Cyphre Crypto Core

Software Version: OpenSSL-FIPS-Cyphre-v1.0

Hardware Version: NXP QorIQ P4080 Rev3

FIPS 140-2 Non-Proprietary Security Policy

Document Version 1.2

Last update: 2020-Jan-06

Prepared by:

atsec information security corporation

9130 Jollyville Road, Suite 260

Austin, TX 78759

www.atsec.com

Table of Contents

List of Tables

List of Figures

Copyrights and Trademarks

Cyphre Crypto Core are registered trademarks of Cyphre Security Solutions, LLC.

All other product and company names are trademarks or registered trademarks of their respective holders.

1 Introduction

This document is the non-proprietary FIPS 140-2 Security Policy for software version OpenSSL-FIPS-Cyphre-v1.0 and hardware version NXP QorIQ P4080 Rev3 of the Cyphre Crypto Core module. It contains the security rules under which the module must be operated and describes how this module meets the requirements as specified in FIPS PUB 140-2 (Federal Information Processing Standards Publication 140-2) for a Security Level 1 module.

1.1 Purpose of the Security Policy

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

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

1.2 Target Audience

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

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

2 Cryptographic Module Specification

The sections in this document describe the cryptographic module and how it conforms to the FIPS 140-2 specification in each of the required areas.

2.1 Module Overview

Cyphre Crypto Core (hereafter also referred to as the “module”) is a Software-Hybrid module offering solutions for network security, encryption key control, and enterprise data.

One of Cyphre's solutions utilizing the module, CyphreLink, is a network encryption solution that provides protection for sensitive and proprietary information transiting any network. CyphreLink interoperates with RigNet’s global MPLS backbone to strengthen the movement of data across cost-advantaged open networks. With CyphreLink, enterprises can be ensured that the secure connection across satellite, fixed, or wireless networks can be done with greater flexibility and agility than traditional connections. CyphreLink is easily incorporated into an enterprise’s existing data protection technologies. By serving as a unifying management solution, CyphreLink offers hardened security that reduces man-in-the-middle attacks and unauthorized eavesdropping, while expanding the abilities of an organization to leverage virtually any network efficiently and cost-effectively.

The module is delivered as a hardware component and a software component. The software component is composed of the binaries of libraries from OpenSSL version 1.0.2g, containing the software implementations of the cryptographic algorithms, and CAAM (Cryptographic Accelerator and Assurance Module) drivers that control the hardware component. The hardware portion is composed of the NXP QorIQ P4080 Rev 3 processor. The hardware implementations of the cryptographic algorithms in the NXP QorIQ P4080 Rev 3 processor are accessed through the OpenSSL API, and self-tests are also provided by OpenSSL. The underlying modifiable operating system, under which the module runs, is Ubuntu Linux version 16.04 LTS, kernel version 4.16.18.

The cryptographic logical boundary consists of the hardware cryptographic implementations and the NXP Random Number Generator (NXP RNG) within the NXP QorIQ P4080 Rev 3 processor and all object files included in the library, as well as the CAAM drivers to access the cryptographic algorithms hardware implementations supported by the module in the processor.

The cryptographic module system block diagram in Figure 1 represents the module as a system and its interfaces with the operational environment. The logical boundary is indicated by the light orange blocks, and the red dotted line marks the physical boundary. Blocks of other colors do not belong to the logical boundary.

Table 1 summarizes the components of the cryptographic module. It is noted that the figure represents some of the algorithms within the OpenSSL Library boundary, not the exhaustive list of supported algorithms by this module. The exhaustive list of algorithms is presented in Section 4.

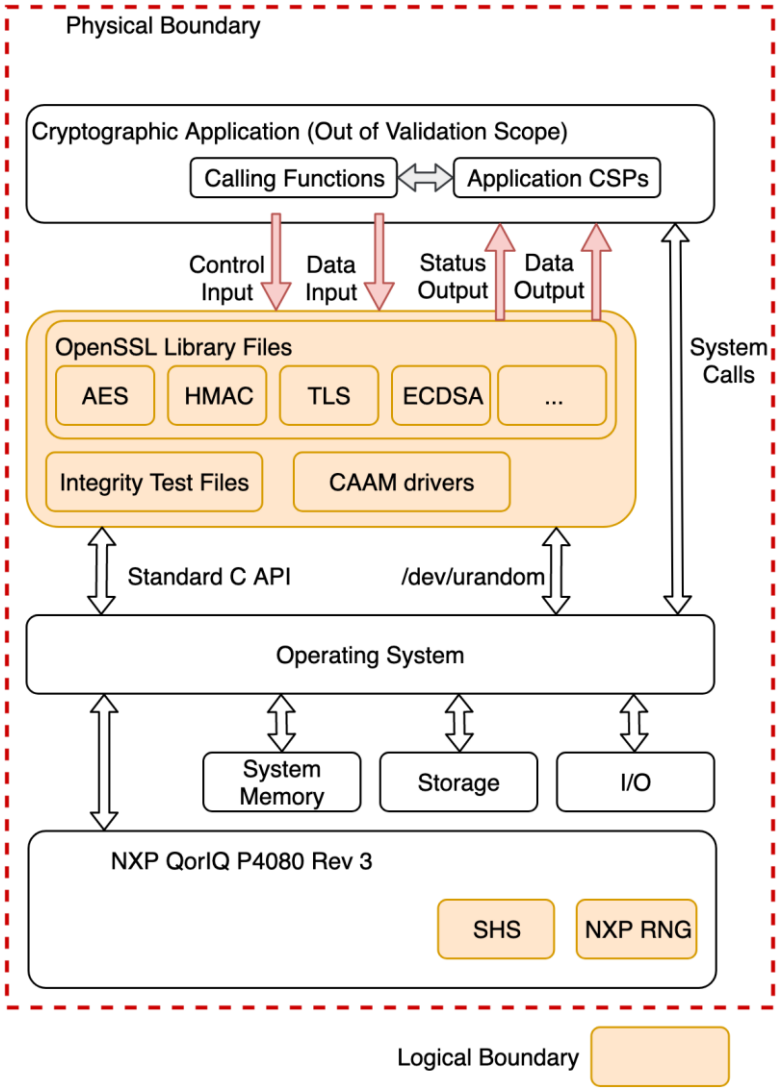
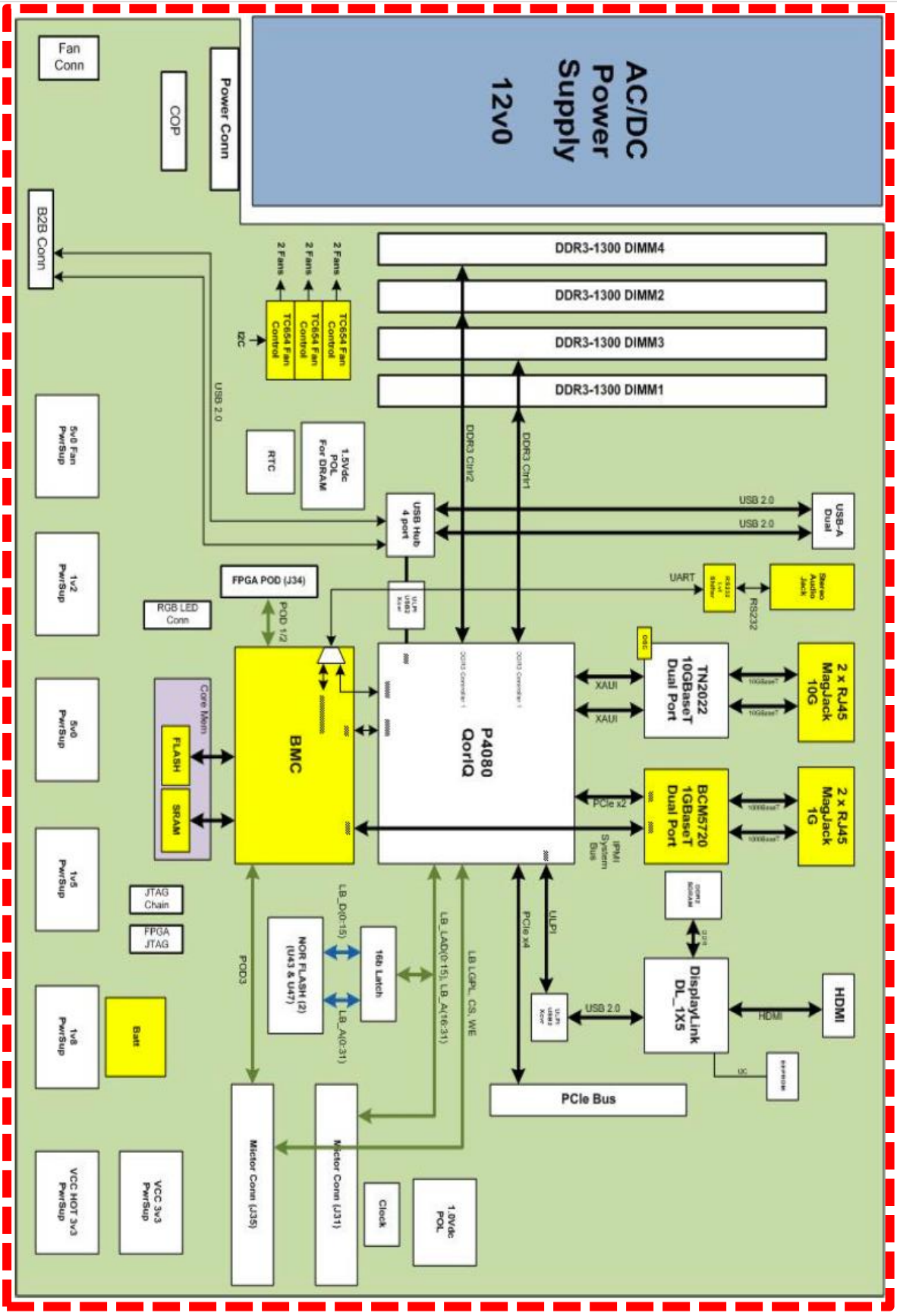


Figure 1: Cryptographic module system block diagram depicting the module and its cryptographic boundaries.

The user application, the cryptographic module itself, and the underlying modifiable Linux operating system run on the NXP QorIQ P4080 Rev 3 processor. The module is typically installed on a security appliance that is a general-purpose computer. The security appliance will contain the processor itself, motherboard, memory, storage, power interface and several communication interfaces (e.g., RS-232, USB, Ethernet). The whole physical enclosure of the security appliance platform constitutes the physical boundary of the module. Figure 2 depicts in (a) the main hardware components of the target hardware platform and the red dotted line shows the physical boundary of the cryptographic module. The different colors inside the figure are provided as artistic representation only. Figure 2 (b) brings a picture of the NXP processor.



(a)

(b)

Figure 2: (a) Hardware block diagram, wherein the physical boundary is marked in a red dotted line; (b) a picture the NXP processor.

Table 1: Components of the Cyphre Crypto Core.

Component	Description
NXP QorIQ P4080 Rev 3	Processor containing hardware cryptographic engines and the NXP RNG.
libssl.so.1.0.0.sha1	Integrity test file containing the HMAC value for the libssl shared library binary.
libssl.so.1.0.0	OpenSSL-based shared library containing support for TLS.
libcrypto.so.1.0.0	Shared Library for cryptographic algorithm routines and API for access to hardware cryptographic engines.
libcrypto.so.1.0.0.sha1	Integrity test file containing the HMAC value for the libcrypto shared library binary.
cryptodev.ko	CAAM driver module for the NXP QorIQ P4080 Rev 3 cryptographic engines.
caam_jr.ko	CAAM driver module for the NXP QorIQ P4080 Rev 3 cryptographic engines.
caam_pkc.ko	CAAM driver module for the NXP QorIQ P4080 Rev 3 cryptographic engines.
caam_prf.ko	CAAM driver module for the NXP QorIQ P4080 Rev 3 cryptographic engines.
caam.ko	CAAM driver module for the NXP QorIQ P4080 Rev 3 cryptographic engines.
caamalg_desc.ko	CAAM driver module for the NXP QorIQ P4080 Rev 3 cryptographic engines.
caamalg_qi.ko	CAAM driver module for the NXP QorIQ P4080 Rev 3 cryptographic engines.
caamalg.ko	CAAM driver module for the NXP QorIQ P4080 Rev 3 cryptographic engines.
caamhash.ko	CAAM driver module for the NXP QorIQ P4080 Rev 3 cryptographic engines.

Component	Description
caamrng.ko	CAAM driver module for the NXP QorIQ P4080 Rev 3 cryptographic engines.

2.2 FIPS 140-2 Validation

For the purpose of the FIPS 140-2 validation, the module is defined as a Software-Hybrid, Multi-chip standalone cryptographic module validated at overall Security Level 1. Table 2 shows the security level claimed for each of the eleven sections that comprise the FIPS 140-2 standard.

Table 2: Security levels.

FIPS 140-2 Section		Security Level
1	Cryptographic Module Specification	1
2	Cryptographic Module Ports and Interfaces	1
3	Roles, Services and Authentication	1
4	Finite State Model	1
5	Physical Security	1
6	Operational Environment	1
7	Cryptographic Key Management	1
8	EMI/EMC	1
9	Self-Tests	1
10	Design Assurance	1
11	Mitigation of Other Attacks	N/A
Overall Level		1

The module has been tested on the platform described in Table 3.

Table 3: Tested operational environment.

Device	Processor	Operational Environment
Cyphre BlackTIE 1 Security Appliance	NXP QorIQ P4080 rev 3	Ubuntu 16.04 LTS, kernel 4.16.18

2.3 Modes of operation

The module supports two modes of operation.

- In "**FIPS mode**" (the Approved mode of operation), only approved or allowed security functions with sufficient security strength can be used.

- In "**non-FIPS mode**" (the non-Approved mode of operation), only non-approved security functions can be used.

The module enters the FIPS mode of operation after the power-on self-tests (POST) succeed. Once the module is operational, the mode of operation is implicitly assumed depending on the security function invoked and the security strength of the cryptographic keys/curves chosen for the function or service.

If the POST fails, the module enters the error state (see Section 9).

Critical security parameters used or stored in FIPS mode are not used in non-FIPS mode, and vice versa.

3 Cryptographic Module Ports and Interfaces

As a Software-Hybrid cryptographic module, the operator can only interact with the module through the API provided by the OpenSSL library, both to the software and hardware implementations of the cryptographic algorithms. Thus, the physical ports within the physical boundary are interpreted to be the physical ports of the hardware platform on which the module runs, and are directed through the logical interfaces provided by the software.

The logical interfaces are the API through which applications request services and receive output data through return values or modified data referenced by pointers. Table 4 summarizes the four logical interfaces and the power input.

Table 4: Ports and interfaces.

Logical Interface	Description
Data Input	API input parameters for data.
Data Output	API output parameters for data.
Control Input	API function calls, API input parameters for control.
Status Output	API return codes, API output parameters for status.
Power Input	Not applicable for the software portion of the Software-Hybrid module. The hardware portion of the module receives power from the circuitry in which the module is installed.

4 Roles, Services and Authentication

4.1 Roles

The module supports the following roles:

- **User role**: performs all services (in both FIPS mode and non-FIPS mode of operation), except module installation and configuration. This role is assumed by the calling application accessing the module.
- **Crypto Officer role**: performs module installation and configuration.

The User and Crypto Officer roles are implicitly assumed depending on the service requested.

4.2 Services

The module provides services to calling applications that assume the user role, and human users assuming the Crypto Officer role. Table 5 and Table 6 depict all services, which are described with more detail in the user documentation.

The tables use the following convention when specifying the access permissions that the module has for each CSP or key.

- **Create**: the calling application can create a new CSP.
- **Read**: the calling application can read the CSP.
- **Update**: the calling application can write a new value to a pre-existing CSP.
- **Zeroize**: the calling application can zeroize the CSP.
- **n/a**: the calling application does not access any CSP or key during its operation.

4.2.1 Services in FIPS Mode of Operation

Table 5 shows the Approved services and the non-Approved but allowed services in FIPS mode of operation, the cryptographic algorithms supported for each service, the roles that can perform each service, and the keys and other Critical Security Parameters (CSPs) involved and how they are accessed. Details about the algorithms supported by the module are found in Section 4.3.

Table 5: Services in FIPS mode of operation.

Service	Algorithms	Role	Keys/CSPs	Access
Cryptographic Services				
Symmetric encryption and decryption	AES	User	AES key	Read
ECDSA digital signature generation and verification	ECDSA, DRBG	User	ECDSA public/private key	Read
ECDSA domain parameter generation and verification	ECDSA, DRBG	User	None	n/a
Message digest	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	User	None	n/a

Service	Algorithms	Role	Keys/CSPs	Access
Message authentication code (MAC)	HMAC (with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512)	User	HMAC key	Read
	AES-GMAC with AES-128, AES-192, AES-256	User	AES key	Read
Random Number Generation	CTR_DRBG with AES-128, AES-192, AES-256 Hash_DRBG with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 HMAC_DRBG with HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	User	Entropy input string, seed, internal state (V, C, key)	Create, Read, Update
Key generation	ECDSA, DRBG	User	ECDSA public/private key	Create
Key derivation	KDF in TLS v1.2	User	Pre-master secret, master secret, derived keys (AES, HMAC), KDF internal state	Create, Read, Update
EC Diffie-Hellman Key Agreement	KAS ECC	User	EC Diffie-Hellman public/private keys	Create, Read
Network Protocol Services				
Transport Layer Security (TLS) network protocol v1.2	Cipher suites that use exclusively algorithms and key sizes allowed in the FIPS-approved mode	User	AES key HMAC key Pre-master secret Master secret EC Diffie-Hellman public/private keys ECDSA public/private keys	Create, Read, Update
Other FIPS-related Services				
Show status	n/a	User	None	n/a
Zeroization	n/a	User	All keys and CSPs	Zeroize
Self-Tests	AES, SHS, HMAC, ECDSA, DRBG, EC Diffie-Hellman	User	None	n/a
Module installation	n/a	Crypto Officer	None	n/a

Service	Algorithms	Role	Keys/CSPs	Access
Module configuration	n/a	Crypto Officer	None	n/a

4.2.2 Services in the Non-FIPS Mode of Operation

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

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

Service	Algorithms/Key sizes	Role	Keys	Access
Symmetric encryption and decryption	DES, Triple-DES, AES-XCBC, AES-XTS, Blowfish, Camellia, CAST, IDEA, RC2, RC4, RC5, SEED.	User	Symmetric keys	Read
Key generation	RSA, DSA; ECDSA using keys/curves not listed in Table 7.	User	Public and private keys	Create
Key derivation	KDF in TLS v1.0, v1.1	User	Pre-master secret, master secret, derived keys (AES, Triple-DES, HMAC, and other algorithms in Table 9), KDF internal state	Create, Read, Update
Digital Signature Generation and Verification	RSA, DSA; ECDSA using keys/curves not listed in Table 7 and/or using SHA-1	User	Public and private keys	Read
Message digest	MD2, MD4, MD5, MDC-2, RIPEMD160	User	None	n/a
Message Authentication Code (MAC)	Triple-DES CMAC, AES-CMAC, HMAC-MD5; HMAC with SHS using keys not listed in Table 7.	User	MAC keys	Read
Key establishment	Diffie-Hellman, RSA, EC J-PAKE; EC Diffie-Hellman using keys/curves not listed in Table 7.	User	Public and private keys	Create, Read

Service	Algorithms/Key sizes	Role	Keys	Access
Transport Layer Security (TLS) network protocol v1.0, v1.1	Any cipher suites.	User	AES key Triple-DES key HMAC key Pre-master secret Master secret Diffie-Hellman public/private keys EC Diffie-Hellman public/private keys RSA, DSA, ECDSA public/private keys	Create, Read, Update
Transport Layer Security (TLS) network protocol v1.2 (non-approved mode)	Cipher suites that use algorithms and key sizes not allowed by this Security Policy. These include Camellia, IDEA, RC4, SRP, etc.	User	AES key Triple-DES key HMAC key Pre-master secret Master secret Diffie-Hellman public/private keys EC Diffie-Hellman public/private keys RSA, DSA, ECDSA public/private keys	Create, Read, Update
Key wrapping	AES-KW, AES-KWP, Triple-DES-KW, Triple-DES-KWP, RSA.	User	RSA public/private key AES, Triple-DES symmetric key	Read
SSLeay Pseudo Random Number Generator (PRNG)	Using the SSLeay PRNG.	User	PRNG internal state.	Create, Read, Update

4.3 Algorithms

4.3.1 FIPS-Approved and Allowed Algorithms

The algorithms implemented in the module approved to be used in FIPS mode of operation are tested and validated by the CAVP. The module provides hardware implementations and software implementations for cryptographic algorithms, as indicated in the CAVP certificates. No parts of the Transport Layer Security (TLS) protocol, other than the key derivation function, have been tested by the CAVP or by the CMVP.

Table 7 shows the cryptographic algorithms that are approved and non-approved-but-allowed in FIPS mode of operation, including the algorithm name, supported standards, available modes and key sizes, and usage. We note that the module does not use all the algorithms contained in the certificates referenced in the table.

Table 7: FIPS-approved cryptographic algorithms.

Algorithm, CAVP Certificate	Standard	Mode/Method	Key size	Use
AES Cert. #C609	[FIPS197] [SP800-38A]	ECB, CBC, CFB, CTR, OFB	128, 192 and 256 bits	Data Encryption and Decryption
	[SP800-38C]	CCM	128, 192 and 256 bits	Data Encryption and Decryption
	[SP800-38D]	GCM	128, 192 and 256 bits	Data Encryption and Decryption
	[SP800-38D]	GMAC	128, 192 and 256 bits	Message Authentication Code
DRBG Cert. #C609 Prerequisites: Cert. #C609 (AES) Cert. SHS #2109 Cert. #C609 (HMAC)	[SP800-90A]	HASH_DRBG SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 with and without PR CTR_DRBG AES-128, AES-192, AES-256 with and without DF, with and without PR HMAC_DRBG with SHA1, SHA-224, SHA-256, SHA-384, SHA-512 with and without PR	n/a	Random Number Generator
ECDSA Cert. #C609 Prerequisites: Cert. SHS #2109 Cert. #C609 (DRBG)	[FIPS186-4]	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	P-256, P-384, P-521	Signature Verification
		SHA-224, SHA-256, SHA-384, SHA-512	P-256, P-384, P-521	Signature Generation
		Testing Candidates	P-224, P-256, P-384	Key Generation
			P-256, P-384, P-521	Public Key Verification
HMAC Cert. #C609 Prerequisites: Cert. SHS #2109	[FIPS198-1]	HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	112 bits or greater	Message Authentication Code

Algorithm, CAVP Certificate	Standard	Mode/Method	Key size	Use
KDF in TLS v1.2 Component (CVL) Cert. #C609 Prerequisites: Cert. #C609 (HMAC) Cert. SHS #2109	[SP800-135]	HMAC with SHA-256, SHA-384, SHA-512		Key Derivation
SHS Cert. SHS #2109	[FIPS180-4]	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	n/a	Message Digest
KAS ECC Component (CVL) Cert. #C609 Prerequisites: Cert. #C609 (DRBG) Cert. #C609 (ECDSA) Cert. SHS #2109	[SP800-56A]	Partial Public Key Validation Ephemeral Unified SHA-256, SHA-384	P-256, P-384	EC Diffie-Hellman Key Agreement
KTS (algorithms with Certs. in this table)	[SP800-38F]	See Section 7.4	See Section 7.4	Key Wrapping

Table 8 shows the cryptographic algorithms that are not approved but allowed in FIPS mode of operation.

Table 8: FIPS-allowed cryptographic algorithms.

Algorithm	Standard	Mode/Method	Key size	Use
EC Diffie-Hellman	[FIPS140-2_IG D.8]	Ephemeral Unified SHA-256, SHA-384 Components: TLS KDF and KAS-ECC (Table 7).	P-256, P-384	Key agreement, key establishment methodology provides 128 or 192 bits of encryption strength

Algorithm	Standard	Mode/Method	Key size	Use
NDRNG	[FIPS140-2_IG 7.15]	n/a	n/a	The module obtains the entropy data from the NDRNG to seed and reseed the DRBG.

4.3.2 Algorithms not Allowed in the FIPS Mode of Operation

Table 9 lists the cryptographic algorithms implemented in the module that are not allowed in FIPS mode of operation, including the algorithm name and the reason for being forbidden. Using any of these algorithms will implicitly turn the module in Non-FIPS mode of operation.

Table 6 can also be used as a reference to algorithms contained in services in the Non-FIPS mode of operation. Such algorithms are not allowed in the FIPS mode of Operation.

Table 9: Non-approved cryptographic algorithms.

Algorithm	Reason
DES, AES-XCBC, Blowfish, Camellia, CAST, IDEA, RC2, RC4, RC5, SEED, EC J-PAKE, MD2, MD4, MDC-2, RIPEMD160	Non FIPS-Approved algorithms.
Two-key Triple-DES	Not allowed per [SP800-131A].
Triple-DES (any mode)	Algorithm was not tested with CAVS or ACVP.
AES-XTS	Algorithm was not tested with CAVS or ACVP.
AES-KW and AES-KWP	Algorithm was not tested with CAVS or ACVP.
Diffie-Hellman	Algorithm was not tested with CAVS or ACVP.
EC Diffie-Hellman	Key agreement using curves not listed in Table 7.
ECDSA	Key generation/verification and signature generation/verification using curves not listed in Table 7.
DSA	Algorithm was not tested with CAVS or ACVP.
MD5	Non FIPS-Approved algorithm.
SHA-1	Not allowed for Digital Signature Generation per [SP800-131A].
HMAC with key size less than 112 bits.	Not allowed key size for Message Authentication Code per [SP800-131A].
RSA of any key size.	Algorithm was not tested with CAVS or ACVP.
SSLeay Pseudo Random Number Generator (PRNG)	Non-FIPS approved algorithm.

4.4 Operator Authentication

The module does not implement user authentication. The role of the user is implicitly assumed based on the service requested.

5 Physical Security

The NXP QorIQ P4080 Rev 3 processor, the hardware component of the module, is a single chip with a production-grade enclosure and hence conforms to the Level 1 requirements for physical security.

This Security Policy does not claim any physical security for the software portion of the module.

6 Operational Environment

6.1 Applicability

The module operates in a modifiable operational environment per FIPS 140-2 Security Level 1 specifications. The module runs on the Linux Ubuntu version 16.04 LTS operating system and uses OpenSSL version 1.0.2g for software implementations of cryptographic algorithms and as API to the hardware implementation of cryptographic algorithms. OpenSSL is also used to provide the self-tests. The module executes on the hardware specified in Section 2.2.

6.2 Policy

The application that requests cryptographic services is the single user of the cryptographic module. Concurrent operators are explicitly excluded by the operating system.

7 Cryptographic Key Management

Table 10 summarizes the secret keys and CSPs that are used by the cryptographic services implemented in the module. The table describes the use of each key/CSP and, as applicable, how they are generated or established, their method of entry and output of the module, their storage location, and details of key length when useful. Note that all lengths and types obey the set of approved algorithms, key lengths and applicable curves listed in Table 7.

The table shows the keys and CSPs under a generic context, and those keys used under the context of the TLS network protocol, as indicated by the subtitle “TLS Network Protocol”. For instance, AES keys used for encryption/decryption outside the TLS protocol are provided by the calling application and passed via API. Inside the TLS protocol, these AES keys are always derived using the TLS KDF in the approved mode. Moreover, the AES mode for which these derived AES keys and their length are defined by the TLS ciphersuite in the approved mode.

The method for zeroizing the key/CSP is explained in Section 7.7.

All key and CSP storage is done in plaintext.

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

Name	Generation, Establishment	Entry/Exit	Type	Storage	Usage and Note
AES key	Provided by the calling application.	Entered via API input parameters in plaintext. No exit.	AES key, all modes per Table 7. Length: 128, 192, 256 bits.	RAM	File/data encryption/decryption. MAC generation and verification for GMAC. Key wrapping with authenticated encryption (see Section 7.4).
HMAC key	Provided by the calling application.	Entered via API input parameters in plaintext. No exit.	HMAC keys of length > 112 bits.	RAM	MAC generation and verification.
ECDSA public and private key	Key pairs are generated (using curves in Table 7) using FIPS 186-4 key generation method, and the random value used is generated using the SP800-90A DRBG.	The key is passed into the module via API input parameters in plaintext. The key is passed out of the module via output parameters in plaintext.	ECDSA keys for curves P-224, P-256, P-384, P-521	RAM	ECDSA signature generation and verification.
Entropy input string	Obtained from the NDRNG.	N/A	384 bits	RAM	Entropy input used to construct the seed for the SP800-90A DRBG.
DRBG internal state (V, C, key)	Generated by the DRBG.	N/A		RAM	Generate random bit strings.
TLS Network Protocol					

Name	Generation, Establishment	Entry/Exit	Type	Storage	Usage and Note
AES derived key	Generated internally by the module (from the ([SP800-135] TLS KDF) during the establishment of the TLS tunnel).	N/A	AES key (CBC, CCM, GCM) with length 128 or 256 bits (defined by ciphersuite).	RAM	Data encryption/decryption. Key wrapping (see Section 7.4).
HMAC derived key	Generated internally by the module (from the ([SP800-135] TLS KDF) during the establishment of the TLS.	N/A	HMAC key with length defined by ciphersuite.	RAM	MAC generation and verification.
Pre-master secret	Generated internally by the module (from the EC Diffie-Hellman key agreement) during the establishment of the TLS tunnel. Received by the module via API if module is acting as a TLS server.	Entry: received by module via API parameters if acting as TLS server; otherwise, no entry. Output: to the TLS service via API parameters if generated by the module as a TLS client; otherwise, no exit.	Length defined per ciphersuite.	RAM	Establishment of TLS session keys.
Master secret	Generated internally by the module (from the ([SP800-135] TLS KDF) during the establishment of the TLS tunnel.	N/A	384 bits.	RAM	Establishment of TLS session keys.
EC Diffie-Hellman public and private key	Generated internally by the module during the establishment of the TLS tunnel. Generation uses FIPS 186-4 key generation method, and the random value used is generated using the SP800-90A DRBG. Provided by the calling application in case of ECDH static.	Entry: provided by the calling application if ECDH static. Otherwise, no entry. No exit.	Curves P-256, P-384	RAM	Key agreement.
TLS KDF internal state	SP800-135 TLS KDF	N/A		RAM	Values of the TLS KDF internal state used in TLS tunnel establishment

The following sections describe how keys and CSPs are managed during its life cycle, including the zeroization methods.

7.1 Random Number Generation

The module employs a Deterministic Random Bit Generator (DRBG) based on [SP800-90A] for the creation of key components of asymmetric keys, signature generation services, and server and client random numbers for the TLS protocol. In addition, the module provides calling applications with a Random Number Generation service.

The DRBG supports the Hash_DRBG (with and without prediction resistance), HMAC_DRBG (with and without prediction resistance) and CTR_DRBG (with and without derivation function, with and without prediction resistance) mechanisms. The DRBG is initialized during module initialization; the module loads by default the DRBG using the CTR_DRBG mechanism with AES-256 and derivation function without prediction resistance. A different DRBG mechanism can be chosen through an API function call.

For seeding and reseeding the DRBG, the module uses a Non-Deterministic Random Number Generator (NDRNG). The NDRNG is provided by the operational environment (i.e., Linux RNG) through the use of /dev/urandom. The output of the NXP RNG (implemented in the hardware portion of the module) is used as a noise source by the Linux RNG. The module then obtains entropy through /dev/urandom. The NDRNG (Linux RNG) is within the module's physical boundary, but outside the module's logical boundary. The NDRNG provides, at its output, the seed for the DRBG in the form of a 384-bit entropy_input string. The DRNG is initialized during the module's initialization.

The output of the NDRNG provides at least 256 bits of entropy in its 384 bits of length that is then used for the DRBG initialization (seed) and reseeding. The entropy is sufficient for the security strength provided by the DRBG algorithm.

The NDRNG implements a continuous test on the output to ensure that consecutive random numbers do not repeat. The module performs DRBG health tests as defined in Section 11.3 of [SP800-90A].

7.2 Key Generation

For generating ECDSA and ECDH keys, the module implements asymmetric key generation services compliant with [FIPS186-4] and using a DRBG compliant to [SP800-90A]. A seed (i.e., the random value) used in asymmetric key generation is obtained from the [SP800-90A] DRBG. In accordance with [FIPS140-2_IG] D.12, the cryptographic module performs Cryptographic Key Generation (CKG) for asymmetric keys as per [SP800-133] (vendor affirmed).

Symmetric keys are derived from the shared secret established by EC Diffie- Hellman in a manner that is compliant to NIST SP 800-135 for TLS v1.2 KDF.

7.3 Key Derivation

The module supports key derivation for the TLS protocol. The module implements the key derivation and pseudo-random functions (PRF) for TLS1.0/1.1 in non-FIPS mode, and for TLS1.2 in FIPS mode.

7.4 Key Establishment

The module provides EC Diffie-Hellman key agreement scheme used as part of the TLS protocol key exchange (allowed by [FIPS140-2_IG] D.8). The module also provides Key Transport Methods, as allowed by [FIPS140-2_IG] D.9, as a part of the TLS protocol connection. These methods employ AES key wrapping per [SP800-38F].

The key transport methods are provided by:

- An approved authenticated encryption mode (i.e., AES-CCM, AES-GCM within the TLS protocol connection).
- A combination method that occurs exclusively within the context of a TLS protocol connection, using TLS ciphersuites. The combination method consists of using an approved

symmetric encryption mode (e.g., AES-128-CBC, AES-256-CBC) together with an approved HMAC authentication method, both defined by the TLS ciphersuite and in the approved mode of this module.

Table 7 specifies the key sizes allowed in FIPS mode of operation. According to “Table 2: Comparable strengths” in [SP800-57], the key sizes of AES key wrapping and EC Diffie-Hellman provide the following security strengths:

- Approved authenticated encryption mode key establishment methodology provides between 128 and 256 bits of encryption strength.
- Combination of approved AES encryption and approved authentication method (e.g., AES-CBC with HMAC) key establishment methodology provides 128 or 256 bits of encryption strength.
- EC Diffie-Hellman key establishment methodology provides 128 or 192 bits of encryption strength.

7.5 Key Entry/Output

AES secret keys and ECDSA private keys are provided to the module via API input parameters in plaintext form and output via API output parameters in plaintext form. The module does not support manual key entry or intermediate key output during the key generation process. The module does not enter or output keys in plaintext format outside its physical boundary.

7.6 Key/CSP Storage

Symmetric keys, HMAC keys, public and private keys are provided to the module by the calling application via API input parameters and are destroyed by the module when invoking the appropriate API function calls.

The module does not perform persistent storage of keys. The keys and CSPs are stored as plaintext in volatile memory (RAM). The protection of these keys and CSPs in RAM is provided by the operating system enforcement of separation of address space.

The HMAC key used for integrity test is stored in the module and relies on the operating system for protection.

7.7 Key/CSP Zeroization

The memory occupied by keys is allocated by regular memory allocation operating system calls. The application is responsible for calling the appropriate zeroization functions provided in the module's API and described in the API documentation. In addition, the calling application is responsible for parameters passed in and out of the module.

The zeroization mechanism replaces the memory occupied by keys and CSPs with “zeroes” and deallocates the memory with the regular memory deallocation operating system call.

8 Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)

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

9 Self-Tests

9.1 Power-On Self-Tests (POSTs)

The module performs power-up or power-on self-tests (POSTs), without operator intervention, when the module is powered up. Power-on self-tests ensure that the module is not corrupted and that the cryptographic algorithms work as expected.

While the module is executing the power-on self-tests, services are not available, and input and output are inhibited. The module is not available to be used by the calling application until the power-on self-tests are completed successfully.

If any power-on test fails, the module enters the error state and returns the FIPS_POST_FAIL error code. Subsequent calls to cryptographic services offered by the module will fail, thus no further cryptographic operations are possible. To recover from the error state, the module must be power-cycled (power-off, power-on) or reloaded, thus causing the POST to be executed.

If the power-on tests complete successfully, the module will set its status to FIPS_POST_SUCCESS, enter the FIPS mode and will accept cryptographic operation service requests.

9.1.1 Integrity Tests

The integrity of the module is verified by comparing an HMAC-SHA-1 value calculated at run time with the HMAC value stored in the module and computed at build time. If the HMAC values do not match, the test fails and the module enters the error state.

9.1.2 Cryptographic algorithm tests

The module performs self-tests on all FIPS-Approved cryptographic algorithms supported in the approved mode of operation, using Known Answer Tests (KAT). Table 11 details these self-tests.

Table 11: Self-tests.

Algorithm	Test
AES	• KAT AES-ECB with 128-bit key, encryption • KAT AES-ECB with 128-bit key, decryption • KAT AES-GCM with 256-bit key, encryption • KAT AES-GCM with 256-bit key, decryption • KAT AES-CCM with 192-bit key, encryption • KAT AES-CCM with 192-bit key, decryption
DRBG	• KAT CTR_DRBG using AES-256 with and without DF, with and without PR • KAT HASH_DRBG using SHA-256 with and without PR • KAT HMAC_DRBG using SHA-256 with and without PR
ECDSA	• PCT ECDSA signature generation and verification with P-384 and SHA-512 (allowed per IG 9.4 [FIPS140-2_IG])
HMAC	• KAT HMAC-SHA-1 • KAT HMAC-SHA-224 • KAT HMAC-SHA-256 • KAT HMAC-SHA-384 • KAT HMAC-SHA-512

Algorithm	Test
SHS	• KAT SHA-1
KAS ECC	• Primitive “Z” Computation KAT with P-384 curve

The module calculates the result and compares it with the known value. If the answer does not match the known answer, the KAT fails and the module enters the Error state. For the Pairwise Consistency Test (PCT), the module generates a signature using the respective algorithm with known keys, and then verifies the generated signature using the same algorithm. If the PCT fails, the module enters the Error state.

9.2 On-Demand self-tests

On-Demand self-tests can be invoked by powering-cycling the module (power-off, power-on) or reloading the module, thus forcing the module to run the power-on self-tests.

9.3 Conditional Tests

The module performs conditional tests on the cryptographic algorithms per Table 12. If any of the conditional tests fail, the module goes into the error state and becomes unavailable as described in Section 9.1.

Table 12: Conditional tests.

Algorithm	Test
ECDSA key generation	• Pairwise consistency test (PCT)
NDRNG	• Continuous test (previous and current random data are tested for equality)

The CRNGT on the [SP800-90A] DRBG is not required per IG 9.8 [FIPS140-2_IG].

10 Guidance

10.1 Crypto Officer Guidance

The module is a hardware-software-hybrid cryptographic module and the software part of the module is pre-installed on a managed security appliance. System administrators who maintain the appliance will receive the libraries and modules in the form of a pre-packaged OS image that is created by Cyphre (the vendor). The end user will not have package installation access to the device.

10.1.1 Module Installation and Configuration

In order to install and configure the module and run the module in the FIPS mode of operation, the crypto officer needs to configure the platform to use the fipscanister build mechanism. The crypto officer must follow the instructions summarized next.

Load into U-Boot to rescue boot the appliance.

1. Press the power button to begin the BT-1 boot process while connected to the console.
2. When prompted with message “Press ESC for Setup or TAB for U-Boot prompt:”
    a. Press TAB to enter U-Boot.
    b. Within U-Boot type:

    ```
    setenv othbootargs sgyboot=break && boot
    ```

    c. Press Enter.
3. The following will be displayed, indicating you are in Rescue Boot:

    ```
    Servergy Boot Loader.
    ==> You are now in a standard shell.
    svy-boot#
    ```

Upgrading Flash Firmware – in Rescue Boot.

1. Issue the following command to obtain an IP address on eth0.

    ```
    udhcpc -i eth0
    ```

2. Change the working directory to /bin/manufacturing.

    ```
    cd /bin/manufacturing
    ```

3. Edit the flash-update.sh file.

    ```
    sed -i 's/-nv/ /g' flash-update.sh
    ```

4. Run the modified flash-update.sh file.

    ```
    ./flash-update.sh
    ```

5. You should see the following message:
    a. II: Completed successfully
    b. If you receive a “EE: SYSTEM MAY NOT BE BOOTABLE!!!!”, run the flash update again. Do not reboot the system. Contact a member of the Cyphre team.
6. If successful, reboot the system.

    ```
    reboot -f
    ```

Provisioning the File System – in Rescue Boot

1. When prompted with message “Press ESC for Setup or TAB for U-Boot prompt:”
    a. Press TAB to enter U-Boot.
    b. Within U-Boot type:

        setenv othbootargs sgyboot=break && boot

    c. Press Enter
2. The following will be displayed, indicating you are in Rescue Boot.

    Servergy Boot Loader.
    ==> You are now in a standard shell.
    svy-boot#

3. Issue the following command to obtain an IP address on eth0.

    udhcpc -i eth0

4. Update the system clock time.

    ntpdate pool.ntp.org
    hwclock –systohc

5. Issue the following command to obtain the Serial Number, make sure the serial number listed machines the number printed on the side of the appliance.

    printserial && echo

6. Change the working directory to /bin/manufacturing.

    cd /bin/manufacturing

7. Delete existing file system.

    ./delete-storage.sh YESIUNDERSTAND

8. Provision new file system with the serial number (This will take a long time).
    a. If the serial number output in serial console is does not match the number printed on the appliance, use the number printed on the appliance in the next step.
    b. Replace SERIALNUM in the below command.

        ./provision.sh SERIALNUM YESIUNDERSTAND

9. Once completed, you should see the following:

    COMPLETED SUCCESSFULLY!
    SUCCESS: Provision passed

10. Reboot the appliance.

    reboot -f

Now, the development environment must be configured to meet the following requirements:

- Application and libraries linking the module need to be built with position independent code.
- The user-space FIPS module and supporting code reside in the same source tree. This requires a two-stage build. The first, fipscanisteronly, creates the FIPS module code. The second, fips, creates the supporting routines around the FIPS module code.

This two-stage build process is defined in the script “native-build.sh”, which resides at the top directory of the source tree. This script shall be used to build the module.

Please refer to Section “Provisioning the File System – in Rescue Boot” and Section “Upgrading the Operating System” in the “Cyphre BT-1 Firmware and OS Provisioning v1.4.0” document for additional details.

10.2 User Guidance

In order to run in FIPS Approved mode of operation, the module must be operated using the FIPS approved services, with their corresponding FIPS approved or FIPS allowed cryptographic algorithms provided in this Security Policy (Section 4.2). In addition, key sizes must comply with [SP800-131A].

10.2.1 API Functions

Services provided by the module shall be requested using the API functions as specified in the User Guide. Directly invoking functions outside of the API specification, or directly modifying global variables and indicators outside of the API specification is prohibited. Doing so implicitly switches the module out of the FIPS-mode, and the module can only achieve the FIPS-mode again by resetting the module. Resetting the module automatically invokes the Power-On Self-Test and ensures the module is properly operational.

10.2.2 AES-GCM IV

AES GCM encryption and decryption are used in the context of the TLS protocol version 1.2. The module is compliant with [SP 800-52] and the mechanism for IV generation is compliant with [RFC5288]. The operations of one of the two parties involved in the TLS key establishment scheme are performed entirely within the cryptographic boundary of the module, including the setting of the counter portion of the IV.

When the nonce_explicit part of the IV exhausts the maximum number of possible values for a given session key, the module (acting as server or client) triggers a handshake to establish a new encryption key per Section 7.4.1.1 and Section 7.4.1.2 in [RFC5246] and compliant to [FIPS140-2_IG] A.5.

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

10.2.3 Key Usage and Management

In general, a single key shall be used for only one purpose (e.g., encryption, integrity, authentication, key wrapping, random bit generation, or digital signatures) and be disjoint between the modes of operations of the module. Thus, if the module is switched between its FIPS mode and non-FIPS mode or vice versa (Section 2.3), the following procedures shall be observed:

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

10.2.4 Handling FIPS Related Errors

When the module fails any self-test, the module enters the error state. In this state, cryptographic operations are not allowed and output is inhibited. Any invocation to API functions returns the value 134 to the calling application.

The user will also receive the following error message:

```
fips.c(139): OpenSSL internal error, assertion failed: FATAL FIPS SELFTEST FAILURE
Aborted
```

Applications can also obtain the status of the module by calling the *FIPS_selftest_failed* function.

11 Mitigation of Other Attacks

There are no mitigations from other attacks.

Appendix A. Glossary and Abbreviations

AES	Advanced Encryption Standard
CAAM	Cryptographic Accelerator and Assurance Module
CAVP	Cryptographic Algorithm Validation Program
CAVS	Cryptographic Algorithm Validation System
CBC	Cipher Block Chaining
CMVP	Cryptographic Module Validation Program
CSP	Critical Security Parameter
CTR	Counter Mode
DES	Data Encryption Standard
DF	Derivation Function
DRBG	Deterministic Random Bit Generator
ECB	Electronic Code Book
FIPS	Federal Information Processing Standards Publication
HMAC	Hash Message Authentication Code
KAT	Known Answer Test
MAC	Message Authentication Code
NIST	National Institute of Science and Technology
PAA	Processor Algorithm Acceleration
PCT	Pair-wise Consistency Test
POST	Power-on Self-Tests
PR	Prediction Resistance
RNG	Random Number Generator
RSA	Rivest, Shamir, Addleman
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard

Appendix B. References

FIPS140-2 **FIPS PUB 140-2 - Security Requirements for Cryptographic Modules**
May 2001
http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

FIPS140-2_IG **Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program**
May 7, 2019
https://csrc.nist.gov/CSRC/media/Projects/cryptographic-module-validation-program/documents/fips140-2/FIPS1402IG.pdf

FIPS180-4 **Secure Hash Standard (SHS)**
March 2012
http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

FIPS186-4 **Digital Signature Standard (DSS)**
July 2013
http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

FIPS197 **Advanced Encryption Standard**
November 2001
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

FIPS198-1 **The Keyed Hash Message Authentication Code (HMAC)**
July 2008
http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf

PKCS#1 **Public Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.2**
November 2016
https://tools.ietf.org/rfc/rfc8017.txt

SP800-38A **NIST Special Publication 800-38A - Recommendation for Block Cipher Modes of Operation Methods and Techniques**
December 2001
http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

SP800-52 **NIST Special Publication 800-52 Revision 1 - Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations**
April 2014
http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r1.pdf

SP800-56A **NIST Special Publication 800-56A - Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised)**
March, 2007
http://csrc.nist.gov/publications/nistpubs/800-56A/SP800-56A_Revision1_Mar08-2007.pdf

SP800-67	**NIST Special Publication 800-67 Revision 2 - Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher** November 2017 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-67r2.pdf
SP800-90A	**NIST Special Publication 800-90A - Revision 1 - Recommendation for Random Number Generation Using Deterministic Random Bit Generators** June 2015 http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf
SP800-131A	**NIST Special Publication 800-131A Revision 2- Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths** March 2019 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar2.pdf
SP800-135	**NIST Special Publication 800-135 Revision 1 - Recommendation for Existing Application-Specific Key Derivation Functions** December 2011 http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-135r1.pdf