Suite B Cryptographic Module

FIPS 140-2 Non-Proprietary Security Policy

Revision: 1.2

Prepared by: KeyW Corporation
7880 Milestone Parkway
Suite 100
Hanover, MD 21076
410-904-5200 *Phone*
410-799-3479 *Fax*

Contents

Tables and Figures

Revision History

Revision	Date	Author	Changes
1.2	February 9, 2017	A. Seaman D. Mackie C. Constantinescu D. Brown	Revised: Section 2.1.1, Section 2.1.1.1, Figure 1, and Table 9
1.1	January 6, 2017	A. Seaman D. Mackie C. Constantinescu D. Brown	Added Security Functions
1.0	July 11, 2014	R. Glenn D. Mackie C. Constantinescu D. Wolff E. Hufford	Initial Release

Acronyms

AAD	Additional Authentication Data
AES	Advanced Encryption Standard
AESAVS	Advanced Encryption Standard Algorithm Validation Suite
ANS	American National Standard
API	Application Programming Interface
CAVP	Cryptographic Algorithm Validation Program
CBC	Cipher Block Chaining
CDH	Cofactor Diffie-Hellman
CM	Cryptographic Module
CMAC	CBC Message Authentication Code
CMACVS	CBC Message Authentication Code Validation System
CSP	Critical Security Parameters
CT	Ciphertext
CTR	Counter
CVL	Component Validation List
DAR	Data At Rest
DEP	Default Entry Point
DIT	Data In Transit
DKM	Derived Keying Material
DLL	Dynamic Link Library
DOC	Department of Commerce
DPI	Double-Pipeline Iteration
DPK	Data Protection Key
DRBG	Deterministic Random Bit Generator
DUNS	Data Unit Sequence Number
EC	Elliptic Curve
ECB	Electronic CodeBook
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
ECDSA2VS	Elliptic Curve Digital Signature Algorithm Validation System
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
FB	Feedback
FFC	Finite Field Cryptography
FIPS	Federal Information Processing Standard
FSM	Finite State Model
GCM	Galois/Counter Mode
GCMVS	Galois/Counter Mode Validation System
GMAC	Galois Message Authentication Code
GPC	General-purpose Computer
HMAC	Keyed-hash Message Authentication Code
HMACVS	Keyed-hash Message Authentication Code Validation System
I/O	Input/Output
IAW	In Accordance With
IETF	Internet Engineering Task Force
IV	Initialization Vector
KAS	Key Agreement Scheme
KASVS	Key Agreement Schemes Validation System
KAT	Known Answer Test
KBKDF	Key-Based Key Derivation Function
KBKDFVS	Key-Based Key Derivation Function Validation System
KC	Key Confirmation
KDF	Key Derivation Function

KW	Key Wrap
KWP	Key Wrap With Padding
KWVS	Key Wrap Validation System
LED	Light Emitting Diode
MAC	Message Authentication Code
MK	Master Key
MQV	Menezes-Qu-Vanstone
NIST	National Institute of Standards and Technology
OS	Operating System
PBKDF	Password-Based Key Derivation Function
PKV	Public Key Validation
POST	Power-On Self-Test
PRF	Pseudo-Random Function
PT	Plaintext
RAM	Random Access Memory
RBG	Random Bit Generator
RFC	Request For Comments
S/MIME	Secure/Multipurpose Internet Mail Extensions
SHA	Secure Hash Algorithm
SHAVS	Secure Hash Algorithm Validation System
SHS	Secure Hash Standard
SO	Shared Object
SP	Special Publication
SSL	Secure Sockets Layer
TLS	Transport Layer Security
USB	Universal Serial Bus
USSOCOM	United States Special Operations Command
VS	Validation Specification
XTS	**X**EX **T**weakable Block Cipher with Ciphertext **S**tealing
XTSVS	XTS Validation System

1. Introduction

1.1. Identification

The following information identifies this document:

- Title: Suite B Cryptographic Module FIPS 140-2 Non-Proprietary Security Policy
- Version: 1.2

1.2. Overview

KeyW Corporation, in coordination with the United States Special Operations Command (USSOCOM), has developed a Federal Information Processing Standard (FIPS) 140-2 Level 1 validated, standards-based Suite B Cryptographic Module that provides an advanced layer of encrypted Data In Transit (DIT) communications and Data At Rest (DAR) encryption via an Application Programming Interface (API).

The Suite B Cryptographic Module, hereafter collectively referred to as the Module, operates as one of several layers of platform encryption. The platform encryption can be invoked automatically when the Module is initialized, providing an additional layer of encryption and obfuscation above the Module. Additional encryption at the application layer can be added by enabling S/MIME encryption on emails, content protection encryption on shared data, and SSL/TLS encryption on web traffic.

1.3. FIPS 140-2 Security Levels

The Module meets the overall requirements applicable to Level 1 security for FIPS 140-2 as shown in the table below:

#	FIPS 140-2 Section	Level
2.1	Cryptographic Module Specification	1
2.2	Cryptographic Module Ports and Interfaces	1
2.3	Roles, Services, and Authentication	1
2.4	Finite State Model	1
2.5	Physical Security	N/A
2.6	Operational Environment	1
2.7	Cryptographic Key Management	1
2.8	EMI/EMC	1
2.9	Self-Tests	1
2.10	Design Assurance	1
2.11	Mitigation of Other Attacks	N/A
	Overall Level	1

Table 1 – Summary of Achieved FIPS 140-2 Security Levels

2. Suite B Cryptographic Module

The Module meets the requirements of the FIPS 140-2 Security Level 1 specification and provides the following cryptographic services:

- Data encryption and decryption
- Key encryption and decryption
- Message digest and authentication code generation
- Digital signature generation and verification
- Elliptic curve key agreement
- Key derivation

2.1. Cryptographic Module Specification

2.1.1. Security Functions

The Module is implemented entirely in software and contains the following FIPS-approved and FIPS non-approved, but allowed security functions:

Algorithm	Use	Specification		Mode / Key Size	CAVP Specification	CAVP Certificate
AES	Block Cipher	FIPS 197, Nov 2001 (Ref. [1])	NIST SP 800-38A, Dec 2001 (Ref. [2])	ECB-128	AESAVS, Nov 2002 (Ref. [16])	#3328
				ECB-192		
				ECB-256		
				CBC-128		#4312
				CBC-192		
				CBC-256		
			NIST SP 800-38B, May 2005 (Ref. [3])	CMAC-128	CMACVS, Aug 2011 (Ref. [17])	#4312
				CMAC-192		
				CMAC-256		
			NIST SP 800-38D, Nov 2007 (Ref. [4])	GCM-128 GMAC-128	GCMVS, Aug 2012 (Ref. [18])	#3328
				GCM-192 GMAC-192		
				GCM-256 GMAC-256		
			NIST SP 800-38E, Jan 2010 (Ref. [5])	XTS-128	XTSVS, Sep 2013 (Ref. [19])	#3328
				XTS-256		
	Key Storage		NIST SP 800-38F, Dec 2012 (Ref. [6])	KW-128	KWVS, Jun 2014 (Ref. [20])	#3328
				KW-192		
				KW-256		
			IETF RFC 5649, Aug 2009 (Ref. [7])	KWP-128		#3328
				KWP-192		
				KWP-256		

Algorithm	Use	Specification	Mode / Key Size		CAVP Specification	CAVP Certificate
SHA	Secure Hashing	FIPS 180-4, Aug 2015 (Reference [8])	SHA-1 (SHA-160)		SHAVS, May 2014 (Ref. [21])	#2761
			SHA-224			
			SHA-256			
			SHA-384			
			SHA-512			
			SHA-512/224			
			SHA-512/256			
CMAC	Message Authentication	NIST SP 800-38B, May 2005 (Ref. [3])	AES-128		CMACVS, Aug 2011 (Ref. [17])	#4312
			AES-192			
			AES-256			
GMAC		NIST SP 800-38D, Nov 2007 (Ref. [4])	AES-128		GCMVS, Aug 2012 (Ref. [18])	#3328
			AES-192			
			AES-256			
HMAC		FIPS 198-1, July 2008 (Reference [9])	SHA-1 (SHA-160)		HMACVS, July 2012 (Ref. [22])	#2119
			SHA-224			
			SHA-256			
			SHA-384			
			SHA-512			
			SHA-512/224			
			SHA-512/256			
ECDSA	Digital Signature Per NIST SP 800-131A, P-192 and SHA-1 are no longer considered secure and shall not be used to generate digital signatures (Ref. [14]).	FIPS 186-4, July 2013 (Reference [12])	P-192	SHA-1 (SHA-160)	ECDSA2VS, Mar 2014 (Ref. [24])	#657
				SHA-224		
				SHA-256		
				SHA-384		
				SHA-512		
				SHA-512/224		
				SHA-512/256		
			P-224	SHA-1 (SHA-160)		
				SHA-224		
				SHA-256		
				SHA-384		
				SHA-512		
				SHA-512/224		
				SHA-512/256		
			P-256	SHA-1 (SHA-160)		
				SHA-224		
				SHA-256		
				SHA-384		
				SHA-512		
				SHA-512/224		

Algorithm	Use	Specification	Mode / Key Size		CAVP Specification	CAVP Certificate
				SHA-512/256		
			P-384	SHA-1 (SHA-160)		
				SHA-224		
				SHA-256		
				SHA-384		
				SHA-512		
				SHA-512/224		
				SHA-512/256		
			P-521	SHA-1 (SHA-160)		
				SHA-224		
				SHA-256		
				SHA-384		
				SHA-512		
				SHA-512/224		
				SHA-512/256		
ECC KAS	Key Establishment	NIST SP 800-56A Rev 2, May 2013 (Reference [15])	FullUnified KC EB P-224, SHA-224		KASVS, May 2014 (Ref. [25])	#55
			FullUnified KC EC P-256, SHA-256			
			FullUnified KC ED P-384, SHA-384			
			FullUnified KC EE P-521, SHA-512			
			FullMQV KC EB P-224, SHA-224			
			FullMQV KC EC P-256, SHA-256			
			FullMQV KC ED P-384, SHA-384			
			FullMQV KC EE P-521, SHA-512			
ECC CDH Primitive	Shared Secret Establishment	NIST SP 800-56A Rev 2, May 2013 (Reference [15], Section 5.7.1.2)	P-224		KASVS, May 2014 (Ref. [25])	#484 (CVL)
			P-256			
			P-384			
			P-521			
KBKDF-CMAC	Key Derivation	NIST SP 800-108, Oct 2009 (Reference [10])	CTR	CMAC-AES-128	KBKDFVS, Jan 2016 (Ref. [23])	#116
				CMAC-AES-192		
				CMAC-AES-256		

Algorithm	Use	Specification	Mode / Key Size		CAVP Specification	CAVP Certificate
			FB	CMAC-AES-128		
				CMAC-AES-192		
				CMAC-AES-256		
			DPI	CMAC-AES-128		
				CMAC-AES-192		
				CMAC-AES-256		
KBKDF-HMAC	Key Derivation	NIST SP 800-108, Oct 2009 (Reference [10])	CTR	HMAC-SHA-1 (SHA-160)	KBKDFVS, Jan 2016 (Ref. [23])	#116
				HMAC-SHA-224		
				HMAC-SHA-256		
				HMAC-SHA-384		
				HMAC-SHA-512		
			FB	HMAC-SHA-1 (SHA-160)		
				HMAC-SHA-224		
				HMAC-SHA-256		
				HMAC-SHA-384		
				HMAC-SHA-512		
			DPI	HMAC-SHA-1 (SHA-160)		
				HMAC-SHA-224		
				HMAC-SHA-256		
				HMAC-SHA-384		
				HMAC-SHA-512		
PBKDF	Key Derivation	NIST SP 800-132, Dec 2010 (Reference [11])	HMAC-SHA-1 (SHA-160)		VS not yet available as of Jan. 2017	Vendor-Affirmed
			HMAC-SHA-224			

Algorithm	Use	Specification	Mode / Key Size	CAVP Specification	CAVP Certificate
		See Section 2.1.1.1.	HMAC-SHA-256		
			HMAC-SHA-384		
			HMAC-SHA-512		

Table 2 – FIPS-Approved and Vendor-Affirmed Security Functions

Algorithm	Use	Specification	Mode / Key Size	CAVP Specification	CAVP Certificate
N/A	N/A	N/A	N/A	N/A	N/A

Table 3 – FIPS Non-Approved but Allowed Security Functions

2.1.1.1. NIST SP 800-132 Password-Based Key Derivation Function (PBKDF)

Per NIST SP 800-132, Recommendation for Password-Based Key Derivation, December 2010 (Reference [11]), the calling application is responsible for selecting which option is used to derive the Data Protection Key (DPK) from the Master Key and shall only use keys derived from passwords in storage applications. The Module API restricts the calling application to select a password/passphrase that is at least 10 characters long in accordance with the guidelines in NIST SP 800-63-2, Electronic Authentication Guideline, August 2013 (Reference [26]) and NIST SP 800-118, Guide to Enterprise Password Management (Draft), April 2009 (Reference [27]). Acceptable values of other parameters used in key derivation are detailed below.

PROTOTYPE:
```
t_STATUS PBKDF(U8 *MK, U32 MKbytes, const U8 *Pswd, U32 Pbytes,
               const U8 *Salt, U32 Sbytes, U32 Icount);
```

ARGUMENTS:
`MK` = pointer to a byte string representing the output (derived) master key
`MKbytes` = length of derived master key, in bytes
`Pswd` = input password, a byte string
`Pbytes` = password length (at least 10 bytes)
`Salt` = input diversification value, a byte string
`Sbytes` = `Salt` length (at least 16 bytes)
`Icount` = a large iteration count (determines how many HMAC iterations are used to generate one block of the `MK`)

RETURNS:
`SUCCESS` if all input parameters are valid
`FAILURE` otherwise

LIMITATIONS:
```
MKbytes >= 14
Pbytes  >= 10
Sbytes  >= 16
Icount  >= 1000
```
The Counter value should fit into one byte (i.e. `MKbytes` / `DigestLenB < 256`)

DESCRIPTION:
Implements the Password-Based Key Derivation Function (PBKDF), IAW NIST SP 800-132 (Reference [11]). An appropriate SHA environment (SHA-1, SHA-224, SHA-256, SHA-384 or SHA-512) must be

selected in advance using `SHA_TypeSelect()`. There is neither a Validation System in place, nor sample test vectors published by CAVP for the PBKDF algorithm, as of January 2017.

2.1.2. Modes of Operation

The Module must be installed on the FIPS 140-2 certified operational environment listed in Section 2.6 manually, and once installed it runs all algorithms in FIPS-approved mode since it is explicitly compiled to only run in FIPS-approved mode. There are no algorithms or "expanded" cryptographic modes within the Module that are not FIPS-approved as listed in Table 2 when calling security functions in the Module API.

The operational environment on which the Module runs shall be configured for FIPS mode when using a FIPS-approved platform-provided Deterministic Random Bit Generator (DRBG) in the following ways:

- ***Windows Server OS***: Enable the FIPS compliant algorithms mode via the Local Security Policy to guarantee the Module generates FIPS-validated random bytes.
- ***BlackBerry OS***: The Module confines its method calls to only those that have been FIPS-approved to guarantee generating FIPS-validated random bytes.

2.1.3. Cryptographic Boundary

The physical boundary of the Module is the physical boundary of the operational environment hardware device that executes the Module as shown in the following figure. The following figure depicts a FIPS-approved DRBG that is provided by the operational environment cryptographic Module listed in Section 2.6 and therefore the Module is bound to either the Windows Server OS cryptographic Module or BlackBerry OS cryptographic Module.

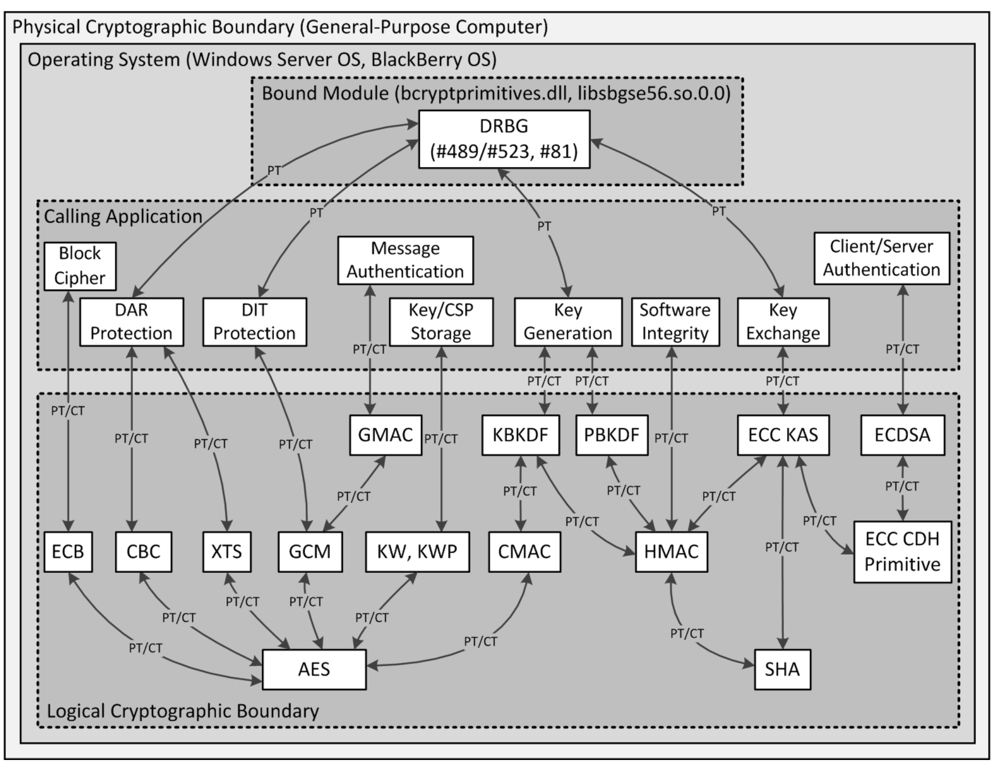

Figure 1 – Module Cryptographic Boundary

2.1.4. Determining Module Version

The operator may determine the version of the Module by performing the following steps:

Dynamic Link Library (DLL) Module Version

1. On Windows, right-click the `KEYWcryptoModule.dll` file and select view Properties
2. Select Details tab
3. The File version property displays the KEYWcryptoModule version as v3.0.0.0

Shared Object (SO) Module Version

1. On BlackBerry, run the following console command:

```
objdump -p libKEYWcryptoModule.so.3 | grep SONAME
```

2. The console displays the KEYWcryptoModule version as v3

2.2. Cryptographic Module Ports and Interfaces

The Module ports correspond to the physical ports of the operational environment hardware device that executes the Module:

- USB devices [keyboard and mouse]
- Video devices [monitors, screens, camera, and LED]
- Optical drives
- Audio devices [speakers, headset, and microphone]
- Network devices [Ethernet and Wireless adapters]
- Battery and power adapter

The Module interfaces correspond to the Module API, which do not interface across any of the physical ports of the operational environment. The following table describes the Module logical interfaces.

FIPS 140-2 Interface	Logical Interface
Data Input	Input parameters of Module constructors and function calls.
Data Output	Output parameters of Module function calls and return values.
Control Input	Module function calls.
Status Output	Return codes of Module function calls.

Table 4 – Module Logical Interfaces

2.3. Roles, Services, and Authentication

2.3.1. Roles

The Module supports a Cryptographic Officer and User role. The Module does not support a maintenance role. The Module does not support multiple or concurrent operators and is intended for use by a single operator, thus it always operates in a single-user mode of operation.

2.3.2. Services

The services described in the following tables are available to the operator roles:

Cryptographic Officer Role			
Service	**Description**	**Input/Output**	**Return**
Load Module	Performs Module initialization implicitly by the operational environment.	[in]: DLL/SO binary path [out]: VOID	Pass/Fail
Power-On Self-Test (POST)	Performs software integrity and cryptographic self-tests implicitly upon Module load.	[in]: DLL/SO binary path, DLL/SO checksum path [out]: VOID	Pass/Fail
Zeroize	Performs HMAC Integrity Checksum and Key zeroization implicitly after Module POST pass/fail. The HMAC Integrity Checksum and Key may also be zeroized by power-cycling the operational environment and reloading the Module.	[in]: HMAC Integrity Checksum, HMAC Integrity Check Key [out]: VOID	VOID
Unload Module	Performs Module destruction implicitly by the operational environment.	[in]: VOID [out]: VOID	VOID

Table 5 – Module Services for Cryptographic Officer Role

User Role				
Service		**Description**	**Input/Output**	**Return**
Run Self Tests		Performs cryptographic self-tests for the Module.	[in]: VOID [out]: VOID	Pass/Fail
CM	Show Title	Gets title info for the Module.	[in]: VOID [out]: VOID	Title Info
	Version Info	Gets version info for the Module.	[in]: VOID [out]: VOID	Version Info
	Self Tests Duration	Get cryptographic self-tests duration for the Module.	[in]: VOID [out]: VOID	Duration
AES	Construct	Constructs an AES object.	[in]: AES bit mode, AES key [out]: VOID	AES object
	Check Encrypt / Decrypt Tables	Verifies integrity of encryption/decryption tables.	[in]: VOID [out]: VOID	Pass/Fail
	ReKey	Rekeys an AES object with alternate AES key.	[in]: AES bit mode, AES key [out]: VOID	Pass/Fail
	ECB Encrypt	Encrypts PT data.	[in]: PT buffer, PT block length [out]: CT buffer	VOID
	ECB Decrypt	Decrypts CT data.	[in]: CT buffer, PT block length [out]: PT buffer	VOID
	CBC Encrypt	Encrypts PT data.	[in]: PT buffer, IV, PT block	VOID

User Role				
Service		**Description**	**Input/Output**	**Return**
			length [out]: CT buffer	
	CBC Decrypt	Decrypts CT data.	[in]: CT buffer, IV, PT block length [out]: PT buffer	VOID
	CMAC Generate	Generates a Message Authentication Code (MAC).	[in]: PT data, PT length [out]: CMAC buffer, CMAC length	VOID
	Key Wrap Encrypt	Encrypts PT keys.	[in]: PT key buffer, PT length, Inverse cipher flag [out]: CT key buffer	VOID
	Key Wrap Decrypt	Decrypts CT keys.	[in]: CT key buffer, CT length, Inverse cipher flag [out]: PT key buffer	Pass/Fail
	KDF CTR/FB/DPI	Generates a derived key.	[in]: Label/IV, Label length, Context, Context length, Counter length, Counter location [out]: Derived key, Derived key length	Pass/Fail
	Destruct	Zeroizes AES key.	[in]: VOID [out]: VOID	VOID
GCM	Construct	Constructs a GCM object.	[in]: AES bit mode, AES key [out]: VOID	GCM object
	ReKey	Rekeys a GCM object with alternate AES key.	[in]: AES bit mode, AES key [out]: VOID	Pass/Fail
	Encrypt	Encrypts PT data.	[in]: Tag length, IV, IV length, PT buffer, PT length, AAD, AAD length [out]: CT buffer, Tag	Pass/Fail
	Decrypt	Decrypts CT data.	[in]: Tag, Tag length, IV, IV length, CT buffer, CT length, AAD, AAD length [out]: PT buffer	Pass/Fail
	GMAC Encrypt	Generates a Message Authentication Code (MAC).	[in]: Tag length, IV, IV length, AAD, AAD length [out]: Tag	Pass/Fail
	GMAC Decrypt	Validates a Message Authentication Code (MAC).	[in]: Tag, Tag length, IV, IV length, AAD, AAD length [out]: VOID	Pass/Fail
	GCM Destruct	Zeroizes AES key and hash key table.	[in]: VOID [out]: VOID	VOID
XTS	Construct	Constructs an XTS object.	[in]: AES bit mode, ECB key, Tweak key, DUNS or Tweak value	XTS object

User Role				
Service		**Description**	**Input/Output**	**Return**
			[out]: VOID	
	ReKey	Rekeys an XTS object with alternate AES key.	[in]: AES bit mode, ECB key, Tweak key, DUNS or Tweak value [out]: VOID	Pass/Fail
	Encrypt	Encrypts PT data.	[in]: AES bit mode, PT buffer, Sector bit length, ECB key, Tweak key, DUNS or Tweak value [out]: CT buffer	Pass/Fail
	Decrypt	Decrypts CT data.	[in]: AES bit mode, CT buffer, Sector bit length, ECB key, Tweak key, DUNS or Tweak value [out]: PT buffer	Pass/Fail
	Destruct	Zeroizes AES key and tweak value.	[in]: VOID [out]: VOID	VOID
ECC	Construct	Constructs an ECC object.	[in]: EC type, SHA type [out]: VOID	ECC object
	Type Select	Changes the EC and SHA types.	[in]: EC type, SHA type [out]: VOID	Pass/Fail
	Check Params	Verifies EC parameters.	[in]: VOID [out]: VOID	Pass/Fail
	Is Point Affine	Determines if point is an affine coordinate.	[in]: EC Affine Point [out]: VOID	Pass/Fail
	Is Point Valid	Determines if point has correct order.	[in]: EC Affine Point [out]: VOID	Pass/Fail
	Projectify	Converts affine point to projective point.	[in]: EC Affine Point [out]: EC Projective Point	VOID
	Affinify	Converts projective point to affine point.	[in]: EC Projective Point [out]: EC Affine Point	Pass/Fail
	Compress	Converts affine point to compressed point.	[in]: EC Affine Point [out]: EC Compressed Point	VOID
	Decompress	Converts compressed point to affine point.	[in]: EC Compressed Point [out]: EC Affine Point	Pass/Fail
	Double Affine	Doubles an affine point.	[in]: EC Affine Point [out]: EC Affine Point	VOID
	Double Projective	Doubles a projective point.	[in]: EC Projective Point [out]: EC Projective Point	VOID
	Double Projective	Doubles a projective point in-place.	[inout]: EC Projective Point	VOID
	Add Affine	Adds affine points.	[in]: EC Affine Point, EC Affine Point [out]: EC Affine Point	VOID
	Add	Adds projective points.	[in]: EC Projective Point, EC	VOID

User Role				
Service		**Description**	**Input/Output**	**Return**
	Projective		Projective Point [out]: EC Projective Point	
	Multiply	Multiplies affine point by a scalar.	[in]: Scalar, EC Affine Point [out]: EC Affine Point	Pass/Fail
	Multiply Base	Multiplies EC Base Point by a scalar.	[in]: Scalar [out]: EC Affine Point	Pass/Fail
	Double Multiply	Multiplies two affine points by two scalars.	[in]: Scalar, EC Affine Point, Scalar, EC Affine Point [out]: EC Affine Point	Pass/Fail
	ECDSA Public Key Gen	Computes the public ECDSA key.	[in]: Private Key [out]: EC Public Affine Point	Pass/Fail
	ECDSA Signature Gen	Computes the ECDSA signature.	[in]: Message, Message length, Private Key, Ephemeral Key [out]: R component, S component	Pass/Fail
	ECDSA Signature Check	Verifies the ECDSA signature.	[in]: Message, Message length, R component, S component, EC Public Affine Point [out]: VOID	Pass/Fail
	ECDSA Signature Check Private	Verifies the ECDSA signature.	[in]: Message, Message length, R component, S component, Private Key [out]: VOID	Pass/Fail
	Destruct	Zeroizes ECC buffers.	[in]: VOID [out]: VOID	VOID
FFC	Construct	Constructs a FFC object.	[in]: VOID [out]: VOID	FFC Object
	Ext Dec 2 Hex	Converts an extended precision ("big") number from decimal to binary (hexadecimal).	[in]: Decimal string buffer [out]: Word buffer, Word buffer length	Pass/Fail
	Ext Hex 2 Dec	Converts an extended precision ("big") number from binary (hexadecimal) to decimal.	[in]: Word buffer, Word buffer length [out]: Decimal string buffer	VOID
	Ext Compare	Compares word buffers.	[in]: Buffer A, Buffer B, Buffer A/B length [out]: VOID	1: a == b 2: A > B 4: A < B
	Ext Mod	Reduces the a-operand modulo the n-operand.	[in]: a-operand, a length, n-operand, n length [out]: x-operand	VOID
	Ext Add	Multi-precision Add routine	[in]: a-operand, b-operand,	Final carry bit

User Role				
Service		**Description**	**Input/Output**	**Return**
		for unsigned integers.	a/b/x length [out]: x-operand	
	Ext Add	Multi-precision Add routine for unsigned integers.	[in]: b-operand, b/x length [inout]: x-operand	Final carry bit
	Ext Subtract	Multi-precision Subtract routine for unsigned integers.	[in]: a-operand, b-operand, a/b/x length [out]: x-operand	Final borrow bit
	Ext Subtract	Multi-precision Subtract routine for unsigned integers.	[in]: b-operand, b/x length [inout]: x-operand	Final borrow bit
	Ext Add Immed	Multi-precision Add routine of a single-precision, signed integer to a multi-precision unsigned integer.	[in]: b-operand, b/x length [inout]: x-operand	Final carry
	Ext Mod Add	Multi-precision modular Add routine for unsigned integers.	[in]: a-operand, b-operand, n-operand, a/b/n/x length [out]: x-operand	VOID
	Ext Mod Add	Multi-precision modular Add routine for unsigned integers.	[in]: b-operand, n-operand, b/n/x length [inout]: x-operand	VOID
	Ext Mod Subtract	Multi-precision modular Subtract routine for unsigned integers.	[in]: a-operand, b-operand, n-operand, a/b/n/x length [out]: x-operand	VOID
	Ext Mod Subtract	Multi-precision modular Subtract routine for unsigned integers.	[in]: b-operand, n-operand, b/n/x length [inout]: x-operand	VOID
	Ext Mod Add Immed	Modular Add routine of a single-precision, signed integer to a multi-precision unsigned integer.	[in]: b-operand, n-operand, b/n/x length [inout]: x-operand	VOID
	Ext Shift Left	Multi-precision 1-bit Left Shift routine for unsigned integers.	[in]: a-operand, Carry bit, a/x length [inout]: x-operand	Final carry
	Ext Shift Left	Multi-precision 1-bit Left Shift routine for unsigned integers.	[in]: x length [inout]: x-operand	Final carry
	Ext Mod Shift Left	Performs a modular addition of a long number to itself.	[in]: a-operand, n-operand, a/n/x length [out]: x-operand	VOID
	Ext Mod Shift Left	Performs a modular addition of a long number to itself.	[in]: n-operand, n/x length [inout]: x-operand	VOID
	Ext Shift Right	Multi-precision 1-bit Right Shift routine for unsigned integers.	[in]: a-operand, a/x length [out]: x-operand	VOID
	Ext Shift Right	Multi-precision 1-bit Right Shift routine for unsigned integers.	[in]: x length [inout]: x-operand	VOID

User Role				
Service		**Description**	**Input/Output**	**Return**
	Ext Mod Shift Right	Multi-precision modular divide-by-2 routine for unsigned integers.	[in]: n-operand, n/x length [inout]: x-operand	VOID
	Ext Shift Var	Multi-precision, multi-bit Left or Right Shift routine for unsigned integers.	[in]: a-operand, signed shift count, a/x length [out]: x-operand	VOID
	Ext Shift Var	Multi-precision, multi-bit Left or Right Shift routine for unsigned integers.	[in]: signed shift count, x length [inout]: x-operand	VOID
	Ext Bin Mod Inverse	Performs modular inversion 1/a with respect to a modulus n (usually a prime number) in multiple precision arithmetic.	[in]: a-operand, n-operand, a/n length [out]: a-inverse-result	VOID
	Ext Bin Mod Divide	Performs modular division b/a with respect to a modulus n (usually a prime number) in multiple precision arithmetic.	[in]: b-operand, a-operand, n-operand, b/a/n length [out]: ba-dividend-result	VOID
	Ext Bin Mod Inverse v2	Performs modular inversion 1/a with respect to a modulus n (usually a prime number) in multiple precision arithmetic.	[in]: a-operand, n-operand, a/n length [out]: a-inverse-result	VOID
	Ext Multiply	Multi-precision multiplication routine for unsigned integers of the same size.	[in]: a-operand, b-operand, a/b/x length [out]: x-operand	VOID
	Ext Multiply	Multi-precision multiplication routine for unsigned integers of different sizes.	[in]: a-operand, a length, b-operand, b length [out]: x-operand	VOID
	Ext Mod Multiply	Multi-precision modular Multiply routine for unsigned integers.	[in]: a-operand, b-operand, n-operand, a/b/n/x length [out]: x-operand	VOID
	Ext Square	Multi-precision squaring routine for unsigned integers.	[in]: a-operand, a length [out]: x-operand	VOID
	Ext Mod Square	Multi-precision modular squaring routine for unsigned integers.	[in]: a-operand, n-operand, a/n/x length [out]: x-operand	VOID
	Ext Divide	Multi-precision division routine for unsigned integers.	[in]: a-operand, a length, n-operand, n length [out]: q-operand, r-operand	VOID
	Ext Mod Inverse	Performs modular inversion 1/a with respect to a modulus n (usually a prime number) in multiple precision	[in]: a-operand, n-operand, a/n length [out]: a-inverse-result	VOID

User Role				
Service		**Description**	**Input/Output**	**Return**
		arithmetic.		
	Ext Mod Divide	Performs modular division b/a with respect to a modulus n (usually a prime number) in multiple precision arithmetic.	[in]: b-operand, a-operand, n-operand, b/a/n length [out]: ba-dividend-result	VOID
	Ext Sqrt	Multi-precision square-root routine for unsigned integers.	[in]: a-operand, a length [out]: sqrt-result	Pass/Fail
	Ext Sqrt v0	Multi-precision square-root routine for unsigned integers.	[in]: a-operand, a length [out]: sqrt-result	Pass/Fail
	Ext Sqrt v1	Multi-precision square-root routine for unsigned integers.	[in]: a-operand, a length [out]: sqrt-result	Pass/Fail
	Find n0 Prime	Computes the Montgomery arithmetic parameter n0'.	[in]: LSW of modulus [out]: VOID	Montgomery arithmetic parameter
	Mont Image v0	Computes the Montgomery Image (aM) of an unsigned integer a with respect to a modulus n.	[in]: a-operand, n-operand, a/n/x length [out]: x-operand	VOID
	Mont Image	Computes the Montgomery Image (aM) of an unsigned integer a with respect to a modulus n.	[in]: a-operand, n-operand, a/n/x length [out]: x-operand	VOID
	Mont Prod	Multi-precision Montgomery Product routine for unsigned integers.	[in]: a-operand, b-operand, n-operand, LSW of modulus, a/b/n/x length [out]: x-operand	VOID
	Mont Square	Multi-precision Montgomery Squaring routine for unsigned integers.	[in]: a-operand, n-operand, LSW of modulus, a/n/x length [out]: x-operand	VOID
	Rev Mont Image	This function converts a multi-precision integer from Montgomery representation to binary (normal) representation.	[in]: a-operand, n-operand, LSW of modulus, a/n/x length [out]: x-operand	VOID
	Mont Exp	Multi-precision Montgomery Exponentiation routine for unsigned integers.	[in]: b-operand, e-operand, e length, n-operand, b/n length [out]: x-operand	VOID
	Mont Mod Inverse	Computes a_inv = 1/aop (mod nop) using Fermat's Little Theorem.	[in]: a-operand, n-operand, a/n length [out]: a-inverse-result	VOID
	Mont Mod Sqrt	Computes the square root of a multi-precision operand (a) modulo a prime modulus (n).	[in]: a-operand, n-operand, a/n length [out]: a-sqrt-result	Pass/Fail
	Barrett	Calculates the modulus-	[in]: n-operand, n/x length	VOID

User Role				
Service		**Description**	**Input/Output**	**Return**
	Inverse	dependent quantity.	[out]: x-operand	
	Barrett Mod Multiply	Multi-precision modular multiplication routine for unsigned integers.	[in]: a-operand, b-operand, n-operand, u-operand, a/b/n/x length [out]: x-operand	VOID
	Barrett Exp	Multi-precision exponentiation routine for unsigned integers.	[in]: b-operand, e-operand, e length, n-operand, u-operand, b/n length [out]: x-operand	VOID
	Barrett Mod Inverse	Computes a_inv = 1/aop (mod nop) using Fermat's Little Theorem.	[in]: a-operand, n-operand, a/n length [out]: a-inverse-result	VOID
	Barrett Mod Sqrt	Computes the square root of a multi-precision operand (a) modulo a prime modulus (n).	[in]: a-operand, n-operand, a/n length [out]: a-sqrt-result	Pass/Fail
	Probab Mod Sqrt	General probabilistic algorithm to compute the square root modulo a prime number.	[in]: a-operand, n-operand, a/n length [out]: a-sqrt-result	Pass/Fail
	Probab Mod Sqrt v2	General probabilistic algorithm to compute the square root modulo a prime number.	[in]: a-operand, n-operand, a/n length [out]: a-sqrt-result	Pass/Fail
	Probab Mod Sqrt v1	General probabilistic algorithm to compute the square root modulo a prime number.	[in]: a-operand, n-operand, a/n length [out]: a-sqrt-result	Pass/Fail
	Probab Mod Sqrt v0	General probabilistic algorithm to compute the square root modulo a prime number.	[in]: a-operand, n-operand, a/n length [out]: a-sqrt-result	Pass/Fail
	Jacobi Symbol	Computes the Jacobi symbol for an integer a and an odd modulus n	[in]: a-operand, n-operand, a/n length [out]: VOID	1 if a in QR(n), else -1/0
	Destruct	Destructs the FFC object.	[in]: VOID [out]: VOID	VOID
KAS ECC	Construct	Constructs a KAS ECC object.	[in]: KAS type, initiator id, responder id, algorithm id, MAC key length, MAC tag length [out]: VOID	KAS ECC object
	Type Select	Changes the KAS type.	[in]: KAS type, initiator id, responder id, algorithm id, MAC key length, MAC tag length	Pass/Fail

User Role				
Service		**Description**	**Input/Output**	**Return**
			[out]: VOID	
	ECDH Init 1	Computes Phase 1 of Full Unified Model on initiator side.	[in]: Initiator ephemeral private key [out]: Initiator ephemeral public key	Pass/Fail
	ECDH Resp 1	Computes Phase 1 of Full Unified Model on responder side.	[in]: Responder static private key, Responder static public key, Responder ephemeral private key, Initiator static public key, Initiator ephemeral public key, Nonce [out]: Responder ephemeral public key, MAC key, AES initiator/responder keys, Responder MAC tag	Pass/Fail
	ECDH Init 2	Computes Phase 2 of Full Unified Model on initiator side.	[in]: Initiator static private key, Initiator static public key, Initiator ephemeral private key, Initiator ephemeral public key, Nonce, Responder static public key, Responder ephemeral public key, Responder MAC tag, [out]: AES initiator/responder keys, Initiator MAC tag	Pass/Fail
	ECDH Resp 2	Computes Phase 2 of Full Unified Model on responder side.	[in]: Responder ephemeral public key, MAC key, Initiator ephemeral public key, Initiator MAC tag [out]: VOID	Pass/Fail
	MQV Primitive	Computes the full form of the ECC MQV primitive.	[in]: Initiator static private key, Initiator ephemeral private key, Initiator ephemeral public key, Responder static public key, Responder ephemeral public key [out]: Shared secret	Pass/Fail
	MQV Init 1	Computes Phase 1 of Full MQV Model on initiator side.	[in]: Initiator ephemeral private key [out]: Initiator ephemeral public key	Pass/Fail
	MQV Resp 1	Computes Phase 1 of Full MQV Model on responder side.	[in]: Responder static private key, Responder static public key, Responder ephemeral private key, Initiator static	Pass/Fail

User Role				
Service		**Description**	**Input/Output**	**Return**
			public key, Initiator ephemeral public key, Nonce [out]: Responder ephemeral public key, MAC key, AES initiator/responder keys, Responder MAC tag	
	MQV Init 2	Computes Phase 2 of Full MQV Model on initiator side.	[in]: Initiator static private key, Initiator static public key, Initiator ephemeral private key, Initiator ephemeral public key, Nonce, Responder static public key, Responder ephemeral public key, Responder MAC tag, [out]: AES initiator/responder keys, Initiator MAC tag	Pass/Fail
	MQV Resp 2	Computes Phase 2 of Full MQV Model on responder side.	[in]: Responder ephemeral public key, MAC key, Initiator ephemeral public key, Initiator MAC tag [out]: VOID	Pass/Fail
	Destruct	Destructs the KAS ECC object.	[in]: VOID [out]: VOID	VOID
SHA	Construct	Constructs a SHA object.	[in]: SHA type [out]: VOID	SHA object
	Type Select	Changes the SHA type.	[in]: SHA type [out]: VOID	Pass/Fail
	Proc Message	Generates a message digest.	[in]: Message, Message length [out]: Digest	VOID
	Proc Message	Generates a message digest.	[in]: SHA type, Message, Message length [out]: Digest	VOID
	Proc Init	Initializes first message digest segment.	[in]: Message, Message length [out]: VOID	VOID
	Proc Init	Initializes first message digest segment.	[in]: SHA type, Message, Message length [out]: VOID	VOID
	Proc Update	Updates middle segment message digest segment.	[in]: Message, Message length [out]: VOID	VOID
	Proc Final	Generates final message digest.	[in]: Message, Message length [out]: Digest	VOID
	160 Proc Message	Generates a message digest.	[in]: Message, Message length, SHA mode [out]: Digest	VOID
	HMAC Proc	Generates a Keyed-Hash	[in]: Message, Message	VOID

User Role				
Service		**Description**	**Input/Output**	**Return**
	Message	Message Authentication Code (HMAC) digest.	length, key, key length [out]: Digest	
	HMAC Proc Message	Generates a HMAC tag.	[in]: Message, Message length, key, key length [out]: MAC tag, MAC tag length	VOID
	HMAC Proc Init	Initializes first HMAC message digest segment.	[in]: Message, Message length, key, key length [out]: VOID	VOID
	HMAC Proc Update	Updates middle HMAC segment message digest segment.	[in]: Message, Message length [out]: VOID	VOID
	HMAC Proc Final	Generates final HMAC message digest.	[in]: Message, Message length [out]: Digest	VOID
	HMAC Proc Final	Generates final HMAC message digest.	[in]: Message, Message length [out]: MAC tag, MAC tag length	VOID
	KDF CTR/FB/DPI	Generates a derived key.	[in]: Label/IV, Label length, Context, Context length, Counter length, Counter location [out]: Derived key, Derived key length	VOID
	PBKDF	Generates a derived key from password and salt.	[in]: Password, Password length, Salt, Salt length, iteration count [inout]: Derived key length [out]: Derived key	VOID
	Destruct	Zeroizes SHA buffers.	[in]: VOID [out]: VOID	VOID
Util's	Zeroize	Zeroizes fixed-size buffers.	[inout]: Buffer	VOID
	Obfuscate	Zeroized fixed-size buffer with random data from DRBG.	[inout]: Buffer	VOID
	Word Str Clr	Zeroizes buffer.	[in]: Buffer length [inout]: Buffer	VOID
	Word Str Cpy	Copies buffer.	[in]: Input Buffer, Buffer length [out]: Copied buffer	VOID
	Word Str Diff	Differences buffers.	[in]: Buffer a, Buffer b, a/b length [out]: VOID	Non-zero value indicates difference
	Word Str Cmp	Compares buffers.	[in]: Buffer a, Buffer b, a/b length	Pass/Fail

User Role				
Service		**Description**	**Input/Output**	**Return**
			[out]: VOID	
	Word Str Cmp v0	Compares buffer to zero.	[in]: Buffer, Buffer length [out]: VOID	Pass/Fail
	Word Str Cmp v1	Compares buffer to zero.	[in]: Buffer, Buffer length [out]: VOID	Pass/Fail
	My Mem Cmp K	Compares byte buffer to byte.	[in]: Buffer, Buffer length, byte value [out]: VOID	Pass/Fail
	CleanUp	Zeroizes word buffer and verifies zeroed.	[in]: Buffer length [inout]: Buffer	VOID
	CleanUp	Zeroizes byte buffer and verifies zeroed.	[in]: Buffer length [inout]: Buffer	VOID
	Words 2 Bytes	Converts word buffer to byte buffer.	[in]: Word buffer, Word buffer length [out]: byte buffer	VOID
	Bytes 2 Words	Converts byte buffer to word buffer.	[in]: Byte buffer, Word buffer length [out]: Word buffer	VOID
	DWords 2 Bytes	Converts double word buffer to byte buffer.	[in]: DWord buffer, DWord buffer length [out]: byte buffer	VOID
	Bytes 2 DWords	Converts byte buffer to double word buffer.	[in]: Byte buffer, DWord buffer length [out]: DWord buffer	VOID
	Quick Random Bytes	Generates pseudo-random bytes from DRBG.	[in]: Buffer length [out]: Buffer	Pass/Fail
	Stristr	Case-insensitive substring search	[in]: Buffer, search string [out]: VOID	Substring
	My Memi Cmp	Case-insensitive byte buffer comparison	[in]: Buffer a, Buffer b, a/b length [out]: VOID	Non-zero value indicates difference
	Scan Hex Data	Decodes a byte string buffer into a byte buffer.	[in]: String buffer [out]: Byte buffer	Length of byte buffer
	Scan Hex Data	Decodes a byte string buffer into a word buffer.	[in]: String buffer [out]: Word buffer	Length of word buffer
	Scan Hex Align Right	Decodes a byte string buffer into a word buffer with right alignment.	[in]: String buffer [inout]: Word buffer length [out]: Word buffer	VOID
	Read Dec Param	Reads decimal parameter from input file stream.	[in]: Input file stream, Offset header [out]: VOID	Decimal parameter
	Scan Hex Data	Decodes a byte string from an input stream into a word buffer.	[in]: Input file stream, Bit length, Offset header [out]: Word buffer	VOID

User Role				
Service		Description	Input/Output	Return
	Scan Hex Data	Decodes a byte string from an input stream into a byte buffer.	[in]: Input file stream, Bit length, Offset header [out]: Byte buffer	VOID
	Scan Hex Data	Decodes a byte string from an input stream into a word buffer.	[in]: Input file stream, Offset header [out]: Word buffer	Length of word buffer
	Scan Hex Align Right	Decodes a byte string from an input stream into a word buffer with right alignment.	[in]: Input file stream, Word buffer length, Offset header [out]: Word buffer	Pass/Fail
	Write Hex Data	Encodes word buffer into string buffer.	[in]: String buffer, Word buffer length [out]: Word buffer	VOID
	Write Hex Data	Encodes byte buffer into string buffer.	[in]: String buffer, Byte buffer length [out]: Byte buffer	VOID
	Write Hex Data	Writes word buffer into output stream as a string.	[in]: Output file stream, Word buffer length, Offset header, Skip zeros [out]: Word buffer	VOID
	Write Hex Data	Writes byte buffer into output stream as a string.	[in]: Output file stream, Byte buffer length, Offset header [out]: Byte buffer	VOID

Table 6 – Module Services for User Role

2.3.3. Authentication

The Module does not support operator authentication. Roles are selected implicitly based on the service performed by the operator.

Role	Type of Authentication	Authentication Data
Cryptographic Officer	N/A	N/A
User	N/A	N/A

Table 7 – Module Authentication

2.4. Finite State Model

The Finite State Model (FSM) describes the overall behavior and transitions the Module undergoes based upon its current state and commands received. The FSM was reviewed as part of the overall FIPS 140-2 validation.

2.5. Physical Security

The Module is implemented entirely in software, thus it is not subject to the FIPS 140-2 Physical Security requirements. The operational environment that executes the Module should be located on production-grade equipment and is expected to be secured by best practices.

2.6. Operational Environment

The Module runs in a single-user FIPS 140-2 certified operational environment where each calling application runs in a virtually separated, independent space and is compatible with the DRBG on which it runs based upon configuration. The Module is implemented entirely in software, and for FIPS 140-2 purposes, is classified as multi-chip standalone per the operational environment on which it runs.

Module	Operational Environment	CMVP Certificate	CAVP DRBG Certificate
`KEYWcryptoModule.dll`	Intel Xeon E5530 w/ Microsoft Windows Server 2012 R2 (64-bit)	#2357	#489, #523
`libKEYWcryptoModule.so.3`	Qualcomm Snapdragon 801 w/ BlackBerry OS 10.3	#1578	#81
`libKEYWcryptoModule.so.3`	Qualcomm Snapdragon S4 w/ BlackBerry OS 10.3	#1578	#81

Table 8 – Operational Environments

2.7. Cryptographic Key Management

The following table describes the cryptographic keys, key components and Critical Security Parameters (CSPs) utilized exclusively by the Module.

Key / CSP	Mode / Key/CSP Size	Use	Access Type	Input / Output	Storage	Destruction
HMAC Integrity Check Key	SHA-512	Symmetric key used for Software Integrity Checksum.	Crypto Officer Role: Read & Write	Symmetric key generated during each Module initialization as input where a new symmetric key is generated after each build. See Section 2.9 for more details on Software Integrity POST.	Held in RAM as plaintext temporarily for single-use and is not stored during Module initialization.	Zeroized immediately after Module initialization via zeroize service from Module API.
HMAC Integrity Checksum CSP	SHA-512	Checksum CSP used in Software Integrity Checksum.	Crypto Officer Role: Read & Write	Checksum CSP entered as input during each Module initialization where a new Checksum CSP is generated after each build.	Held in RAM as plaintext temporarily for single-use and is not stored during Module initialization.	Zeroized immediately after Module initialization via zeroize service from Module API.
AES-ECB Key	ECB-128 ECB-192	Symmetric key used for	User Role:	Symmetric key entered,	Held in RAM as plaintext.	Calling application is

Key / CSP	Mode / Key/CSP Size	Use	Access Type	Input / Output	Storage	Destruction
	ECB-256	encryption and decryption of user data.	Read & Write	established, or generated by operational environment DRBG as input.		responsible for zeroizing symmetric key via zeroize service from Module API or via platform-provided API.
AES-CBC Key	CBC-128 CBC-192 CBC-256	Symmetric key used for encryption and decryption of user data.	User Role: Read & Write	Symmetric key entered, established, or generated by operational environment DRBG as input and plaintext or ciphertext as output.	Held in RAM as plaintext.	Calling application is responsible for zeroizing symmetric key via zeroize service from Module API or via platform-provided API.
AES-CBC IV CSP	CBC-128 CBC-192 CBC-256	IV CSP used in encryption and decryption of user data.	User Role: Read & Write	IV CSP entered, established, or generated by operational environment DRBG as input and plaintext or ciphertext as output.	Held in RAM as plaintext.	Calling application is responsible for zeroizing IV CSP via zeroize service from Module API or via platform-provided API.
AES-GCM Key	GCM-128 GCM-192 GCM-256	Symmetric key used for encryption and decryption of traffic data.	User Role: Read & Write	Symmetric key entered, established, or generated by operational environment DRBG as input and plaintext or ciphertext with Tag as output.	Held in RAM as plaintext.	Calling application is responsible for zeroizing symmetric key via zeroize service from Module API or via platform-provided API.
AES-GCM IV CSP	GCM-128 GCM-192 GCM-256	IV CSP used in encryption and decryption of traffic data.	User Role: Read & Write	IV CSP entered, established, or generated by operational environment DRBG as input and plaintext or	Held in RAM as plaintext.	Calling application is responsible for zeroizing IV CSP via zeroize service from Module

Key / CSP	Mode / Key/CSP Size	Use	Access Type	Input / Output	Storage	Destruction
				ciphertext with Tag as output.		API or via platform-provided API.
AES-XTS Keys	XTS-128 XTS-256	Symmetric keys used for encryption and decryption of stored data.	User Role: Read & Write	Symmetric keys entered, established, or generated by operational environment DRBG as input and plaintext or ciphertext as output.	Held in RAM as plaintext.	Calling application is responsible for zeroizing symmetric keys via zeroize service from Module API or via platform-provided API.
AES-XTS Tweak Value CSP	XTS-128 XTS-256	Tweak value CSP used in encryption and decryption of stored data.	User Role: Read & Write	Tweak value CSP entered, established, or generated by operational environment DRBG as input and plaintext or ciphertext as output.	Held in RAM as plaintext.	Calling application is responsible for zeroizing Tweak value CSP via zeroize service from Module API or via platform-provided API.
AES-KW/KWP Key	KW-128 KW-192 KW-256 KWP-128 KWP-192 KWP-256	Symmetric key used for encryption and decryption of other keys.	User Role: Read & Write	Symmetric key entered, established, or generated by operational environment DRBG as input and plaintext or ciphertext as output.	Held in RAM as plaintext.	Calling application is responsible for zeroizing symmetric key via zeroize service from Module API or via platform-provided API.
CMAC Key	AES-128 AES-192 AES-256	Symmetric key used for message authentication.	User Role: Read & Write	Symmetric key entered, established, or generated by operational environment DRBG as input and MAC as output.	Held in RAM as plaintext.	Calling application is responsible for zeroizing symmetric key via zeroize service from Module API or via platform-provided API.

Key / CSP	Mode / Key/CSP Size	Use	Access Type	Input / Output	Storage	Destruction
GMAC Key	AES-128 AES-192 AES-256	Symmetric key used for message authentication.	User Role: Read & Write	Symmetric key entered, established, or generated by operational environment DRBG as input and MAC as output.	Held in RAM as plaintext.	Calling application is responsible for zeroizing symmetric key via zeroize service from Module API or via platform-provided API.
GMAC IV CSP	AES-128 AES-192 AES-256	IV CSP used for message authentication.	User Role: Read & Write	IV CSP entered, established, or generated by operational environment DRBG as input and MAC as output.	Held in RAM as plaintext.	Calling application is responsible for zeroizing IV CSP via zeroize service from Module API or via platform-provided API.
HMAC Key	SHA-1 (SHA-160) SHA-224 SHA-256 SHA-384 SHA-512 SHA-512/224 SHA-512/256	Symmetric key used for message authentication.	User Role: Read & Write	Symmetric key entered, established, or generated by operational environment DRBG as input and MAC as output.	Held in RAM as plaintext.	Calling application is responsible for zeroizing symmetric key via zeroize service from Module API or via platform-provided API.
ECDSA Key	P-192: SHA-1 (SHA-160) SHA-224 SHA-256 SHA-384 SHA-512 SHA-512/224 SHA-512/256 P-224: SHA-1 (SHA-160) SHA-224 SHA-256 SHA-384 SHA-512	Asymmetric key used for digital signature. Per NIST SP 800-131A, P-192 and SHA-1 are no longer considered secure and shall not be used to generate digital	User Role: Read & Write	Asymmetric key entered or generated by operational environment DRBG as input and digital signature scalars computed as output.	Held in RAM as plaintext.	Calling application is responsible for zeroizing asymmetric key via zeroize service from Module API or via platform-provided API.

Key / CSP	Mode / Key/CSP Size		Use	Access Type	Input / Output	Storage	Destruction
		SHA-512/224	signatures (Ref. [14]).				
		SHA-512/256					
	P-256	SHA-1 (SHA-160)					
		SHA-224					
		SHA-256					
		SHA-384					
		SHA-512					
		SHA-512/224					
		SHA-512/256					
	P-384	SHA-1 (SHA-160)					
		SHA-224					
		SHA-256					
		SHA-384					
		SHA-512					
		SHA-512/224					
		SHA-512/256					
	P-521	SHA-1 (SHA-160)					
		SHA-224					
		SHA-256					
		SHA-384					
		SHA-512					
		SHA-512/224					
		SHA-512/256					
ECC KAS Keys	FullUnified KC EB P-224, SHA-224		Asymmetric keys and MAC keys used for key establishment.	User Role: Read & Write	Asymmetric keys and MAC keys entered or generated by operational environment DRBG as input and symmetric keys derived as output.	Held in RAM as plaintext.	Calling application is responsible for zeroizing asymmetric/symmetric keys via zeroize service from Module API or via platform-provided API.
	FullUnified KC EC P-256, SHA-256						
	FullUnified KC ED P-384, SHA-384						
	FullUnified KC EE P-521, SHA-512						
	FullMQV KC EB P-224, SHA-224						
	FullMQV KC EC P-256, SHA-256						
	FullMQV KC ED P-384, SHA-384						
	FullMQV KC EE P-521, SHA-512						

Key / CSP	Mode / Key/CSP Size	Use	Access Type	Input / Output	Storage	Destruction
ECC KAS Nonce & MAC tag CSPs	FullUnified KC EB P-224, SHA-224	Nonce and MAC tag CSPs used in key establishment.	User Role: Read & Write	Nonce and MAC tag CSPs entered or generated by operational environment DRBG as input and symmetric keys derived as output.	Held in RAM as plaintext.	Calling application is responsible for zeroizing Nonce and MAC tag CSPs via zeroize service from Module API or via platform-provided API.
	FullUnified KC EC P-256, SHA-256					
	FullUnified KC ED P-384, SHA-384					
	FullUnified KC EE P-521, SHA-512					
	FullMQV KC EB P-224, SHA-224					
	FullMQV KC EC P-256, SHA-256					
	FullMQV KC ED P-384, SHA-384					
	FullMQV KC EE P-521, SHA-512					
ECC KAS Shared Secret & DKM CSPs	FullUnified KC EB P-224, SHA-224	Shared Secret and DKM CSPs derived during key establishment.	User Role: N/A	Shared Secret and DKM CSPs derived as output between KAS phases.	Held in RAM as plaintext temporarily for single-use and is not stored between KAS phases.	Zeroized immediately between KAS phases via zeroize service from Module API.
	FullUnified KC EC P-256, SHA-256					
	FullUnified KC ED P-384, SHA-384					
	FullUnified KC EE P-521, SHA-512					
	FullMQV KC EB P-224, SHA-224					
	FullMQV KC EC P-256, SHA-256					
	FullMQV KC ED P-384, SHA-384					
	FullMQV KC EE P-521, SHA-512					
ECC CDH Primitive Keys	P-224	Asymmetric keys used for shared secret CSP establishment.	User Role: Read & Write	Asymmetric keys, entered or generated by operational environment DRBG as input and shared secret CSP derived as output.	Held in RAM as plaintext.	Calling application is responsible for zeroizing asymmetric keys via zeroize service from Module API or via platform-provided API.
	P-256					
	P-384					
	P-521					
ECC CDH Primitive	P-224	Shared secret CSPs derived	User Role:	Shared secret CSP derived as	Held in RAM as plaintext.	Calling application is
	P-256					

Key / CSP	Mode / Key/CSP Size	Use	Access Type	Input / Output	Storage	Destruction
Shared Secret CSPs	P-384 P-521	from establishment.	Read & Write	output when asymmetric keys entered or generated by operational environment DRBG as input.		responsible for zeroizing shared secret CSPs via zeroize service from Module API or via platform-provided API.
KBKDF-CMAC-CTR Keys	CMAC-AES-128 CMAC-AES-192 CMAC-AES-256	Symmetric key used for key derivation.	User Role: Read & Write	Symmetric key entered, established, or generated by operational environment DRBG as input and symmetric key derived as output.	Held in RAM as plaintext.	Calling application is responsible for zeroizing symmetric keys via zeroize service from Module API or via platform-provided API.
KBKDF-CMAC-FB Keys	CMAC-AES-128 CMAC-AES-192 CMAC-AES-256	Symmetric key used for key derivation.	User Role: Read & Write	Symmetric key entered, established, or generated by operational environment DRBG as input and symmetric key derived as output.	Held in RAM as plaintext.	Calling application is responsible for zeroizing symmetric keys via zeroize service from Module API or via platform-provided API.
KBKDF-CMAC-FB IV CSP	CMAC-AES-128 CMAC-AES-192 CMAC-AES-256	IV CSP used in key derivation.	User Role: Read & Write	IV CSP entered, established, or generated by operational environment DRBG as input and symmetric key derived as output.	Held in RAM as plaintext.	Calling application is responsible for zeroizing IV CSP via zeroize service from Module API or via platform-provided API.
KBKDF-CMAC-DPI Keys	CMAC-AES-128 CMAC-AES-192 CMAC-AES-256	Symmetric key used for key derivation.	User Role: Read & Write	Symmetric key entered, established, or generated by	Held in RAM as plaintext.	Calling application is responsible for zeroizing

Key / CSP	Mode / Key/CSP Size	Use	Access Type	Input / Output	Storage	Destruction
				operational environment DRBG as input and symmetric key derived as output.		symmetric keys via zeroize service from Module API or via platform-provided API.
KBKDF-HMAC-CTR Keys	HMAC-SHA-1 (SHA-160) HMAC-SHA-224 HMAC-SHA-256 HMAC-SHA-384 HMAC-SHA-512	Symmetric key used for key derivation.	User Role: Read & Write	Symmetric key entered, established, or generated by operational environment DRBG as input and symmetric key derived as output.	Held in RAM as plaintext.	Calling application is responsible for zeroizing symmetric keys via zeroize service from Module API or via platform-provided API.
KBKDF-HMAC-FB Keys	HMAC-SHA-1 (SHA-160) HMAC-SHA-224 HMAC-SHA-256 HMAC-SHA-384 HMAC-SHA-512	Symmetric key used for key derivation.	User Role: Read & Write	Symmetric key entered, established, or generated by operational environment DRBG as input and symmetric key derived as output.	Held in RAM as plaintext.	Calling application is responsible for zeroizing symmetric keys via zeroize service from Module API or via platform-provided API.
KBKDF-HMAC-FB IV CSP	HMAC-SHA-1 (SHA-160) HMAC-SHA-224 HMAC-SHA-256 HMAC-SHA-384 HMAC-SHA-512	IV CSP used in key derivation.	User Role: Read & Write	IV CSP entered, established, or generated by operational environment DRBG as input and symmetric key derived as output.	Held in RAM as plaintext.	Calling application is responsible for zeroizing IV CSP via zeroize service from Module API or via platform-provided API.
KBKDF-HMAC-DPI Keys	HMAC-SHA-1 (SHA-160) HMAC-SHA-224 HMAC-SHA-256 HMAC-SHA-384 HMAC-SHA-512	Symmetric key used for key derivation.	User Role: Read & Write	Symmetric key entered, established, or generated by operational environment	Held in RAM as plaintext.	Calling application is responsible for zeroizing symmetric keys via

Key / CSP	Mode / Key/CSP Size	Use	Access Type	Input / Output	Storage	Destruction
				DRBG as input and symmetric key derived as output.		zeroize service from Module API or via platform-provided API.
PBKDF Password CSP	HMAC-SHA-1 (SHA-160) HMAC-SHA-224 HMAC-SHA-256 HMAC-SHA-384 HMAC-SHA-512	Password CSP used in password-based key derivation.	User Role: Read & Write	Password CSP entered by calling application as input and symmetric key derived as output.	Held in RAM as plaintext.	Calling application is responsible for zeroizing Password CSP via zeroize service from Module API or via platform-provided API.
PBKDF Key	HMAC-SHA-1 (SHA-160) HMAC-SHA-224 HMAC-SHA-256 HMAC-SHA-384 HMAC-SHA-512	Symmetric key derived from password-based key derivation.	User Role: Read & Write	Symmetric key derived as output when Password CSP entered by calling application as input.	Held in RAM as plaintext.	Calling application is responsible for zeroizing symmetric key via zeroize service from Module API or via platform-provided API.

Table 9 – Module Cryptographic Keys and Critical Security Parameters

2.7.1. Key Zeroization

The Module API leverages fixed-size buffer zeroization via `memset` and pseudorandom buffer filling. The Cryptographic Officer operator may request HMAC Integrity Check Key zeroization at any time by power-cycling the operational environment and reloading the Module. Also, the Cryptographic Officer operator may manually uninstall the Module from the operational environment and reformat (i.e. overwrite at least once) the platform's hard drive or other permanent storage media while only performing the procedural uninstallation of the Module is not an acceptable key zeroization method. The User operator must zeroize keys/CSPs stored in the operational environment by calling a zeroize service provided by the Module API or via platform-provided API.

2.8. Electromagnetic Interference and Compatibility

The Module meets the requirements of the FIPS 140-2 EMI/EMC Level 1 specification as the operational environment on which the Module software runs passed validation executing upon the general-purpose computer (GPC) that confirms to the EMI/EMC requirements specific by 47 Code of Federal Regulations, Part 15, Subpart B, Unintentional Radiators, Digital Devices, Class A (i.e., for business use).

2.9. Self-Tests

The Module implements Power-On Self-Tests (POST) and conditional self-tests that are described in the following tables:

Test	Description
Software Integrity	The Module validates its own software integrity upon load of the Module DLL/SO file. The integrity check is a two-step process consisting of an HMAC verification (based on the FIPS-approved HMAC-512 algorithm), applied to the whole Module DLL/SO image processed as a binary data file. In the first step, the 512-bit (64-byte) HMAC key for the HMAC verification is derived via a FIPS-approved KBKDF from several build-specific data fields including the current version string and build date, which are compiled into the Module and are not modifiable. This HMAC key customization is aimed at preventing malicious Module DLL/SO rebuilds and authenticating the original build only. In the second step, the 512-bit HMAC key is used to perform an HMAC-512 integrity check of the whole Module DLL/SO image. This computation produces a 512-bit checksum that is compared against a hexadecimal value pre-stored in a properties file.
AES Check Encryption/Decryption Tables	Verifies the integrity of the pre-built Sbox substitution table and inverse Sbox substitution table. The Sbox substitution table is pre-converted to four 32-bit tables, in order to speed up AES encryption in 32-bit processing mode while the inverse Sbox substitution table is pre-converted to four 32-bit tables, in order to speed up AES decryption in 32-bit processing mode.
GCM Encrypt/Decrypt KAT	Exercises a set of Known Answer Tests (KATs) extracted from the GCM test vectors published by NIST in the GCMVS specification (Reference [18]) on all three GCM encryption modes corresponding to AES key sizes of 128, 192 and 256 bits featuring the largest combinations of PT, IV and AAD. The comprehensive GCM KATs implicitly provide assurance about the validity of the underlying AES cryptographic algorithms.
SHA KAT	Exercises a set of Known Answer Tests (KATs) extracted from the SHA test vectors published by NIST in the SHAVS specification (Reference [21]) on all SHA versions (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256) specified in FIPS Publication 180-4 featuring mixed hash/digest size combinations with the longest input data. The comprehensive SHA KATs implicitly provide assurance about the validity of the Key Derivation Function (KDF) employed by the ECDH Key Agreement Scheme (as recommended in NIST SP 800-56A – Reference [15], a SHA-based concatenation KDF is being used).

Test	Description
HMAC KAT	Exercises a set of Known Answer Tests (KATs) extracted from the HMAC test vectors published by NIST in the HMACVS specification (Reference [22]) featuring the largest combinations of key and tag sizes covering all versions of the underlying hashing algorithm (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256). The comprehensive HMAC KATs implicitly provide assurance about the validity of the Bilateral Key Confirmation method employed by the ECDH Key Agreement Scheme (Reference [15], Section 8.4).
ECDSA KeyPair/PKV KAT	Exercises a set of Known Answer Tests (KATs) adapted from the ECDSA KeyPair (private/public key verification) and PKV (Public Key Validation) test vectors published by NIST in the ECDSA2VS specification (Reference [24]) covering each version of the underlying prime-field EC (P-192, P-224, P-256, P-384 and P-521). The ECDSA KeyPair tests include multiple KAT verifications of ECC point multiplication, which is the ECC primitive used for shared-secret ("Z") computation by the ECDH Key Agreement Scheme.
ECDSA SigGen KAT	Exercises a set of Known Answer Tests (KATs) adapted from the SigGen test vectors published by NIST in the ECDSA2VS specification (Reference [24]). In this test category, ECDSA2VS only provides the message to be signed. The module generates a private key, computes the corresponding public key, generates an ECDSA "secret number" (ephemeral key) from the DRBG, computes the message signature using the private key and verifies the signature with the public key. For completeness, the signature is verified with the private key as well. One long test vector is exercised for each combination of prime field EC (P-224, P-256, P-384 and P-521) and hashing algorithm (SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256). In the latest NIST Suite B specifications P-192 EC and SHA-1 are no longer considered suitable for secure ECDSA generation (Reference [14]).
ECDSA SigVer KAT	Exercises a set of Known Answer Tests (KATs) adapted from the SigVer test vectors published by NIST in the ECDSA2VS specification (Reference [24]). These test cases are in compliance with the latest ECDSA specification (FIPS 186-4, Reference [12]), which allows any prime-field EC (P-192, P-224, P-256, P-384 or P-521) to be combined with each SHA version from FIPS 180-4 (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 or SHA-512/256) in an ECDSA computation. One test case from each EC/SHA combination, featuring the longest message, is exercised.

Test	Description
ECDH Full Unified Key Agreement Scheme (KAS) KAT	Exercises a set of Known Answer Tests (KATs) adapted from the ECDH test vectors published by NIST in the KASVS specification (Reference [25]) featuring the Full Unified Model of ECDH covering each version of the underlying prime-field EC (P-224, P-256, P-384 and P-521). Each test run includes both Initiator-side and Responder-side functions. The underlying cryptographic algorithms used during ECDH key agreement are fully validated via individual POSTs: • ECC point multiplication is validated via ECDSA KeyPair KATs • The Key Derivation Function is validated via SHA KATs • The Key Confirmation function is validated via HMAC KATs
ECDH Full MQV Key Agreement Scheme (KAS) KAT	Exercises a set of Known Answer Tests (KATs) adapted from the ECDH test vectors published by NIST in the KASVS specification (Reference [25]) featuring the Full MQV model of ECDH covering each version of the underlying prime-field EC (P-224, P-256, P-384 and P-521). Each test run includes both Initiator-side and Responder-side functions. The underlying cryptographic algorithms used during ECDH key agreement are fully validated via individual POSTs: • ECC point multiplication is validated via ECDSA KeyPair KATs • The Key Derivation Function is validated via SHA KATs • The Key Confirmation function is validated via HMAC KATs
XTS Encrypt/Decrypt KAT	Exercises a set of Known Answer Tests (KATs) extracted from the XTS test vectors published by NIST in the XTSVS specification (Reference [19]). Both formats specified for the tweak value input (128-bit hexadecimal string or 64-bit Data Unit Sequence Number) are being tested with various, non-trivial Data Unit bit sizes in encrypt and decrypt mode. The comprehensive XTS KATs implicitly provide assurance about the validity of the underlying AES cryptographic algorithms.
KW/KWP Encrypt/Decrypt KAT	Exercises a set of Known Answer Tests (KATs) extracted from KW and KWP test vectors published by NIST with the Key Wrap Validation System (KWVS) specification (Reference [20]). All three encryption modes are tested for KW and KWP, corresponding to AES key sizes of 128, 192 and 256 bits. Also, the underlying AES block cipher is tested in either forward direction or inverse direction during KW/KWP encryption. Two non-trivial test vectors are exercised for each combination of AES key size, KW/KWP and forward/inverse block cipher. The comprehensive KW/KWP KATs implicitly provide assurance about the validity of the underlying AES cryptographic algorithms.

Test	Description
KBKDF KAT	Exercises a set of Known Answer Tests (KATs) extracted from KDF test vectors published by NIST with the Key Derivation using Pseudorandom Functions (SP800-108) Validation System (KBKDFVS) (Reference [23]). Both CMAC and HMAC algorithms are exercised as underlying pseudo-random function (PRF). For each PRF, SP800-108 specifies three modes of key derivation from a set of inputs: Counter Mode (CTR), FeedBack Mode (FB) and Double-Pipeline Iteration Mode (DPI), which are all represented during a KDF self-test run. At least one non-trivial test case has been included for each input parameter combination specified in KBKDFVS, adding up to 12 KDF CTR tests, 32 KDF FB tests and 16 KDF DPI tests.
PBKDF KAT	The comprehensive HMAC KATs implicitly provide assurance about the validity of the Password-Based Key Derivation Function (PBKDF) as recommended in IAW NIST SP 800-132 (Reference [11]). There is neither a Validation System in place, nor sample test vectors published by CAVP for the PBKDF algorithm, as of January 2017.

Table 10 – Module Power-On Self-Tests

Test	Description
ECC KAS (FullUnified, FullMQV) Conditional Pair-Wise Consistency Self-Test	The ECC KAS implementation provides built-in assurance (verification) of the arithmetic validity of each newly generated key pair by performing a pair-wise consistency self-test where the key pair is used in conjunction with a second newly generated compatible key pair to calculate shared values for both sides of the key agreement algorithm such that if the resulting shared values are not equal the self-test fails. Every invocation of ECC KAS involves (within the class constructors) a verification of the arithmetic validity of the selected set of ECC domain parameters (Reference [15], Section 5.5.2). The ECC KAS implementation performs a full ECC public key validation each time such a key is being used where each side verifies both own and opposite static public keys, each side verifies opposite side's ephemeral public key (Reference [15], Section 5.6.2). Also, during key agreement, each side renews its assurance of possessing the correct private key by using the Key Regeneration method (Reference [15], Section 5.6.3), while the ephemeral (generated) private key is subjected to the constraints specified in Reference [15], Section 5.6.1.2.

Test	Description
ECDSA Conditional Pair-Wise Consistency Self-Test	The ECDSA implementation provides built-in assurance (verification) of the arithmetic validity of each newly generated key pair by performing a pair-wise consistency self-test where the key pair is used to generate and verify a digital signature such that if the digital signature cannot be verified the self-test fails. Every invocation of ECDSA involves (within the class constructors) a verification of the arithmetic validity of the selected set of ECC domain parameters. The ECDSA implementation performs an ECC public key validation each time such a key is used during digital signature generation and verification.

Table 11 – Module Conditional Self-Tests

2.9.1. Invoking Self-Tests

The Cryptographic Officer operator invokes the POST automatically by loading the Module. During load the operational environment executes the following Module Default Entry Point (DEP) automatically, which invokes the self-tests. The Module does not rely on any other external service to initiate the POST and all data output via the data output interface is inhibited when the POST is performed. The POST may be invoked automatically at any time by power-cycling the operational environment and reloading the Module.

Dynamic Link Library (DLL) Default Entry Point

```
BOOL APIENTRY DLLMain( HMODULE hModule,
                       DWORD ul_reason_for_call,
                       LPVOID lpReserved)
```

Shared Object (SO) Default Entry Point

```
void __attribute__((constructor)) runModulePOST(void)
```

2.9.2. Self-Tests Results

Upon successful self-test completion, the Module will complete its initialization and transition to the idle operational state. Subsequent Module self-tests are exercised automatically when any Suite B cryptographic algorithms are called by the operator, either for communications encryption/decryption, data encryption/decryption, and/or during key establishment. In the event the Software Integrity and/or KAT self-test fail, the Module will not complete loading and will transition to the error state and a specific error code will be returned indicating which self-test has failed. The Module will not provide any cryptographic services while in this error state. Recovery from the error state is possible by power-cycling the operational environment and reloading the Module.

Self-Test	Error Code
Software Integrity	441, 444
GCM Encrypt	2100 + Test Count
GCM Decrypt	2200 + Test Count
SHA	2300 + Test Count
HMAC	2400 + Test Count

Self-Test	Error Code
ECDSA Key	2800 + Test Count
ECDSA SigGen	3300 + Test Count
ECDSA SigVer	3400 + Test Count
KAS Full Unified	2500 + Test Count (combined indicator of the EC type and failing sub-test)
KAS Full MQV	3000 + Test Count
XTS Encrypt	2600 + Test Count
XTS Decrypt	2700 + Test Count
KW Encrypt	3100 + Test Count
KW Decrypt	3200 + Test Count
KBKDF	3500 + Test Count

Table 12 – Module Self-Test Error Codes

2.10. Design Assurance

The Module meets the requirements of the FIPS 140-2 Security Level 1 specification and provides the following Cryptographic Officer guidance and User guidance.

The Cryptographic Officer is responsible for manually installing the Module on the operational environment and ensuring FIPS mode of operation as described in Section 2.1.2. Also, the Cryptographic Officer is responsible for initializing the Module causing the POST to run automatically as described in Section 2.9.

The User operator is responsible for confining method calls to only FIPS 140-2 approved security functions as listed in Table 2 when calling the Module API as well as confining method calls to a FIPS 140-2 approved DRBG from the operational environment as listed in Section 2.6.

2.11. Mitigation of Other Attacks

The Module has not been designed to mitigate any specific attacks outside the scope of the FIPS 140-2 requirements. The Module resides within a FIPS 140-2 operational environment, which provides an additional layer of protection to attacks of the Module.

3. Referenced Documents

[1] FIPS Publication 197, The Advanced Encryption Standard (AES), U.S. DoC/NIST, November 26, 2001, National Institute of Standards and Technology, [Web page], http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[2] NIST Special Publication 800-38A, Recommendation for Block Cipher Modes of Operation: Methods and Techniques, December 2001, [Web page], http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

[3] NIST Special Publication 800-38B, Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, May 2005, National Institute of Standards and Technology, [Web page], http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf

[4] NIST Special Publication 800-38D, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, November 2007, National Institute of Standards and Technology, [Web page], http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf

[5] NIST Special Publication 800-38E, Recommendation for Block Cipher Modes of Operation: the XTS-AES Mode for Confidentiality on Storage Devices, January 2010, National Institute of Standards and Technology, [Web page], http://csrc.nist.gov/publications/nistpubs/800-38E/nist-sp-800-38E.pdf

[6] NIST Special Publication 800-38F, Recommendation for Block Cipher Modes of Operation: Methods of Key Wrapping, December 2012, National Institute of Standards and Technology, [Web page], http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38F.pdf

[7] RFC 5649, Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm, August 2009, Network Working Group, [Web page], https://tools.ietf.org/html/rfc5649

[8] FIPS Publication 180-4, Secure Hash Standard (SHS), August 2015, National Institute of Standards and Technology, [Web page], http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[9] FIPS Publication 198-1, The Keyed-Hash Message Authentication Code (HMAC), July 2008, National Institute of Standards and Technology, [Web page], http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf

[10] NIST Special Publication 800-108, Recommendation for Key Derivation Using Pseudorandom Functions, October 2009, National Institute of Standards and Technology, [Web page], http://csrc.nist.gov/publications/nistpubs/800-108/sp800-108.pdf

[11] NIST Special Publication 800-132, Recommendation for Password-Based Key Derivation, December 2010, National Institute of Standards and Technology, [Web page], http://csrc.nist.gov/publications/nistpubs/800-132/nist-sp800-132.pdf

[12] FIPS Publication 186-4, Digital Signature Standard (DSS), July 2013, National Institute of Standards and Technology, [Web page], http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

[13] ANS X9.62-2005: Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), November 2005

[14] NIST Special Publication 800-131A, Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths, January 2011, National Institute of Standards and Technology, [Web page], http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf

[15] NIST Special Publication 800-56A, Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography, Revision 2, May 2013, National Institute of Standards and Technology, [Web page], http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar2.pdf

[16] The Advanced Encryption Standard Algorithm Validation Suite (AESAVS), November 15, 2002, National Institute of Standards and Technology, [Web page], http://csrc.nist.gov/groups/STM/cavp/documents/aes/AESAVS.pdf

[17] The CMAC Validation System (CMACVS), Updated August 23, 2011, National Institute of Standards and Technology, [Web page], http://csrc.nist.gov/groups/STM/cavp/documents/mac/CMACVS.pdf
[18] The Galois/Counter Mode (GCM) and GMAC Validation System (GCMVS), National Institute of Standards and Technology, Updated: August 30, 2012, [Web page], http://csrc.nist.gov/groups/STM/cavp/documents/mac/gcmvs.pdf
[19] The XTS-AES Validation System (XTSVS), Updated: September 5, 2013, National Institute of Standards and Technology, [Web page], http://csrc.nist.gov/groups/STM/cavp/documents/aes/XTSVS.pdf
[20] The Key Wrap Validation System (KWVS), June 20, 2014, National Institute of Standards and Technology, [Web page], http://csrc.nist.gov/groups/STM/cavp/documents/mac/KWVS.pdf
[21] The Secure Hash Algorithm Validation System (SHAVS), Updated: May 21, 2014, National Institute of Standards and Technology, [Web page], http://csrc.nist.gov/groups/STM/cavp/documents/shs/SHAVS.pdf
[22] The Keyed-Hash Message Authentication Code Validation System (HMACVS), Updated: July 23, 2012, National Institute of Standards and Technology, [Web page], http://csrc.nist.gov/groups/STM/cavp/documents/mac/HMACVS.pdf
[23] Key Derivation using Pseudorandom Functions (SP 800-108) Validation System (KBKDFVS), Updated January 4, 2016, National Institute of Standards and Technology, [Web page], http://csrc.nist.gov/groups/STM/cavp/documents/KBKDF800-108/kbkdfvs.pdf
[24] The FIPS 186-4 Elliptic Curve Digital Signature Algorithm Validation System (ECDSA2VS), Updated: March 18, 2014, National Institute of Standards and Technology, [Web page], http://csrc.nist.gov/groups/STM/cavp/documents/dss2/ecdsa2vs.pdf
[25] The Key Agreement Schemes Validation System (KASVS), Updated May 22, 2014, National Institute of Standards and Technology, [Web page], http://csrc.nist.gov/groups/STM/cavp/documents/keymgmt/KASVS.pdf
[26] NIST Special Publication 800-63-2, Electronic Authentication Guideline, August 2013, National Institute of Standards and Technology [Web page], http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63-2.pdf
[27] NIST Special Publication 800-118, Guide to Enterprise Password Management (Draft), April 2009, National Institute of Standards and Technology, [Web page], http://csrc.nist.gov/publications/drafts/800-118/draft-sp800-118.pdf