Gallagher OpenSSL Cryptographic Module

FIPS 140-2 Non-Proprietary Security Policy

Software Version: 2.0.12

Contents

Document history

Edition	Date	Author	Comment
1.0	2018/02/22	Dave Herron	First Gallagher version. Document converted from the OpenSSL original to reflect the Gallagher rebranding of OpenSSL FIPS Object Module.
1.1	2020/03/19		Updated to reflect the SP800-131 transition for removing ANSI X9.31 and to reflect the IG G.18 FIPS 186-2 transition guidelines.

Acknowledgements

The Gallagher OpenSSL Cryptographic Module is a rebranding of the OpenSSL FIPS Object Module targeted and tested for specific Gallagher platforms. This document was derived from the security policy associated with the OpenSSL FIPS 140-2 validation certificate number 2398.

References

Reference	Full Specification Name
[ANS X9.31]	Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA)
[FIPS 140-2]	Security Requirements for Cryptographic modules, May 25, 2001
[FIPS 180-4]	Secure Hash Standard
[FIPS 186-4]	Digital Signature Standard
[FIPS 197]	Advanced Encryption Standard
[FIPS 198-1]	The Keyed-Hash Message Authentication Code (HMAC)
[SP 800-38B]	Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication
[SP 800-38C]	Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality
[SP 800-38D]	Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC
[SP 800-56A]	Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography
[SP 800-67R1]	Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher
[SP 800-89]	Recommendation for Obtaining Assurances for Digital Signature Applications
[SP 800-90A]	Recommendation for Random Number Generation Using Deterministic Random Bit Generators
[SP 800-131A]	Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths

1 Introduction

This document is the non-proprietary security policy for the Gallagher OpenSSL Cryptographic Module, hereafter referred to as the Module.

The Gallagher Module is a rebranding of the OpenSSL FIPS Object Module SE, tested and targeted for specific Gallagher platforms.

The Module is a software library providing a C-language application program interface (API) for use by other processes that require cryptographic functionality. The Module is classified by FIPS 140-2 as a software module, multi-chip standalone module embodiment. The physical cryptographic boundary is the Gallagher device on which the module is installed. The logical cryptographic boundary of the Module is the fipscanister object module, a single object module file named fipscanister.o (Linux®[1]) or fipscanister.lib (Microsoft Windows®[2]). The Module performs no communications other than with the calling application (the process that invokes the Module services).

The FIPS 140-2 security levels for the Module are as follows:

Security Requirement	**Security Level**
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles, Services, and Authentication	2
Finite State Model	1
Physical Security	NA
Operational Environment	1
Cryptographic Key Management	1
EMI/EMC	1
SelfTests	1
Design Assurance	3
Mitigation of Other Attacks	NA

Table 1 -- Security Level of Security Requirements

The Module's software version for this validation is 2.0.12, rebranded and based on the OpenSSL FIPS Object SE Module (certificate 2398) of the same version.

[1] Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.
[2] Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

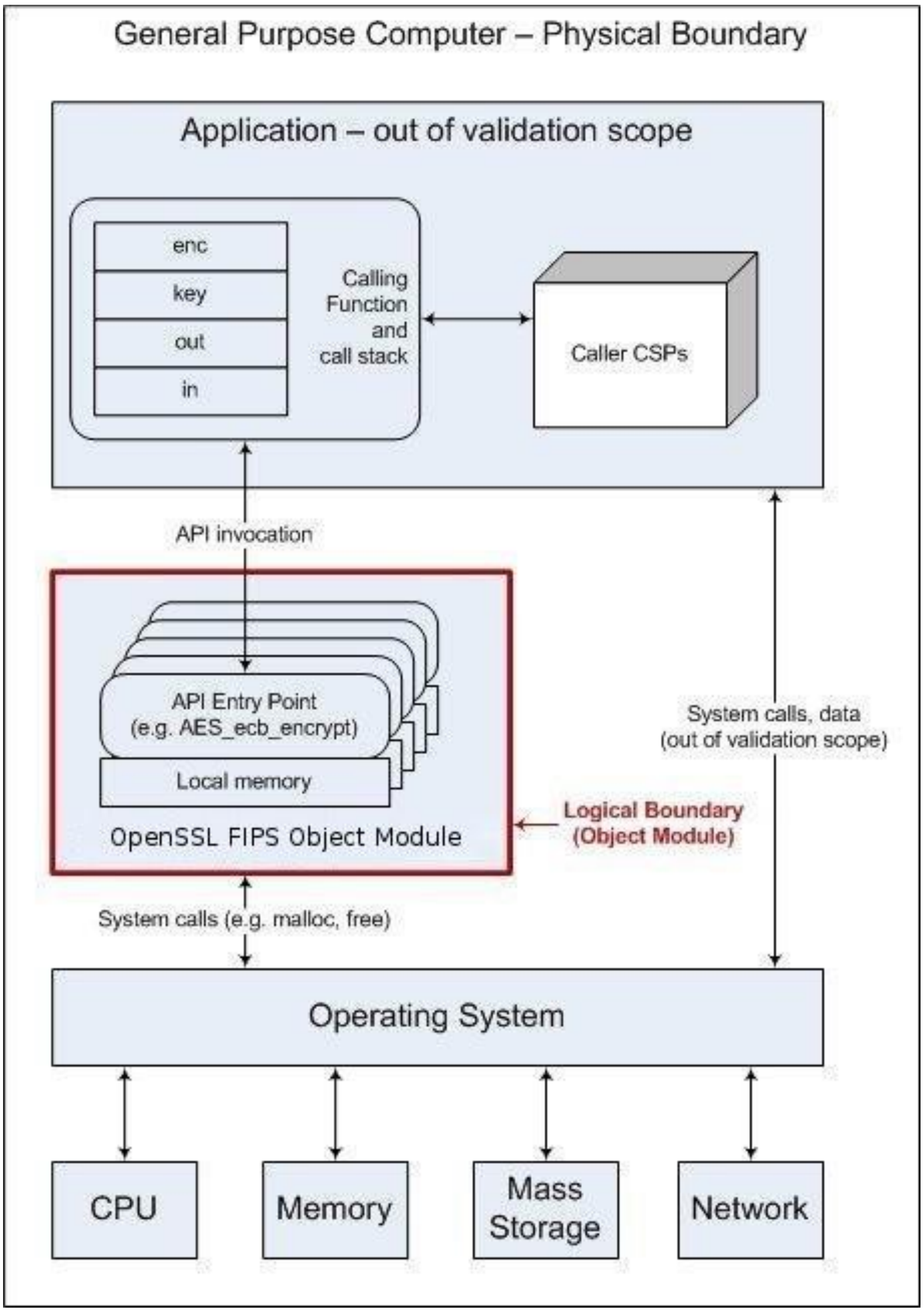
General Purpose Computer – Physical Boundary
Application – out of validation scope
enc
key
out
in
Calling Function and call stack
Caller CSPs
API invocation
API Entry Point (e.g. AES_ecb_encrypt)
Local memory
OpenSSL FIPS Object Module
Logical Boundary (Object Module)
System calls, data (out of validation scope)
System calls (e.g. malloc, free)
Operating System
CPU
Memory
Mass Storage
Network

Figure 1 -- Module Block Diagram

2 Tested Configurations

	Operational Environment	Processor	Optimisations (Target)	Elliptic Curve Support	Build Method
1	Linux 4.9	ARM926EJ-S (ARMv5TEJ)	None	BKP	U2

Table 2 -- Tested Configurations.
The Elliptic Curve column indicates support for prime curve only (P), or all NIST defined B, K, and P curves (BKP).
The Build Method column refers to methods listed in Appendix A

See Appendix A for additional information on build method and optimizations. See Appendix C for a list of the specific compilers used to generate the Module for the respective operational environments.

3 Ports and Interfaces

The physical ports of the Module are the same as the computer system on which it is executing. The logical interface is a C-language application program interface (API).

Logical Interface Type	Description
Control input	API entry point and corresponding stack parameters
Data input	API entry point data input stack parameters
Status output	API entry point return values and status stack parameters
Data output	API entry point data output stack parameters

Table 3 -- Logical interfaces

As a software module, control of the physical ports is outside module scope. However, when the module is performing self-tests, or is in an error state, all output on the logical data output interface is inhibited. The module is single-threaded and in error scenarios returns only an error value (no data output is returned).

4 Modes of Operation and Cryptographic Functionality

The Module supports only a FIPS 140-2 Approved mode. Tables 4a and 4b list the Approved and Non-approved but Allowed algorithms, respectively.

Function	Algorithm	Options	Cert #
Random Number Generation; Symmetric key Generation	[SP 800-90A][3] DRBG Prediction resistance supported for all variations	Hash DRBG HMAC DRBG, no reseed CTR DRBG (AES), no derivation function	1978
Encryption, Decryption and CMAC	[SP 800-67]	3-Key TDES TECB, TCBC, TCFB, TOFB; CMAC generate and verify	2640
	[FIPS 197] AES	128/ 192/256 ECB, CBC, OFB, CFB 1, CFB 8, CFB 128, CTR, XTS; CCM; GCM; CMAC generate and verify	5206
	[SP 800-38B] CMAC [SP 800-38C] CCM [SP 800-38D] GCM [SP 800-38E] XTS		
Message Digests	[FIPS 180-3]	SHA1, SHA2 (224, 256, 384, 512)	4196
Keyed Hash	[FIPS 198] HMAC	SHA1, SHA2 (224, 256, 384, 512)	3448
Digital Signature and Asymmetric Key Generation	[FIPS 186-2] RSA	SigGen9.31(4096 with SHA-256, SHA-384 and SHA-512), SigVer9.31 (1024/1536/2048/3072/4096 with SHA-1, SHA-256, SHA-384 and SHA-512), SigGenPKCS1.5 (4096 with SHA-224, SHA-256, SHA-384 and SHA-512, SigGenPSS (4096 with SHA-224, SHA-256, SHA-384 and SHA-512, SigVerPKCS1.5 (1024/1536/2048/3072/4096 with SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512), SigVerPSS (1024/1536/2048/3072/4096 with SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512)	2788
	[FIPS 186-4] RSA	SigGen9.31 (2048/3072 with SHA-256, SHA-384 and SHA-512) SigGenPKCS1.5 (2048/3072 with SHA-224, SHA-256, SHA-384 and SHA-512), SigGenPSS (2048/3072 with SHA-224, SHA-256, SHA-384 and SHA-512)	2788
	[FIPS 186-4] DSA	PQG Gen, PQG Ver, Key Pair Gen, Sig Gen, Sig Ver (1024/2048/3072 with all SHA2 sizes)	1348

[3] For all DRBGs the "supported security strengths" is just the highest supported security strength per [SP800-90A] and [SP800-57]

Function	**Algorithm**	**Options**	**Cert #**
	[FIPS 186-2] ECDSA	PKG: CURVES (P224 P384 P521 K233 K283 K409 K571 B233 B283 B409 B571) PKV: CURVES (P192 P224 P256 P384 P521 K163 K233 K283 K409 K571 B163 B233 B283 B409 B571)	1349
	[FIPS 186-4] ECDSA	**PKG**: CURVES (P224 P256 P384 P521 K224 K256 K384 K521 B224 B256 B384 B521 ExtraRandomBits TestingCandidates) **PKV**: CURVES (ALLP ALLK ALLB) **SigGen**: CURVES (P224: (SHA224, 256, 384, 512) P256: (SHA224, 256, 384, 512) P384: (SHA224, 256, 384, 512) P521: (SHA224, 256, 384, 512) K233: (SHA224, 256, 384,512) K283: (SHA224, 256, 384, 512) K409:(SHA224, 256, 384, 512) K571: (SHA224, 256, 384, 512) B233: (SHA224, 256, 384, 512) B283: (SHA224, 256, 384, 512) B409: (SHA224, 256, 384, 512) B571: (SHA224, 256, 384, 512)) **SigVer**: CURVES (P192: (SHA1, 224, 256, 384, 512) P224: (SHA1,224, 256, 384, 512) P256: (SHA1, 224, 256, 384, 512) P384: (SHA1, 224, 256, 384, 512) P521: (SHA1, 224, 256, 384, 512) K163: (SHA1, 224, 256, 384, 512) K233: (SHA1, 224, 256, 384, 512) K283: (SHA1, 224, 256, 384, 512) K409: (SHA1, 224, 256, 384, 512) K571: (SHA1, 224, 256, 384, 512) B163: (SHA1, 224, 256, 384, 512) B233: (SHA1, 224, 256, 384, 512) B283: (SHA1, 224, 256, 384, 512) B409: (SHA1, 224, 256, 384, 512)	1349

Function	Algorithm	Options	Cert #
		B571: (SHA1, 224, 256, 384, 512))	
ECC CDH (KAS)	[SP 800-56A] (§5.7.1.2)	All NIST defined B, K and P curves except sizes 163 and 192	1712

Table 4a -- FIPS-Approved Cryptographic Functions

The Module supports only NIST defined curves for use with ECDSA and ECC CDH.

Category	Algorithm	Description
Key Agreement	EC DH	Non-compliant (untested) DH scheme using elliptic curve, supporting all NIST defined B, K and P curves. Key agreement is a service provided for calling process use, but is not used to establish keys into the Module.
Key Encryption, Decryption	RSA	The RSA algorithm may be used by the calling application for encryption or decryption of keys. No claim is made for SP800-56B compliance, and no CSPs are established into or exported out of the module using these services.

Table 4b -- Non-FIPS Approved But Allowed Cryptographic Functions

The Module implements the following services which are Non-Approved per the SP 800-131A transition:

Function	Algorithm	Options	Cert #
Random Number Generation; Symmetric key generation	[ANS X9.31] RNG	AES 128/192/256	
	[SP 800-90A] DRBG	Dual EC DRBG	
Digital Signature and	[FIPS 186-2] RSA	GenKey9.31, SigGen9.31, SigGenPKCS1.5, SigGenPSS (1024/1536 with all SHA sizes, 2048/3072/4096 with SHA-1) GenKey9.31 (2048/3072/4096 with all SHA2 sizes)	

Function	Algorithm	Options	Cert #
Asymmetric Key Generation	[FIPS 186-2] DSA	PQG Gen, Key Pair Gen, Sig Gen (1024 with all SHA sizes, 2048/3072 with SHA-1)	
	[FIPS 186-4] DSA	PQG Gen, Key Pair Gen, Sig Gen (1024 with all SHA sizes, 2048/3072 with SHA-1)	
	[FIPS 186-2] ECDSA	PKG: CURVES(P-192 K-163 B-163) SIG(gen): CURVES(P-192 P-224 P-256 P-384 P-521 K-163 K-233 K-283 K-409 K-571 B-163 B-233 B-283 B-409 B-571)	
	[FIPS 186-4] ECDSA	PKG: CURVES(P-192 K-163 B-163) SigGen: CURVES(P-192: (SHA-1, 224, 256, 384, 512) P-224:(SHA-1) P-256:(SHA-1) P-384:(SHA-1) P-521:(SHA-1) K-163: (SHA-1, 224, 256, 384, 512) K-233:(SHA-1) K-283:(SHA-1) K-409:(SHA-1) K-571:(SHA-1) B-163: (SHA-1, 224, 256, 384, 512) B-233:(SHA-1) B-283:(SHA-1) B-409:(SHA-1) B-571:(SHA-1))	
ECC CDH (CVL)	[SP 800-56A] (§5.7.1.2)	All NIST Recommended B, K and P curves sizes 163 and 192	

Table 4c -- FIPS Non-Approved Cryptographic Functions

These algorithms shall not be used when operating in the FIPS Approved mode of operation.

EC DH Key Agreement provides a maximum of 256 bits of security strength. RSA Key Wrapping provides a maximum of 256 bits of security strength.

The Module requires an initialization sequence (see IG 9.5): the calling application invokes FIPS_mode_set() , which returns a "1" for success and "0" for failure. If FIPS_mode_set() fails then all cryptographic services fail from then on. The application can test to see if FIPS mode has been successfully performed.

The Module is a cryptographic engine library, which can be used only in conjunction with additional software. Aside from the use of the NIST defined elliptic curves as trusted third party domain parameters, all other FIPS 186-4 assurances are outside the scope of the Module, and are the responsibility of the calling process.

4.1 Critical Security Parameters and Public Keys

All CSPs used by the Module are described in this section. All access to these CSPs by Module services are described in Section 4. The CSP names are generic, corresponding to API parameter data structures.

CSP Name	Description
RSA SGK	RSA (1024 to 16384 bits) signature generation key
RSA KDK	RSA (1024 to 16384 bits) key decryption (private key transport) key

DSA SGK	[FIPS 186-4] DSA (1024/2048/3072) signature generation key or [FIPS 186-2] DSA (1024) signature generation key
ECDSA SGK	ECDSA (All NIST defined B, K, and P curves) signature generation key
EC DH Private	EC DH (All NIST defined B, K, and P curves) private key agreement key.
AES EDK	AES (128/192/256) encrypt / decrypt key
AES CMAC	AES (128/192/256) CMAC generate / verify key
AES GCM	AES (128/192/256) encrypt / decrypt / generate / verify key
AES XTS	AES (256/512) XTS encrypt / decrypt key
TDES EDK	TDES (3-Key) encrypt / decrypt key
TDES CMAC	TDES (3-Key) CMAC generate / verify key
HMAC Key	Keyed hash key (160/224/256/384/512)
Hash DRBG CSPs	V (440/888 bits) and C (440/888 bits), entropy input (length dependent on security strength)
HMAC_DRBG CSPs	V (160/224/256/384/512 bits) and Key (160/224/256/384/512 bits), entropy input (length dependent on security strength)
CTR_DRBG CSPs	V (128 bits) and Key (AES 128/192/256), entropy input (length dependent on security strength)
CO-AD-Digest	Pre-calculated HMAC-SHA-1 digest used for Crypto Officer role authentication
User-AD-Digest	Pre-calculated HMAC-SHA-1 digest used for User role authentication

Table 4.1a – Critical Security Parameters

Authentication data is loaded into the module during the module build process, performed by an authorized operator (Crypto Officer), and otherwise cannot be accessed.

The module does not output intermediate key generation values.

CSP Name	Description
RSA SVK	RSA (1024 to 16384 bits) signature verification public key
RSA KEK	RSA (1024 to 16384 bits) key encryption (public key transport) key
DSA SVK	[FIPS 186-4] DSA (1024/2048/3072) signature verification key or [FIPS 186-2] DSA (1024) signature verification key
ECDSA SVK	ECDSA (All NIST defined B, K and P curves) signature verification key
EC DH Public	EC DH (All NIST defined B, K and P curves) public key agreement key.

Table 4.1b – Public Keys

For all CSPs and Public Keys:

Storage: RAM, associated to entities by memory location. The Module stores DRBG state values for the lifetime of the DRBG instance. The module uses CSPs passed in by the calling application on the stack. The Module does not store any CSP persistently (beyond the lifetime of an API call), with the exception of RNG and DRBG state values used for the Modules' default key generation service.
Generation: The Module implements SP 800-90A compliant DRBG services for creation of symmetric keys, and for generation of DSA and elliptic curve keys as shown in Table 4a. The calling application is responsible for storage of generated keys returned by the module.

Entry: All CSPs enter the Module's logical boundary in plaintext as API parameters, associated by memory location. However, none cross the physical boundary.
Output: The Module does not output CSPs, other than as explicit results of key generation services. However, none cross the physical boundary.
Destruction: Zeroization of sensitive data is performed automatically by API function calls for temporarily stored CSPs. In addition, the module provides functions to explicitly destroy CSPs related to random number generation services. The calling application is responsible for parameters passed in and out of the module.

Private and secret keys as well as seeds and entropy input are provided to the Module by the calling application, and are destroyed when released by the appropriate API function calls. Keys residing in internally allocated data structures (during the lifetime of an API call) can only be accessed using the Module defined API. The operating system protects memory and process space from unauthorized access. Only the calling application that creates or imports keys can use or export such keys. All API functions are executed by the invoking calling application in a non-overlapping sequence such that no two API functions will execute concurrently. An authorized application as user (Crypto-Officer and User) has access to all key data generated during the operation of the Module.

In the event Module power is lost and restored the calling application must ensure that any AES-GCM keys used for encryption or decryption are re-distributed.

Module users (the calling applications) shall use entropy sources that meet the security strength required for the random number generation mechanism: 128 bits for the as shown in [SP 800-90A] Table 2 (Hash_DRBG, HMAC_DRBG) and Table 3 (CTR_DRBG). This entropy is supplied by means of callback functions. Those functions must return an error if the minimum entropy strength cannot be met.

5 Roles, Authentication and Services

The Module implements the required User and Crypto Officer roles and requires authentication for those roles. Only one role may be active at a time and the Module does not allow concurrent operators. The User or Crypto Officer role is assumed by passing the appropriate password to the FIPS_module_mode_set() function. The password values may be specified at build time and must have a minimum length of 16 characters. Any attempt to authenticate with an invalid password will result in an immediate and permanent failure condition rendering the Module unable to enter the FIPS mode of operation, even with subsequent use of a correct password.

Authentication data is loaded into the Module during the Module build process, performed by the Crypto Officer, and otherwise cannot be accessed.

Since minimum password length is 16 characters, the probability of a random successful authentication attempt in one try is a maximum of $1/256^{16}$, or less than $1/10^{38}$. The Module permanently disables further authentication attempts after a single failure, so this probability is independent of time.

Both roles have access to all of the services provided by the Module.

- User Role (User): Loading the Module and calling any of the API functions.
- Crypto Officer Role (CO): Installation of the Module on the host computer system and calling of any API functions.

All services implemented by the Module are listed below, along with a description of service CSP access.

Service	Role	Description
Initialize	User, CO	Module initialization. Does not access CSPs.
Self-test	User, CO	Perform self tests (FIPS_selftest). Does not access CSPs.
Show status	User, CO	Functions that provide module status information: • Version (as unsigned long or const char *) • FIPS Mode (Boolean) Does not access CSPs.
Zeroize	User, CO	Functions that destroy CSPs: • fips_drbg_uninstantiate: for a given DRBG context, overwrites DRBG CSPs (Hash_DRBG CSPs, HMAC_DRBG CSPs, CTR_DRBG CSPs) All other services automatically overwrite CSPs stored in allocated memory. Stack cleanup is the responsibility of the calling application.
Random number generation	User, CO	Used for random number and symmetric key generation. • Seed or reseed an DRBG instance • Determine security strength of an DRBG instance • Obtain random data Uses and updates Hash_DRBG CSPs, HMAC_DRBG CSPs, CTR_DRBG CSPs.
Asymmetric key generation	User, CO	Used to generate DSA and ECDSA keys: DSA SGK, DSA SVK; ECDSA SGK, ECDSA SVK There is one supported entropy strength for each mechanism and algorithm type, the maximum specified in SP800-90A

Symmetric encrypt/decrypt	User, CO	Used to encrypt or decrypt data. Executes using AES EDK, TDES EDK (passed in by the calling process).
Symmetric digest	User, CO	Used to generate or verify data integrity with CMAC. Executes using AES CMAC, TDES, CMAC (passed in by the calling process).
Message digest	User, CO	Used to generate a SHA-1 or SHA-2 message digest. Does not access CSPs.
Keyed Hash	User, CO	Used to generate or verify data integrity with HMAC. Executes using HMAC Key (passed in by the calling process).
Key transport[4]	User, CO	Used to encrypt or decrypt a key value on behalf of the calling process (does not establish keys into the module). Executes using RSA KDK, RSA KEK (passed in by the calling process).
Key agreement	User, CO	Used to perform key agreement primitives on behalf of the calling process (does not establish keys into the module). Executes using EC DH Private, EC DH Public (passed in by the calling process).
Digital signature	User, CO	Used to generate or verify RSA, DSA or ECDSA digital signatures. Executes using RSA SGK, RSA SVK; DSA SGK, DSA SVK; ECDSA SGK, ECDSA SVK (passed in by the calling process).
Utility	User, CO	Miscellaneous helper functions. Does not access CSPs.

Table 5 -- Services and CSP Access

6 Self-test

The Module performs the self-tests listed below on invocation of Initialize or Self-test.

Algorithm	Type	Test Attributes
Software integrity	KAT	HMAC-SHA1
HMAC	KAT	One KAT per SHA1, SHA224, SHA256, SHA384 and SHA512 Per IG 9.3, this testing covers SHA POST requirements.
AES	KAT	Separate encrypt and decrypt, ECB mode, 128 bit key length

[4] "Key transport" can refer to a) moving keys in and out of the module or b) the use of keys by an external application. The latter definition is the one that applies to the Gallagher OpenSSL Cryptographic Module.

AES CCM	KAT	Separate encrypt and decrypt, 192 key length
AES GCM	KAT	Separate encrypt and decrypt, 256 key length
XTS-AES	KAT	128, 256 bit key sizes to support either the 256-bit key size (for XTS-AES-128) or the 512-bit key size (for XTS-AES-256)
AES CMAC	KAT	Sign and verify CBC mode, 128, 192, 256 key lengths
TDES	KAT	Separate encrypt and decrypt, ECB mode, 3-Key
TDES CMAC	KAT	CMAC generate and verify, CBC mode, 3-Key
RSA	KAT	Sign and verify using 2048 bit key, SHA-256, PKCS#1
DSA	PCT	Sign and verify using 2048 bit key, SHA-384
DRBG	KAT	CTR_DRBG: AES, 256 bit with and without derivation function HASH_DRBG: SHA256 HMAC_DRBG: SHA256 Dual_EC_DRBG: P-256 and SHA256
ECDSA	PCT	Keygen, sign, verify using P-224, K-233 and SHA512. The K-233 self-test is not performed for operational environments that support prime curve only (see Table 2).
ECC CDH	KAT	Shared secret calculation per SP 800-56A §5.7.1.2, IG 9.6

Table 6a - Power On Self Tests (KAT = Known answer test; PCT = Pairwise consistency test)

The Module is installed using one of the set of instructions in Appendix A, as appropriate for the target system. The HMAC-SHA-1 of the Module distribution file as tested by the CMT Laboratory and listed in Appendix A is verified during installation of the Module file as described in Appendix A.

The FIPS_mode_set()[5] function performs all power-up self-tests listed above with no operator intervention required, returning a "1" if all power-up self-tests succeed, and a "0" otherwise. If any component of the power-up self-test fails an internal flag is set to prevent subsequent invocation of any cryptographic function calls. The module will only enter the FIPS Approved mode if the module is reloaded and the call to FIPS_mode_set()[5] succeeds.

The power-up self-tests may also be performed on-demand by calling FIPS_selftest(), which returns a "1" for success and "0" for failure. Interpretation of this return code is the responsibility of the calling application.

[5] `FIPS_mode_set()` calls Module function `FIPS_module_mode_se()`

The Module also implements the following conditional tests:

Algorithm	Test
DRBG	Tested as required by [SP800-90] Section 11
DRBG	FIPS 140-2 continuous test for stuck fault
DSA	Pairwise consistency test on each generation of a key pair
ECDSA	Pairwise consistency test on each generation of a key pair

Table 6b - Conditional Tests

In the event of a DRBG self-test failure the calling application must uninstantiate and re-instantiate the DRBG per the requirements of [SP 800-90A]; this is not something the Module can do itself.

Pairwise consistency tests are performed for both possible modes of use, e.g. Sign/Verify and Encrypt/Decrypt.

The Module supports two operational environment configurations for elliptic curve: NIST prime curves only (listed in Table 2 with the EC column marked "P") and all NIST defined curves (listed in Table 2 with the EC column marked "BKP").

7 Operational Environment

The tested operating systems segregate user processes into separate process spaces. Each process space is logically separated from all other processes by the operating system software and hardware. The Module functions entirely within the process space of the calling application, and implicitly satisfies the FIPS 140-2 requirement for a single user mode of operation.

8 Mitigation of other Attacks

The module is not designed to mitigate against attacks which are outside of the scope of FIPS 140-2.

Appendix A Installation and Usage Guidance

The tested Gallagher module is built in a cross-compilation environment. This means the system used for building the module is different from the tested target system that the module will run on. The details of both systems are:

Build system operating system:	Ubuntu Linux 16.04 LTS running on Intel® Core™ i7
Build system toolchain:	crosstool-NG GCC 7.2
Target system:	Linux 4.9 running on ARM926EJ-S (ARMv5TEJ)

Preparation for installation

Before the module can be built, the cross-compile toolchain needs to be present and configured on the build system. Example configuration looks like this:

```
export INSTALLDIR=~/dev/tools/arm-unknown-linux-gnueabi
export PATH=$INSTALLDIR/bin:$PATH
export MACHINE=armv4
export RELEASE=4.9.0
export SYSTEM=Linux
export ARCH=arm
export CROSS_COMPILE="arm-unknown-linux-gnueabi-"
export HOSTCC=gcc
```

Build and install instructions

1. Download and copy the distribution file to the build system.

   This file can be downloaded: from https://www.openssl.org/source/openssl-fips-2.0.12.tar.gz

2. Verify the HMAC-SHA-1 digest of the distribution file; see Appendix B. An independently acquired FIPS 140-2 validated implemention of SHA-1 HMAC must be used for this digest verification. Note that this verification can be performed on any convenient system and not necessarily on the specific build or target system.

3. Unpack the distribution

   ```
   tar -zxvf openssl-fips-2.0.12.tar.gz
   cd openssl-fips-2.0.12
   ```

4. Execute the following commands:

   ```
   ./config
   make
   make install
   ```

5. The resulting *fipscanister.o* or *fipscanister.lib* file is now available for use.

6. The calling application enables FIPS mode by calling the FIPS_mode_set()[6] function.

Linking the Runtime Executable Application

Note that applications interfacing with the FIPS Object Module are outside of the cryptographic boundary. When linking the application with the FIPS Object Module two steps are necessary:

1. The HMAC-SHA-1 digest of the FIPS Object Module file must be calculated and verified against the installed digest to ensure the integrity of the FIPS object module.

2. A HMAC-SHA1 digest of the FIPS Object Module must be generated and embedded in the FIPS Object Module for use by the FIPS_mode_set()[6] function at runtime initialization.

The fips_standalone_sha1 command can be used to perform the verification of the FIPS Object Module and to generate the new HMAC-SHA-1 digest for the runtime executable application. Failure to embed the digest in the executable object will prevent initialization of FIPS mode.

At runtime the FIPS_mode_set()[6] function compares the embedded HMAC-SHA-1 digest with a digest generated from the FIPS Object Module object code. This digest is the final link in the chain of validation from the original source to the runtime executable application file.

Appendix B Controlled Distribution File Fingerprint

The *Gallagher OpenSSL Cryptographic Module v2.0.12* consists of the FIPS Object Module (the *fipscanister.o* or *fipscanister.lib* contiguous unit of binary object code) generated from the specific source files.

The source files are in the specific special OpenSSL distribution *openssl-fips-2.0.12.tar.gz* with HMAC-SHA-1 digest of

```
86ec30179f1bfb2edde4ababf0fb519ba7380b69
```

located at http://www.openssl.org/source/openssl-fips-2.0.12.tar.gz.

The openssl command from a version of OpenSSL that incorporates a previously validated version of the module may be used:

```
openssl sha1 -hmac etaonrishdlcupfm openssl-fips-2.0.12.tar.gz
```

The set of files specified in this tar file constitutes the complete set of source files of this module. There shall be no additions, deletions, or alterations of this set as used during module build. The OpenSSL distribution tar file (and patch file if used) shall be verified using the above HMAC-SHA-1 digest(s).

The arbitrary 16 byte key of:

[6] `FIPS_mode_set()` calls the Module function `FIPS_module_mode_set()`

```
65 74 61 6f 6e 72 69 73 68 64 6c 63 75 70 66 6d
```

(equivalent to the ASCII string "etaonrishdlcupfm") is used to generate the HMAC-SHA-1 value for the FIPS Object Module integrity check.

Appendix C Compilers

This appendix lists the specific compilers used to generate the Module for the respective Operational Environments. Note this list does not imply that use of the Module is restricted to only the listed compiler versions, only that the use of other versions has not been confirmed to produce a correct result.

#	Operational Environment	Compiler
1	Linux 4.9	gcc 7.2.0

Table C - Compilers