke manually thereafter.
P3	Periodic method 3: Automatically during key generation service
P4	Automatically by the module upon first operational use of the DRBG algorithm. When an external entropy source is registered with the module by application level entropy callback function it is considered the first operational use of the DRBG. Does a periodic reseed every 1 million invocations, during the reseed the DRBG health test will be automatically executed.

Table 23: Periodic Method Descriptions

10.4 Error States

Name	Description	Conditions	Recovery Method	Indicator
FIPS_MODE_FAILED	Module has failed its software integrity check	HMAC-SHA2-256 CAST Failure Module Integrity Check Failure	Power Cycle	fipsModeId set to FIPS_MODE_FAILED (3)
FIPS_CAST_STATE_FAILURE	One or more algorithm(s) are no longer usable and the module mode is	AES-CBC AES-GCM	Power Cycle	One or more algorithms CAST status values set to

Name	Description	Conditions	Recovery Method	Indicator
	set to FIPS_MODE_DEGRADED (2)	HMAC-SHA1 HMAC-SHA2-256 HMAC-SHA2-512 HMAC-SHA3-256 RSA-PKCSv1.5 DRBG ECC Diffie-Hellman FFC Diffie-Hellman ECDSA TLSv1.2 KDF TLSv1.3 KDF KDF SSH		FIPS_CAST_STATE_FAILURE (3)
FIPS_MODE_DEGRADED	One or more of the CASTS have failed anytime following a successful power on and integrity check. Upon entering this mode the module will automatically run all CASTS prior to the operational use of any cryptographic algorithm.	Any CAST Failure	Power Cycle	fipsModeId set to FIPS_MODE_DEGRADED (2)
RSA_KEY_PAIR_E	RSA Pairwise Consistency Test Failure	RSA-PCT	Manual self-test service call or	RSA_KEY_PAIR_E (-262)

Name	Description	Conditions	Recovery Method	Indicator
			power cycle	
ECC_PCT_E	ECC Pairwise Consistency Test Failure	ECC-PCT	Manual self-test service call or power cycle	ECC_PCT_E (-286)
MP_CMP_E	DH Pairwise Consistency Test Failure	DH-PCT	Manual self-test service call or power cycle	MP_CMP_E (-120)

Table 24: Error States

10.5 Operator Initiation of Self-Tests

For calling applications the following is required:

1. Include the library configuration header wolfssl/options.h (or user_settings.h via wolfssl/wolfcrypt/settings.h) first.
2. After including the library configuration header, include wolfssl/wolfcrypt/fips_test.h then use the API specified below to execute a given self-test.

CO may initiate all CAST self-tests in one-shot. The API wc_RunAllCast_fips() is provided as a public API to applications using the module that have included the headers above in proper order.

CO may initiate CAST self-tests individually using the API wc_RunCast_fips(algorithm type) with any of the below “algorithm type” inputs:

- FIPS_CAST_AES_CBC
- FIPS_CAST_AES_GCM
- FIPS_CAST_HMAC_SHA1
- FIPS_CAST_HMAC_SHA2_256
- FIPS_CAST_HMAC_SHA2_512
- FIPS_CAST_HMAC_SHA3_256
- FIPS_CAST_DRBG
- FIPS_CAST_RSA_SIGN_PKCS1v15
- FIPS_CAST_ECC_CDH
- FIPS_CAST_ECC_PRIMITIVE_Z
- FIPS_CAST_DH_PRIMITIVE_Z

- FIPS_CAST_ECDSA
- FIPS_CAST_KDF_TLS12
- FIPS_CAST_KDF_TLS13
- FIPS_CAST_KDF_SSH

CO may re-run the POST at any time after power on using the public API wolfCrypt_IntegrityTest_fips(). This function always returns a value of zero regardless if the integrity check passed or failed so the CO *shall* then check the status of the module using the API wolfCrypt_GetStatus_fips(). The return value of the GetStatus API shall then be checked against the status indicators below:

- FIPS_MODE_INIT status indicator value is 0. This indicator means then integrity test has not completed and is likely running in another thread (multi-threaded)
- FIPS_MODE_NORMAL status indicator value is 1. This indicator means the integrity test passed and the module is in a state without errors
- FIPS_MODE_FAILED status indicator value is 3. This indicator means the integrity test failed and the module is unusable. The CO shall power cycle or reloaded (equivalent to power cycle) to restore the module.

11 Life-Cycle Assurance

11.1 Installation, Initialization, and Startup Procedures

The CO *shall* use the provided AVIAT NETWORKS FIPS 140-3 User Guide hereafter referred to as [UG]. A common name for this document is also the Cryptographic Officer Guidance Manual [COGM]. [UG] and [COGM] are one and the same for this module and include all administrative guidance. The [UG] will have a section specific to each Operational Environment [OE] that appears on the modules FIPS certificate and/or in Table 6: Tested Operational Environments - Software, Firmware, Hybrid. The instructions provided in the [UG] *shall* be followed or the module will never have been properly initialized and built and therefore non-compliant. To create the compliant module, as per this Security Policy, the configuration steps *shall* be followed.

- [UG] will include library configuration settings that are: *required*, *allowed* and *not allowed*.
    - For any setting that is not specifically covered the CO *shall* contact AVIAT by emailing “TACAM at aviatnet dot com” for clarification about that settings impact on compiling the compliant module.

- [UG] will include details about the toolchain, compiler, compiler configuration settings and methods to compile the source code into an executable form.
    - While the module is not “open source” since it is only shipped under a commercial license, open source practice of source code delivery with a commercial license is standard for the module. As such the module (while not required to) will abide by ISO/IEC 19790:2012 B.2.5 “If the module is open source, specify the compilers and control parameters required to compile the

code into an executable format” even though the module is not claiming “open source”.

The following initialization instructions apply to all use-cases for the module generically by a consuming application. [OE] specific details will be covered in the [UG].

- When planning on using the module the CO shall first include the library settings headers so the application knows how the library was configured.
    - On Unix or Linux like systems where auto-tools were used to configure the library (./configure && make) the CO *shall* include wolfssl/options.h as the very first header.
    - When working with IDEs or Makefile setups the CO *shall* include wolfssl/wolfcrypt/settings.h as the very first header and ensure that the define WOLFSSL_USER_SETTINGS is set globally at the project level. No other wolfSSL specific build options should be set globally, all configurations will be managed by a custom user_settings.h header that is included anytime WOLFSSL_USER_SETTINGS is defined globally.
        - Once the library configuration settings have been included only then shall the CO include other wolfSSL headers as needed, any other headers shall always come after the configuration settings header.

- Upon entry into main() of an application the vendor recommends that the CO first register a fips callback. The fips callback will trigger anytime the module is in an unusable state. A sample of such a callback is provided below.

```
static void myFipsCb(int ok, int err, const char* hash)
{
    printf("in my Fips callback, ok = %d, err = %d\n", ok, err);
    printf("message = %s\n", wc_GetErrorString(err));
    printf("hash = %s\n", hash);

    if (err == IN_CORE_FIPS_E) {
        printf("In core integrity hash check failure,"
               "copy above hash\n");
        printf("into verifyCore[] in fips_test.c and rebuild\n");
    }
}
```

Figure 2: Code Sample A

    - If using a FIPS callback the CO *will* register the FIPS callback by passing the function pointer of the FIPS callback function to the following API like so: wolfCrypt_SetCb_fips(myFipsCb);
- Prior to operational use of the module the CO *shall* register an entropy callback to load entropy into the module from an external entropy source. A portable callback is available but must be registered by the application on startup since the entropy source is external to the module. To register the portable callback provided with the module the application will call “ret = wc_SetSeed_Cb(wc_GenerateSeed);” where “ret” is an integer to capture the status return of the call and should be checked against the value 0 for success or < 0 for failure. A successful register of any entropy callback function is considered the first operational use of the module outside of pre-operational tests and the DBRG CAST will run during registration of the callback.

- When working with a private key the application *must* programmatically unlock access to private key material with the API: wolfCrypt_SetPrivateKeyReadEnable_fips(true/false, key-type).
    - To unlock key access pass true (1) as the first input. CO *shall* pass WC_KEYTYPE_ALL as the second input parameter.
    - Once done working with the private key CO *shall* lock access to private key materials before resuming operations by calling the same API with input false (0) as the first parameter and WC_KEYTYPE_ALL as the second input parameter.

11.2 Administrator Guidance

The CO *shall* use the provided AVIAT NETWORKS FIPS 140-3 User Guide [UG].

11.3 Non-Administrator Guidance

The Module supports the Cryptographic Officer (CO) operator role and does not support non-administrators or non-administrative roles.

11.7 Additional Information

Please defer to AVIAT NETWORKS FIPS 140-3 User Guide [UG].

12 Mitigation of Other Attacks

The module does not claim mitigation of other attacks.