Intel® QuickAssist Technology (QAT) Provider v1.3.1

Cryptographic Module

Non-Proprietary FIPS 140-3 Security Policy

FIPS 140-3 Security Level:	1
Document version:	1.17
Date:	May 9, 2025

Table of Contents

0 Introduction 5

0.1 Purpose 5
0.2 References 5
0.3 Document Organization 5

1 General Overview 6

2 Cryptographic Module Specification 7

2.1 Module Specification 7
2.2 Algorithms Implementation 9
2.3 Modes of Operation 11

3 Cryptographic Module Interfaces 12

4 Roles, Services and Authentication 13

4.1 Authorized Roles and Authentication 13
4.2 Module Services 14

5 Software/Firmware Security 17

6 Operational environment 18

7 Physical security 19

8 Non-invasive security 20

9 Sensitive Security Parameter Management 21

9.1 Random number generation 24
9.2 SSP generation 24
9.3 SSP Agreement and SSP Derivation 24
9.4 SSP Entry/Output 24
9.5 SSP Storage 24
9.6 SSP Zeroization 25

10 Self-Tests 26

10.1 Pre-operational Self-Tests 26
10.2 Conditional Self-Tests 26
10.3 Error states 28
10.4 Operator Initiation Self-Tests 29

11 Life-cycle Assurance 30

11.1 Delivery and Operation 30
11.2 Crypto Officer Guidance 30
11.3 Installation 30
11.4 Component versioning identification 30
11.5 Non-reconfigurable memory 31

12 Mitigation of Other Attacks 32

13 Secure Guidance 33

13.1 AES-GCM Usage 33
13.2 Intel suggestion on use of cryptographic algorithms 33

14 Appendix A 34

15 References and Acronyms 35

15.1 References 35
15.2 Acronyms 35

List of Tables

Table 1 – Security Levels 6
Table 2 – Cryptographic Module Components 7
Table 3 – Tested Operational Environments 9
Table 4 – Cryptographic Algorithm Providers 9
Table 5 – Approved Algorithms 10
Table 6 – Ports and Interfaces 12
Table 7 – Roles, Services Commands, Input, and Output 13
Table 8 – Approved Services 15
Table 9 – SSPs 21
Table 10 - Storage Areas 25
Table 11 – Pre-operational integrity Self-Tests 26
Table 12 - Conditional Self-Tests of Hardware implementation 27
Table 13 - Conditional Self-Tests of Software implementation 28
Table 14 - Conditional Self-Tests of Firmware implementation 28
Table 15 - Error States 29
Table 16 - References 35
Table 17 - Acronyms definitions 35

List of Figures

Figure 1 – Cryptographic boundary 8
Figure 2 – CPU 9

0 Introduction

0.1 Purpose

This is a non-proprietary Cryptographic Module Security Policy for the Intel® QuickAssist Technology (QAT) Provider from Intel Corporation. This Security Policy describes how the Intel® QAT Provider meets the security requirements of Federal Information Processing Standards (FIPS) Publication 140-3, which details the U.S. and Canadian Government requirements for cryptographic modules. More information about the FIPS 140-3 standard and validation program is available on the National Institute of Standards and Technology (NIST) and the Canadian Centre for Cyber Security (CCCS) Cryptographic Module Validation Program (CMVP) website at https://csrc.nist.gov/projects/cryptographic-module-validation-program.

This document also describes how to run the Intel® QAT Provider in a secure Approved mode of operation. This policy was prepared as part of the Level 1 FIPS 140-3 validation. The Intel® QuickAssist Technology (QAT) Provider is referred to in this document as the module.

0.2 References

This document deals only with operations and capabilities of the module in the technical terms of a FIPS 140-3 cryptographic module security policy. More information is available on the module from the following sources:

- The module website (https://github.com/intel/QAT_Engine) contains additional information of the module.
- The search page on the CMVP website (https://csrc.nist.gov/Projects/cryptographic-module-validation-program/Validated-Modules/Search) can be used to locate and obtain vendor contact information for technical or sales-related questions about the module.

0.3 Document Organization

ISO/IEC 19790 Annex B uses the same section naming convention as ISO/IEC 19790 section 7 - Security requirements. For example, Annex B section B.2.1 is named “General” and B.2.2 is named “Cryptographic module specification,” which is the same as ISO/IEC 19790 section 7.1 and section 7.2, respectively. Therefore, the format of this Security Policy is presented in the same order as indicated in Annex B, starting with “General” and ending with “Mitigation of other attacks.” If sections are not applicable, they have been marked as such in this document.

1 General Overview

This document is the non-proprietary FIPS 140-3 Security Policy of the Intel® QAT Provider cryptographic module. For the purpose of the FIPS 140-3 validation, the module is a software-hybrid, multiple-chip standalone cryptographic module validated at overall security level 1. The following table shows the claimed security level for each of the twelve sections that comprise the FIPS 140-3 standard.

Table 1 – Security Levels

Section	FIPS 140-3 Section	Security Level
1	General	1
2	Cryptographic module specification	1
3	Cryptographic module interfaces	1
4	Roles, services, and authentication	1
5	Software/Firmware security	1
6	Operational environment	1
7	Physical security	1
8	Non-invasive security	N/A
9	Sensitive security parameter management	1
10	Self-tests	1
11	Life-cycle assurance	1
12	Mitigation of other attacks	N/A

2 Cryptographic Module Specification

2.1 Module Specification

Intel® QuickAssist Technology (QAT) Provider (hereafter referred to as "the module") supports acceleration for both hardware as well as optimized software based on vectorised instructions.

It is a software-hybrid module (compatible with OpenSSL 3.0.8) which supports the ability to accelerate the operations from the stand OpenSSL 3.0.8 to basic Intel instruction set, to either hardware acceleration path (via the qat_hw) or via the optimized software acceleration path (qat_sw). Both are packaged under the same shared library, called qatprovider.so.

OpenSSL 3.0.8 is a toolkit for TLS/SSL protocols and has developed a modular system to plugin device-specific engines. As mentioned above, within the module are two separate internal entities by which acceleration can be performed. Depending on your particular use case, the module can be configured to meet specific acceleration needs.

Software acceleration in the module is achieved by using the qat_sw in conjunction with the following supporting libraries:

- Intel® Integrated Performance Primitives Cryptographic library (called libcrypto_mb.so) for asymmetric key-based algorithms acceleration.
- Intel® Multi-Buffer Crypto library (called libIPsec_MB.so) for the acceleration for the AES-GCM algorithm used in the TLS context.

As to the hardware acceleration in the module is achieved by using the qat_hw in conjunction with a dedicated hardware accelerator as well as the Intel® QAT Driver which provides the API interface for the accelerator.

Based on that, the cryptographic boundary consists of the following shared libraries as well as the dedicated hardware acceleration device and the firmware running on it. The following table enumerates the elements that comprise the module (surrounded with red lines in Figure 1).

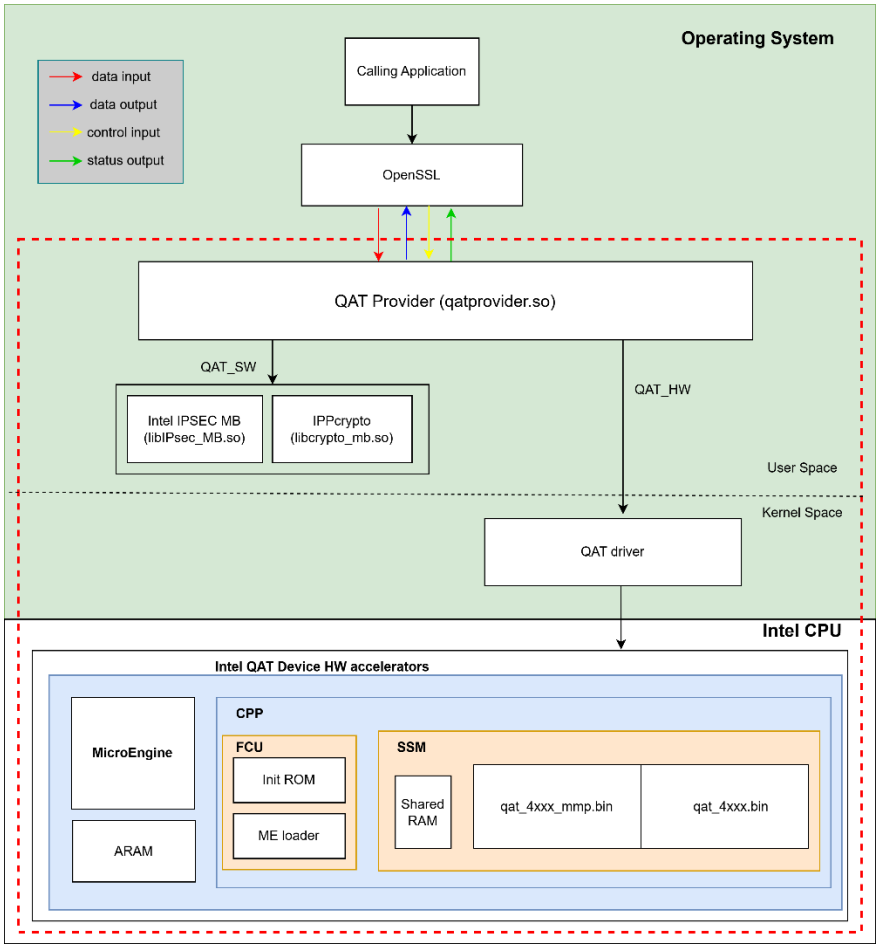
Table 2 – Cryptographic Module Components

Component	Description
qatprovider.so	Shared library for QAT_Provider implementation
libIPsec_MB.so	Shared library for AES-GCM SW acceleration.
libcrypto_mb.so	Shared library for asymmetric key-based algorithms SW acceleration
qat_4xxx.ko, usdm_drv.ko, intel_qat.ko, libusdm_drv_s.so, libqat_s.so	Intel® QAT Driver for the hardware accelerator (not cryptographic implementation)
qat_4xxx.bin and qat_4xxx_mmp.bin	Firmware running over the dedicated hardware accelerator

Component	Description
Intel® QuickAssist Technology SoC	Dedicated hardware acceleration device

The image below illustrates the high-level boundary of the module. Applications such as NGINX and HAProxy are common applications which interfaces to OpenSSL. The block diagram below shows the module, its interfaces with the operational environment (EVP API coming from OpenSSL) and the delimitation of the cryptographic module boundary (dotted red line), which comprised the module and related files and the dedicated Intel® QAT hardware accelerator. Note that green zone represents the Operating System, which is split by the black dotted line in User Space and Kernel Space. Red, blue, yellow and green arrows represent the data input, data output, control input and status output flow data respectively. The TOEPP is the general-purpose computer (GPC) where the processor is installed together with the accelerator hardware device.

Figure 1 – Cryptographic boundary



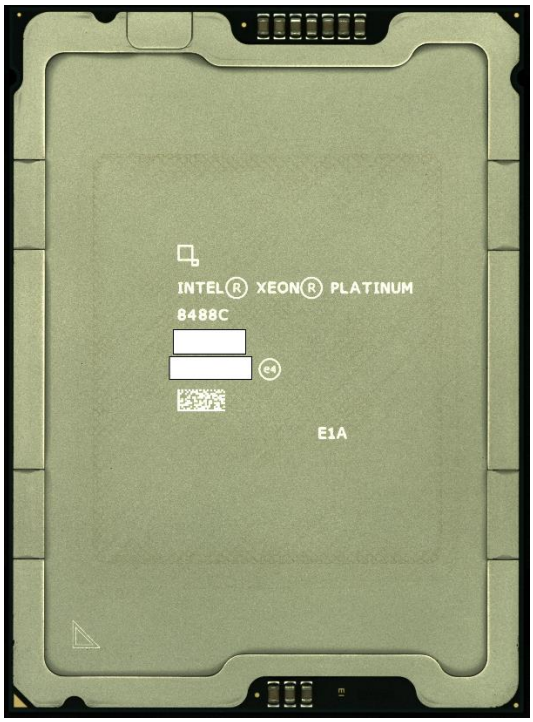
The module is aimed to run on a general-purpose computer with compatible processor with QAT that supports hardware acceleration. The module has been tested and found compliant on the following platforms, being the platform the physical perimeter of the module:

Table 3 – Tested Operational Environments

Operating system	Hardware Platform	Processor	PAA/Acceleration	OpenSSL Version	Hardware acceleration device
Red Hat Enterprise Linux 9.0	Intel Eagle Stream	Intel® Xeon® Platinum 8488C	Yes	OpenSSL 3.0.8	Intel Corporation Device 4940 (rev 40)

There is no additional vendor affirmed operating environments.

Figure 2 – CPU



2.2 Algorithms Implementation

The module includes the cryptographic algorithm providers listed in Table 4 below:

Table 4 – Cryptographic Algorithm Providers

CAVP Certificate	Implementation Name and Version	Use
A4390	Intel® QAT Provider Hardware Acceleration Device 4940 (rev40)	Cryptographic primitives for those algorithms which support hardware acceleration.
A4391	Intel® Multi-Buffer Crypto for IPsecMB Library v1.3 for Software Acceleration	Cryptographic library for AES GCM and SHA2 which support software acceleration.
A4389	Crypto Multi-buffer library IPP Crypto v2021.7.1 for Software Acceleration	Cryptographic library for RSA, ECDSA and ECDH which support software acceleration.

A4392	Intel® Authentication Firmware V1.0.40	Cryptographic implementation for RSA SigVer and SHA256 for Hardware Integrity Test binaries

The module supports the following Approved Algorithms listed in Table 5 below:

Table 5 – Approved Algorithms

CAVP Certificate	Algorithm	Standard	Mode/Method	Key Lengths Curves/Module (in bits)	Use
A4390	AES	NIST SP 800-38D	GCM	128, 256	Authenticated encryption/decryption
A4390	CVL	NIST SP 800-135rev1 RFC 8446	TLS versions v1.2 (The extended master secret parameter is defined in Section 4 of RFC 7627) and v1.3	SHA2-256, SHA2-384	Application-specific Key Derivation *No parts of these protocols, other than the KDFs, have been tested by the CAVP or CMVP.*
A4390	DSA	FIPS PUB 186-4	SigGen	L:2048 - N:224 L:2048 - N:256 L:3072 - N:256	Digital Signature Generation
			SigVer	L:1024- N:160 L:2048 - N:224 L:2048 - N:256 L:3072 - N:256	Digital Signature Verification
A4390	ECDSA	FIPS PUB 186-4	SigGen	P-224, P-256, P-384, P-521 B-233, B-283, B-409, B-571 K-233, K-283, K-409, K-571	Digital Signature Generation
			SigVer		Digital Signature Verification
A4390	KAS-ECC-SSC	NIST SP 800-56Arev3	ephemeralUnified scheme	P-224, P-256, P-384, P-521 B-233, B-283, B-409, B-571 K-233, K-283, K-409, K-571	EC Diffie-Hellman Key Agreement Shared Secret Computation *Key establishment methodology provides between 112 and 256 bits of encryption strength*
A4390	KAS-FFC-SSC	NIST SP 800-56Arev3	dhEphem scheme	ffdhe2048, ffdhe3072, ffdhe4096, ffdhe8192	Diffie-Hellman Key Agreement Shared Secret Computation *Key establishment methodology provides between 112 and 200 bits of encryption strength*

A4390	RSA	FIPS PUB 186-4	SigGen (PKCS#1-v1.5)	2048, 3072, 4096	Digital Signature Generation
			SigVer (PKCS#1-v1.5)	1024, 2048, 3072, 4096	Digital Signature Verification
A4390	SHA-3	FIPS PUB 202	SHA3-224, SHA3-256, SHA3-384, SHA3-512	-	Message Digest
A4391	AES	NIST SP 800-38D	GCM	128, 192, 256	Authenticated encryption/decryption
A4389	ECDSA	FIPS PUB 186-4	SigGen	P-256, P-384	Digital Signature Generation
			SigVer		Digital Signature Verification
A4389	KAS-ECC-SSC	NIST SP 800-56Arev3	EphemeralUnified scheme	P-256, P-384	EC Diffie-Hellman Key Agreement Shared Secret Computation *Key establishment methodology provides between 112 and 192 bits of encryption strength*
A4389	RSA	FIPS PUB 186-4	SigGen (PKCS#1-v1.5)	2048, 3072, 4096	Digital Signature Generation
			SigVer (PKCS#1-v1.5)		Digital Signature Verification
A4391	SHS	FIPS 180-4	SHA2-224, SHA2-256, SHA2-384, SHA2-512	-	Message Digest
A4392	RSA	FIPS PUB 186-4	SigVer (PKCS#1-v1.5)	3072	Digital Signature Verification
A4392	SHS	FIPS 180-4	SHA2-256	-	Message Digest

The module does not implement either non-approved but allowed, non-approved allowed algorithms with no security claimed or non-approved and not allowed algorithms in the Approved mode of operation.

The module does not implement any vendor affirmed algorithm or security method.

2.3 Modes of Operation

The module supports one mode of operation: Approved. The module will be in Approved mode when all pre-operational and conditional algorithms self-tests have completed successfully, and only Approved and non-approved but allowed security functions are invoked (see Table 5 and **¡Error! No se encuentra el origen de la referencia.** above).

The module does not support degraded operation.

3 Cryptographic Module Interfaces

As a software-hybrid module, the module does not have physical ports. For the purpose of the FIPS 140-3 validation, the module interfaces are defined as Hybrid Software (HSMI), and the physical ports are interpreted to be the physical ports of the hardware platform on which the module runs. The physical ports include the computer network ports, keyboard port, mouse port, power plug and display.

The logical interfaces are a C language entry functions pointed by OpenSSL library interface through which calling application request services.

The following table summarizes the FIPS 140-3 logical interfaces:

Table 6 – Ports and Interfaces

Physical Port	Logical Interface	Data that passes over port/interface
Keyboard interface, Mouse interface, Network interface	Control input	Command function Command control parameters
Keyboard interface, Mouse interface, Network interface	Data input	Plaintext data to be ciphered or signed Ciphertext data to be decrypted Signed data to be verified Cryptographic keys and other key management data
Display controller, Network interface	Data output	Decrypted plaintext data Encrypted ciphertext data Generated signature data Derived plaintext key material
Display controller, Network interface	Status output	Command function result Command status parameters

When the module is performing self-tests, or is in an error state, all output on the logical data output interface is inhibited.

As to the power interface, there is no separate power interface beyond the power interface provided by the module host platform itself. The module does not implement control output interface.

4 Roles, Services and Authentication

The sections below describe the module's authorized role, services, and operator authentication method employed.

4.1 Authorized Roles and Authentication

The module only supports the following role: the Crypto Officer role. It performs all services as well as the module installation and configuration.

The module does not support authentication mechanisms. The module does not support concurrent operators.

The Crypto Officer role is implicitly assumed by using the module and invoking any of the available services.

Table 7 – Roles, Services Commands, Input, and Output

Roles	Service	Input	Output
Crypto Officer	Installation	None	None
Crypto Officer	Initialization	API call parameters	Status
Crypto Officer	Self-test on demand	Power cycle	Status
Crypto Officer	Show Status	None	Status
Crypto Officer	Show version information	API call parameters	Status
Crypto Officer	Zeroization	Power cycle or using provided free APIs	Status
Crypto Officer	Symmetric encryption/decryption	API call parameters, key, IV, plaintext/ciphertext	Status, ciphertext/plaintext
Crypto Officer	DSA digital signature generation	API call parameters, DSA private key, message	Status, signature
Crypto Officer	DSA digital signature verification	API call parameter, DSA public key, signature	Status, message
Crypto Officer	ECDSA digital signature generation	API call parameters, ECDSA private key, message	Status, signature
Crypto Officer	ECDSA digital signature verification	API call parameter, ECDSA public key, signature	Status, message
Crypto Officer	RSA digital signature generation	API call parameters, RSA private key, message	Status, signature
Crypto Officer	RSA digital signature verification	API call parameter, RSA public key, signature	Status, message
Crypto Officer	EC Diffie-Hellman Key Agreement Shared Secret Computation	API call parameter, key	Status, key components, key

Roles	Service	Input	Output
Crypto Officer	Diffie-Hellman Key Agreement Shared Secret Computation	API call parameters, key	Status, domain parameters, key
Crypto Officer	Message digest	API call parameters, message	Status, hash
Crypto Officer	TLS key derivation	API call parameters, TLS pre-master secret	Status, session key, integrity key

4.2 Module Services

All services implemented by the module are listed in tables below. The approved and allowed services are shown in Table 8. Please note that the Sensitive Security Parameters (SSPs) listed below indicate the type of access required using the following notation:

- G - Generate: The SSP is generated or derived.
- R – Read: The SSP is read from the module.
- W – Write: The SSP is updated, imported or written to the module.
- X – Execute: The SSP is used within an Approved or Allowed security function.
- Z – Zeroize: The SSP is zeroized.

Table 8 – Approved Services

Service	Description	Approved Security Function	Roles	SSP and Type of Access	Indicator
Installation	Module installation and configuration	None	CO	None	-
Initialization	Perform initialization of the module	None	CO	None	-
Self-test on demand	Perform pre-operational and conditional algorithms self-tests	None	CO	None	-
Show Status	Implicit service, since only one mode of operation is available, the status of the module can be determined with the indicator of a completion execution of the pre-operational and conditional self-tests.	None	CO	None	-
Show version information	Display the module name and version	None	CO	None	-
Zeroization	Zeroize and de-allocate memory that contains sensitive data	None	CO	All SSP's - Z	-
Symmetric encryption/decryption	Encrypt/Decrypt plaintext/ciphertext using supplied key	AES (Cert. A4390; Cert. A4391)	CO	AES GCM Key - WX AES GCM IV -WX	qat_fips_service_indicator=1
DSA digital signature generation	Generate a DSA digital signature	DSA sigGen (Cert. A4390)	CO	DSA Private Key - WX	qat_fips_service_indicator=1
DSA digital signature verification	Verify a DSA digital signature	DSA sigVer (Cert. A4390)	CO	DSA Public Key - WX	qat_fips_service_indicator=1
ECDSA digital signature generation	Generate an ECDSA digital signature	ECDSA sigGen (Cert. A4390; Cert. A4389)	CO	ECDSA Private Key - WX	qat_fips_service_indicator=1
ECDSA digital signature verification	Verify an ECDSA digital signature	ECDSA sigVer (Cert. A4390; Cert. A4389)	CO	ECDSA Public Key - WX	qat_fips_service_indicator=1
RSA digital signature generation	Generate an RSA digital signature	RSA sigGen (Cert. A4390; Cert. A4389)	CO	RSA Private Key - WX	qat_fips_service_indicator=1

Service	Description	Approved Security Function	Roles	SSP and Type of Access	Indicator
RSA digital signature verification	Verify an RSA digital signature	RSA sigVer (Cert. A4390; Cert. A4389)	CO	RSA Public Key - WX	qat_fips_service_indicator=1
EC Diffie-Hellman Key Agreement Shared Secret Computation	Shared secret computation using an ECDH scheme	KAS-ECC-SSC (Cert. A4390; Cert. A4389)	CO	ECDH Public Key - RX ECDH Private – RX Shared Secret - G	qat_fips_service_indicator=1
Diffie-Hellman Key Agreement Shared Secret Computation	Shared secret computation using a DH scheme	KAS-FFC-SSC (Cert. A4390)	CO	DH Public Key - RX DH Private – RX Shared Secret - G	qat_fips_service_indicator=1
Message digest	Compute and return a message digest using SHS and SHA-3 algorithms	SHA (Cert. A4390; Cert. A4391);	CO	None	qat_fips_service_indicator=1
TLS key derivation	Key derivation for TLS v1.2/1.3	CVL (Cert. A4390)	CO	TLS pre-master secret -WX TLS master secret - GRX TLS session key - G TLS integrity key - G	qat_fips_service_indicator=1

5 Software/Firmware Security

As it is described in section 2.1 the cryptographic module is composed of several software shared libraries as well as the firmware component running over the dedicated QAT hardware accelerator.

The integrity of the software libraries is achieved by applying an ECDSA (with a P-256 curve) with SHA2-256 algorithm over them at the build time. All the obtained signature values are stored in the headers of the qatprovider.so file.

After that, every time that a calling application uses the module, the execution flow passes first through the initialization part of the module, where the integrity is self-verified as follows:

- The SHA2-256 and ECDSA algorithms used for the integrity verification are self-tested using a Known Answer Test (KAT) together with the rest of the cryptographic algorithm self-tests (both Hardware and Software algorithms implementation).
- Once the cryptographic algorithms self-tests have been successfully executed, the module verifies the integrity of the software components and firmware component of the hardware part using the expected signature of each one stored in the headers of the “qatprovider.so” file.
- In case that any operation fails, the error condition is triggered, and the module execution is totally stopped.

As to the firmware part, the integrity is achieved based on and RSA digital signature using a 3072-bit key and SHA2-256. The signature is embedded within the firmware together with the associated RSA public key. The process of the verification is as follows:

- The RSA and SHA algorithms are implemented in a separate authentication firmware which is hardcoded in a ROM. This ROM also contains a table with the digest value for all Intel valid public keys.
- First of all, the execution flow performs a KAT test over the above-mentioned algorithms to verify its correct operation.
- After that, the public key embedded within the QAT firmware is extracted, its digest value is computed and compared with the values stored in the table, in order to verify that the embedded public key matches with one of the valid public keys.
- Once the public key has been verified, the digital signature associated with the firmware is extracted for its verification. If the verification is successful, the executable image of the firmware is loaded in the secure RAM for its execution. If not, the process is aborted and the firmware is not loaded, and therefore, executed.

The pre-operational integrity test of the firmware part is triggered always that the QAT accelerators located in the processor are turned on.

6 Operational environment

The module operates in a modifiable operational environment per FIPS 140-3 level 1 specifications. The module runs on a commercially available general-purpose operating system executing on the hardware specified in Table 3.

The operating system is restricted to a single operator. Concurrent operators are explicitly excluded.

The application that requests cryptographic services is the single user of the module. All SSPs are under the control of the OS, which protects its CSPs against unauthorized disclosure, modification, and substitution as well as PSPs against modification and substitution. Additionally, the OS provides dedicated process space to each executing process, and the module operates entirely within the calling application's process space. The module only allows access to SSPs through its well-defined interfaces. The module does not have the ability of spawning new processes.

7 Physical security

The module is a software-hybrid with a multiple-chip standalone embodiment. The hardware components of the module consist of production-grade components that include standard passivation techniques. Further, the module is entirely contained within a hard metal enclosure, which blocks physical access to the module.

Since the module is evaluated at security level 1 and there is no maintenance interface, no more physical security mechanisms than the above described are implemented.

8 Non-invasive security

The module does not implement any non-invasive attack mitigation technique to protect itself and the module's unprotected SSPs from non-invasive attacks.

9 Sensitive Security Parameter Management

The following section includes all the information related to the Sensitive Security Parameters (SSPs) and its management. Table 9 below identifies all the SSPs handled by the cryptographic module as well as its purpose, generation method, input and output method, where they are stored and how they are zeroized.

Table 9 – SSPs

Key/SSP Name	Strength	Security Function Cert Number	Generation	Import/Export	Establishment	Storage	Zeroization	Use
AES GCM Key	128, 192, 256	AES GCM (Cert. A4390; Cert. A4391)	External	Plaintext (MD/EE) / Never exits the module	N/A	Not persistently stored	Power cycle or API call	Authenticated encryption and decryption
AES GCM IV	Depends on the used AES GCM key	AES GCM (Cert. A4390; Cert. A4391)	External	Plaintext (MD/EE) / Never exits the module	N/A	Not persistently stored	Power cycle or API call	Initialization vector for AES_GCM
DSA Public Key	80, 112, 128	DSA sigVer (Cert. A4390)	External	Plaintext (MD/EE) / Never exits the module	N/A	Not persistently stored	Power cycle or API call	Signature verification
DSA Private Key	112, 128	DSA sigGen (Cert. A4390)	External	Plaintext (MD/EE) / Never exits the module	N/A	Not persistently stored	Power cycle or API call	Signature generation
ECDSA Public Key	112 - 256	ECDSA sigVer (Cert. A4390; Cert. A4389)	External	Plaintext (MD/EE) / Never exits the module	N/A	Not persistently stored	Power cycle or API call	Signature verification
ECDSA Private Key	112 - 256	ECDSA sigGen (Cert. A4390; Cert. A4389)	External	Plaintext (MD/EE) / Never exits the module	N/A	Not persistently stored	Power cycle or API call	Signature generation

Key/SSP Name	Strength	Security Function Cert Number	Generation	Import/Export	Establishment	Storage	Zeroization	Use
RSA Public Key	80, 112, 128, 152	RSA sigVer (Cert. A4390; Cert. A4389)	External	Plaintext (MD/EE) / Never exits the module	N/A	Not persistently stored	Power cycle or API call	Signature verification
RSA Private Key	112, 128, 152	RSA sigGen (Cert. A4390; Cert. A4389)	External	Plaintext (MD/EE) / Never exits the module	N/A	Not persistently stored	Power cycle or API call	Signature generation
DH Public Key	112, 128, 152, 200	KAS-FFC-SSC (Cert. A4390)	External	Plaintext (MD/EE) / Never exits the module	N/A	Not persistently stored	Power cycle or API call	Shared secret generation
DH Private Key	112, 128, 152, 200	KAS-FFC-SSC (Cert. A4390)	External	Plaintext (MD/EE) / Never exits the module	N/A	Not persistently stored	Power cycle or API call	Shared secret generation
ECDH Public Key	112 - 256	KAS-ECC-SSC (Cert. A4390; Cert. A4389)	External	Plaintext (MD/EE) / Never exits the module	N/A	Not persistently stored	Power cycle or API call	Shared secret generation
ECDH Private Key	112 - 256	KAS-ECC-SSC (Cert. A4390; Cert. A4389)	External	Plaintext (MD/EE) / Never exits the module	N/A	Not persistently stored	Power cycle or API call	Shared secret generation
Shared Secret	112 - 256	KAS-FFC-SSC (Cert. A4390); KAS-ECC-SSC (Cert. A4390; Cert. A4389); CVL (Cert. A4390)	Generated inside the module	NA / Plaintext (MD/EE)	N/A	Not persistently stored	Power cycle or API call	Derivation of the TLS pre-master secret

Key/SSP Name	Strength	Security Function Cert Number	Generation	Import/Export	Establishment	Storage	Zeroization	Use
TLS pre-master secret	256, 384	CVL (Cert. A4390)	External	Plaintext (MD/EE) / Never exits the module	N/A	Not persistently stored	Power cycle or API call	Derivation of the TLS master secret
TLS master secret	256, 384	CVL (Cert. A4390)	Derived internally using the TLS pre-master secret via TLS KDF	NA / Never exits the module	N/A	Not persistently stored	Power cycle or API call	Derivation of the TLS session key and TLS integrity key
TLS session key	128, 192, 256	CVL (Cert. A4390)	Derived internally using the TLS master secret via TLS KDF	NA / Plaintext (MD/EE)	N/A	Not persistently stored	Power cycle or API call	Encryption and decryption of TLS session packets.
TLS integrity key	112 or greater	CVL (Cert. A4390)	Derived internally using the TLS master secret via TLS KDF	NA / Plaintext (MD/EE)	N/A	Not persistently stored	Power cycle or API call	Authentication of TLS session packets.

9.1 Random number generation

The module does not contain an entropy source. The module does not contain a Deterministic Random Bit Generator (DRBG).

9.2 SSP generation

For RSA, DSA, ECDSA and Diffie-Hellman and EC Diffie-Hellman keys, the module imports them using API calls. The module does not generate keys.

Additionally, the module also imports symmetric keys in the same manner for symmetric algorithms.

9.3 SSP Agreement and SSP Derivation

The module provides Diffie-Hellman and EC Diffie-Hellman shared secret computation to obtain “shared secret” values.

The security strength of the preceding algorithms is as follows:

1. Diffie-Hellman key agreement provides between 112 and 200 bits of encryption strength.
2. EC Diffie-Hellman key agreement provides between 112 and 256 bits of encryption strength.

The Diffie-Hellman and EC Diffie-Hellman are under scenario 2 of [IG] D.F.

The module supports key derivation for the TLS protocol. The module implements the pseudo-random function (PRF) for TLSv1.2 using the extended master secret RFC 7627 and the HKDF for TLSv1.3.

9.4 SSP Entry/Output

The keys and SSPs to be entered or exited are provided to the module via API input/output parameters in plaintext form and output via API output parameters in plaintext form.

This is allowed per section 7.9.5 of the ISO/IEC 19790:2012 since all CSPs or key components are maintained within the environment and the requirements from section 7.6.3 are met.

The module does not support either manual key entry or intermediate key generation values.

Additionally, the module implements two independent internal actions in order to prevent the inadvertent output of any plaintext SSP. The mechanism implemented by the module is described below:

First of all, the module reserves the required memory where the CSP will be stored after being derived. Then the CSP is derived using the specific entry point of hardware or software implementation. Finally, the derived CSP is copy into a variable received in a parameter of the invoked function and this last function ends.

Particular information for the input and output for each SSP, if applicable, is provided under column "Import/Export” of Table 9.

9.5 SSP Storage

Symmetric keys, and 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.

No physical storage is offered within the logical boundary, and therefore the module does not store any SSPs persistently beyond the lifetime of the API call. Any persistent key storage occurs outside the module's logical boundary but within the physical perimeter and the management of these keys is responsibility of the calling application.

Table 10 - Storage Areas

Name	Description	Type
RAM	System Memory	Dynamic
ARAM	Intel® QAT Hardware Device accelerator working memory	Dynamic
Shared RAM	Intel® QAT Hardware Device accelerator CryptoEngine memory	Dynamic

Particular information for storage of each SSP, is provided under column "Storage" of Table 9.

9.6 SSP 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 listed in Table 9.

The zeroization functions overwrite the memory occupied by keys with "zeros" and deallocate the memory with the regular memory deallocation operating system call. In case of abnormal termination, or swap in/out of a physical memory page of a process, the keys in physical memory are overwritten by the Linux kernel before the physical memory is allocated to another process.

Particular information for zeroization of each SSP, is provided under column "Zeroization" of Table 9.

The zeroization of the SSPs starts just after the invocation of the zeroization command. Once invoked, these techniques take effect immediately and do not allow sufficient time to compromise any plaintext secret, private keys and CSPs.

During the zeroization process, services are not available, and input and output are inhibited.

10 Self-Tests

FIPS 140-3 requires that the module performs a set of self-test in order to provide the operator assurance that faults have not been introduced that would prevent the module's correct operation.

The module includes two different set of self-test: pre-operational self-test (for software and hardware parts) which are executed prior to the module providing any data output via the data output interface; and conditional self-tests (for software and hardware parts) which are executed in the initialization phase of the module.

The determination of pass or fail of each self-test is made by the module itself, without external controls, externally provided input test vectors, expected output results, or operator intervention.

The following sections list the self-tests performed by the module, their expected error status and error resolutions.

10.1 Pre-operational Self-Tests

The module executes the following pre-operational software and firmware integrity tests. If some of them fail, the module flows to an error state:

Table 11 – Pre-operational integrity Self-Tests

Algorithm	OE	Test Properties	Type	Details
ECDSA	Red Hat Enterprise Linux 9.0	P-256 curves with SHA2-256	KAT	Signature verification for each software library component.
RSA	Intel Corporation Device 4940 (rev 40)	3072 bits key size with SHA2-256	KAT	Signature verification for the firmware components of Intel® QAT Hardware Accelerator device as it is described in section 5.

As to the Intel Authentication Firmware running on Microengine and used to check the integrity of the firmware components (qat_4xxx.bin and qat_4xxx_mmp.bin), no integrity tested is performed because this Authentication Firmware is stored in a non-reconfigurable memory as allowed per FIPS IG 5.A.

While the module is executing the pre-operational self-tests, services are not available, and input and output are inhibited. The module is not available for use by the calling application until the pre-operational tests are completed successfully.

10.2 Conditional Self-Tests

The module executes the following conditional algorithms self-tests at the module initialization phase prior to the pre-operational integrity tests just after an applicable security function is invoked via the available services:

Table 12 - Conditional Self-Tests of Hardware implementation

Algorithm	OE	Test Properties	Type	Details	Condition
RSA	Red Hat Enterprise Linux 9.0 / Intel Corporation Device 4940 (rev 40)	2048 bits key size with SHA2-256	KAT	Signature generation and Signature verification	Power-Up
ECDSA	Red Hat Enterprise Linux 9.0 / Intel Corporation Device 4940 (rev 40)	P-256 and P-384 curves with SHA2-256	KAT	Signature generation and Signature verification	Power-Up
DSA	Red Hat Enterprise Linux 9.0 / Intel Corporation Device 4940 (rev 40)	2048,224 key size with SHA2-256	KAT	Signature generation and Signature verification	Power-Up
ECDH	Red Hat Enterprise Linux 9.0 / Intel Corporation Device 4940 (rev 40)	P-256 and P-384 curves	KAT	Shared Secret computation	Power-Up
DH	Red Hat Enterprise Linux 9.0 / Intel Corporation Device 4940 (rev 40)	ffdhe2048 safe prime group	KAT	Shared Secret computation	Power-Up
AES GCM	Red Hat Enterprise Linux 9.0 / Intel Corporation Device 4940 (rev 40)	256 bits key size	KAT	Encryption and Decryption	Power-Up
SHA3	Red Hat Enterprise Linux 9.0 / Intel Corporation Device 4940 (rev 40)	SHA3-256	KAT	Generation	Power-Up
TLS 1.2 (PRF)	Red Hat Enterprise Linux 9.0 / Intel Corporation Device 4940 (rev 40)	Using SHA2-256 and SHA2-384	KAT	Key Derivation. This self-test is implemented to cover the requirement of self-testing the underlying algorithms, based on the “10.3.B Self-test for Embedded Cryptographic Algorithms” of the IG document	Power-Up
TLS 1.3 (HKDF)	Red Hat Enterprise Linux 9.0 / Intel Corporation Device 4940 (rev 40)	Using SHA2-256 and SHA2-384	KAT	Key Derivation This self-test is implemented to cover the requirement of self-testing the underlying algorithms, based on the “10.3.B Self-test for Embedded Cryptographic Algorithms” of the IG document	Power-Up

Table 13 - Conditional Self-Tests of Software implementation

Algorithm	OE	Test Properties	Type	Details	Conditions
RSA	Red Hat Enterprise Linux 9.0	2048 bits key size with SHA2-256	KAT	Signature generation and Signature verification	Power-Up
ECDSA	Red Hat Enterprise Linux 9.0	P-256 and P-384 curves with SHA2-256	KAT	Signature generation and Signature verification	Power-Up
ECDH	Red Hat Enterprise Linux 9.0	P-256 and P-384 curves	KAT	Shared Secret computation	Power-Up
AES GCM	Red Hat Enterprise Linux 9.0	256 bits key size	KAT	Encryption and Decryption	Power-Up
SHS	Red Hat Enterprise Linux 9.0	SHA2-256 and SJA2-512	KAT	Generation	Power-Up

The module executes the following conditional algorithms self-tests at the module initialization phase prior to the pre-operational integrity tests of the hardware part.

Table 14 - Conditional Self-Tests of Firmware implementation

Algorithm	OE	Test Properties	Type	Details	Conditions
RSA	Intel Corporation Device 4940 (rev 40)	3072 bits key size with SHA2-256	KAT	Signature verification. Executed by checking the integrity test of a "dummy" firmware as allowed per FIPS IG 10.2.A, path 2.	Power-Up
SHS	Intel Corporation Device 4940 (rev 40)	SHA2-256	KAT	Generation. Used to check RSA public key embedded in authentication ROM	Power-Up

While the module is executing the conditional self-tests, services are not available, and input and output are inhibited.

10.3 Error states

The module has two different error states: One error state at software level, “Critical Error (Software)” and one at hardware level, “Critical Error (Hardware)”.

The module can flow to "Critical Error" state of the software part if the pre-operational integrity test or if the conditional algorithms self-tests fail. The only manner to recover from this error is by rebooting the module (only the software part). No CSPs output are available on this state.

The module can flow to "Critical Error" state of the firmware of the hardware part if the pre-operational integrity test or if the conditional algorithms self-tests fail. The only manner to recover from this error is by rebooting the module (only the software part). No CSPs output are available on this state.

In both Error states, the cryptographic functions are not available because they are inhibited as well as the logical interfaces. The only manner to recover cryptographic functionality is to reboot the module and pass the Pre-Operational Integrity self-tests as well as Conditional algorithm self-tests again.

Table 15 - Error States

Name	Description	Conditions	Recovery Method	Indicator
Critical Error (Hardware)	The module reaches to this state if the firmware pre-operational integrity self-tests or conditional algorithms self-tests fail.	Pre-operational integrity self-tests or conditional algorithms self-tests firmware failure	Restart the hardware component of the module.	Integrity check failure and algorithms self-tests failure: "FW integrity self-test failed!"
Critical Error (Software)	The module reaches to this state if the pre-operational integrity self-tests or conditional algorithms self-tests fail.	Pre-operational integrity self-tests or conditional algorithms self-tests failure	Restart the application that exercise the software component of the module.	Integrity check failure: “QAT FIPS Integrity test result: FAIL” Software algorithms self-test failure: “QAT FIPS self-tests(KAT) result: FAIL”

10.4 Operator Initiation Self-Tests

The operator can initiate the execution of the self-test (of the software and hardware algorithms implementation) by powering-off and reloading the module which cause the module to run the pre-operational and conditional algorithms self-test again.

Regarding the hardware component pre-operational and conditional self-tests, they are initiated when the QAT accelerators are loaded and will be ready for providing cryptographic services. “adf_ctl up” command starts all the available QAT accelerators of the processor, and consequently, the pre-operational and conditional self-tests are initiated.

By executing “adf_ctl restart” command, it is possible to execute the same pre-operational and conditional self-tests on-demand.

During the execution of the operator initiation self-tests, services are not available, and no data output or input is possible.

11 Life-cycle Assurance

11.1 Delivery and Operation

The module has been tested in the following operational environments:

- Hardware Platform: Intel Eagle Stream, Operating System: Red Hat Enterprise Linux 9.0 with OpenSSL 3.0.8

For installation, the operator shall follow the guidance below described. The RPM package can be downloaded from Intel GitHub repository.

11.2 Crypto Officer Guidance

The only role allowed in the module is the Crypto Officer who is in charge to perform all the operations and services of the module.

The module only supports one mode of operation: Approved. The module will be in Approved mode when all pre-operational and conditional self-tests have been completed successfully, and only Approved and allowed security functions are invoked. There are no additional installation, configuration, or usage instructions for operators intending to use the module.

11.3 Installation

The operator shall first install the OpenSSL 3.0.8 in a customized path:

- ***. /config -g --prefix=<customized path>***
- ***make -j; make install***

Since the module is already compiled and packed in a “rpm” package, the operator shall run the following command to install it:

- ***rpm -ivh qatprovider-fips-1.3.1-1.el9.x86_64.rpm--target noarch***

The operator shall also execute the following commands in order to finalize the installation of the module:

- ***export LD_LIBRARY_PATH=/<customized path>/lib64***
- ***cp -rf /usr/lib64/ossl-modules/qatprovider.so /<customized path>/lib64/ossl-modules/***
- ***cp -rf /usr/lib64/ossl-modules/qatprovider.la /<customized path>/lib64/ossl-modules/***

11.4 Component versioning identification

By executing the service 'Show version information', an operator can obtain information about the versioning identification for each component of the cryptographic module. The output return by the service execution is as follows (being relevant the information in bold text):

Module Info
Name: QAT Provider FIPS
ID: qatprovider
Version: QAT Engine v1.3.1

QAT_HW Driver version: QAT20.l.1.0.40-00004
IPSec-mb version: v1.3
IPP-crypto version: ippcp_2021.7.1

Additionally, the operator can check the hardware device version embedded in the CPU using the "*lspci*" Linux command. The module has been tested and found compliant on the hardware acceleration device:

Intel Corporation Device 4940 (rev 40).

Using the information, a potential user of the cryptographic module can correlate the module version information with the one included in the FIPS 140-3 certificate.

11.5 Non-reconfigurable memory

The module boundary contains a non-reconfigurable memory where it is stored the "Authentication firmware" that implements the RSA Signature Verification with 3072 bits key size with SHA2-256 and SHA2-256 (both CAVP certified under #A4392 certificate number). Since this firmware is stored in a non-reconfigurable memory created by mechanical means, there is not performed an integrity test to check that it has not been modified. This is allowed per "5.A Non-Reconfigurable Memory Integrity Test" of the IG document.

As included in the "5 Software/Firmware security" section of this document, this firmware is used to check the integrity of the "Production firmware" that runs in the Cryptoengines.

This Non-reconfigurable memory does not store any sensitive information such as SSPs or keys, so there is no need to follow any procedure prior to be distributed to other operators or disposed to remove sensitive information.

12 Mitigation of Other Attacks

This section is not applicable. The module does not claim to mitigate any attacks beyond the FIPS 140-3 Level 1 requirements for this validation.

13 Secure Guidance

13.1 AES-GCM Usage

The module does not implement the TLS protocol itself, however, it provides the cryptographic functions required for implementing the protocol. AES GCM encryption is used in the context of the TLS protocol versions 1.2 and 1.3 (per Scenario 1 and Scenario 5 in FIPS 140-3 C.H respectively). For TLS v1.2, the mechanism for IV generation is compliant with RFC 5288. The counter portion of the IV is strictly increasing. When the IV exhausts the maximum number of possible values for a given session key, this results in a failure in encryption and a handshake to establish a new encryption key will be required. It is the responsibility of the user of the module i.e., the first party, client or server, to encounter this condition, to trigger this handshake in accordance with RFC 5246. For TLS v1.3, the mechanism for IV generation is compliant with RFC 8446.

The IV is at least 96-bits in length per NIST SP 800-38D, Section 8.2.1 so that the (key, IV) pair collision probability is less than $2^{-32.}$ The module uses at least 32 bits of the IV field as a name and use 64 bits as a deterministic non-repetitive counter. When the counter part of the IV exhausts the maximum number of possible values for a given session key the encryptor aborts the session.

In the event that the module power is lost and restored the user must ensure that the AES-GCM encryption/decryption keys are re-distributed.

The module supports importing of GCM IVs when an IV is not generated within the module. In the Approved mode, an IV must not be imported for encryption from outside the cryptographic boundary of the module as this will result in a non-conformance.

13.2 Intel suggestion on use of cryptographic algorithms

Intel recommends following the latest NIST standards when considering the selection of cryptographic algorithms. Consider stronger alternatives for the following algorithms

- DSA (consider ECDSA)
- DH based on finite fields (consider ECDH)
- Hash functions with digests shorter than 256 bits, e.g., SHA3-224
- ECDSA and ECDH on curves with modulus shorter than 256 bits, e.g., P-224, B-233 & K-233

14 Appendix A

The module implements TLS versions 1.2 and 1.3. The AES GCM supported ciphersuites for each version are listed below:

Supported AES GCM ciphersuites for TLS v1.2:

- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384

Supported AES GCM ciphersuites for TLS v1.3:

- TLS_AES_128_GCM_SHA256
- TLS_AES_256_GCM_SHA384

15 References and Acronyms

15.1 References

Table 16 - References

Abbreviation	Full Specification Name
FIPS 140-3	FIPS 140-3 Security Requirements for Cryptographic modules
NIST SP 800-140	FIPS 140-3 Derived Test Requirements (DTR): CMVP Validation Authority Updates to ISO/IEC 24759
ISO 19790	ISO/IEC 19790:2012/Cor.1:2015(E), Information technology — Security techniques — Security requirements for cryptographic modules
ISO 24759	ISO/IEC 24759:2017(E), Information technology — Security techniques — Test requirements for cryptographic modules
IG	Implementation Guidance for FIPS 140-3 and the Cryptographic Module Validation Program
FIPS PUB 197	FIPS 197 Advanced Encryption Standard
FIPS PUB 180-4	FIPS 180-4 Secure Hash Standard
FIPS PUB 186-4	FIPS 186-4 Digital Signature Standard (DSS)
FIPS PUB 202	FIPS 202 SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions
NIST SP 800-38A	NIST SP 800-38A, Recommendation for Block Cipher Modes of Operation Methods and Techniques
NIST SP 800-38D	NIST SP 800-38D, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC
NIST SP 800-56Arev3	NIST SP 800-56Arev3, Recommendation for Pair-Wise Key Establishment Schemes using Discrete Logarithm Cryptography
NIST SP 800-131Arev2	NIST SP 800-131Arev2, Transitioning the Use of Cryptographic Algorithms and Key Lengths
NIST SP 800-135rev1	NIST SP 800-135rev1, Recommendation for Existing Application-Specific Key Derivation Functions
RFC 7627	Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension
RFC 8846	The Transport Layer Security (TLS) Protocol Version 1.3

15.2 Acronyms

Table 17 - Acronyms definitions

Acronym	Definition
AES	Advanced Encryption Standard
API	Application Program Interface

Acronym	Definition
CAVP	Cryptographic Algorithm Validation Program
CMVP	Cryptographic Module Validation Program
CVL	Component Validation List
DH	Diffie-Hellman
DSA	Digital Signature Algorithm
DRGB	Deterministic Random Bit Generator
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
FFC	Finite Field Cryptographic
FIPS	Federal Information Processing Standards Publication
GCM	Galois Counter Mode
GPC	General Purpose Computer
KAS	Key Agreement Scheme
KAT	Known Answer Test
KDF	Key Derivation Function
MAC	Message Authentication Code
NIST	National Institute of Science and Technology
QAT	QuickAssist Technology
RSA	Rivest, Shamir, Addleman
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
SSC	Shared Secret Computation
TLS	Transport Layer Security