QTI Cryptographic Module on Crypto 5 Core

FIPS 140-2 Non-Proprietary Security Policy

Version: 2.4

2015-11-18

Prepared for:

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121

Prepared by:

atsec information security Corp.
9130 Jollyville Road, Suite 260
Austin, TX 78759

Table of Contents

Copyrights and Trademarks ... 3
1. Introduction ... 4
1.1. Purpose of the Security Policy ... 4
2. Cryptographic Module Specification ... 5
2.1. Module description ... 5
2.1.1. Hardware description ... 6
2.1.2. Software description ... 6
2.1.3. Module Validation Level ... 7
2.2. Description of Approved modes ... 8
2.3. Cryptographic Module Boundary ... 9
2.3.1. Hardware Block Diagram ... 10
3. Cryptographic Module Ports and Interfaces ... 12
4. Roles, Services and Authentication ... 13
4.1. Roles ... 13
4.2. Services ... 13
5. Physical Security ... 17
5.1. Type ... 17
6. Operational Environment ... 18
6.1. Applicability ... 18
6.2. Debugger ... 18
7. Cryptographic Key Management ... 19
7.1. Random Number Generation ... 19
7.2. Key/CSP Generation Management ... 19
8. Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC) ... 21
9. Powerup Tests ... 22
9.1. Self-tests ... 23
9.2. Cryptographic algorithm tests (known answer tests) ... 23
9.3. Integrity Test Summary ... 24
9.4. Conditional Tests ... 25
10. Design Assurance ... 26
10.1. Configuration Management ... 26
10.1.1. Software ... 26
10.1.2. Hardware ... 26
10.2. Delivery and Operation ... 26
10.3. Guidance ... 26
10.3.1. Crypto Officer Guidance ... 26
11. User Guidance ... 28
11.1. Application Programming Interfaces ... 28
11.2. User Space APIs ... 28
11.3. Kernel Space APIs ... 29
11.3.1. Block Cipher Algorithms ... 29

11.3.2. Hash/HMAC Algorithms 30
11.3.3. Status 30
12. Mitigation of Other Attacks 32
13. Terms and Abbreviations 33

Copyrights and Trademarks

1. Introduction

This document is a *FIPS 140-2 Security Policy* for the QTI Cryptographic Module on Crypto 5 Core. It contains a specification of the rules under which the module must operate and describes how this module meets the requirements as specified in Federal Information Processing Standards Publication 140-2 (FIPS PUB 140-2) for a Security Level 1 module. It is intended for the FIPS 140-2 testing lab, Cryptographic Module Validation Program (CMVP), developers working on the release, administrators of the cryptographic module and users of the cryptographic module.

For more information about the FIPS 140-2 standard and validation program, refer to the NIST website at http://csrc.nist.gov/groups/STM/cmvp/index.html.

In this document, the terms “QTI Cryptographic Module on Crypto 5 Core”, “cryptographic module”, “QTI Cryptographic Module” or “the module” are used interchangeably to refer to the QTI Cryptographic Module on Crypto 5 Core.

1.1.Purpose of the Security Policy

There are three major reasons that a security policy is required

- It is required for FIPS 140-2 validation.
- It allows individuals and organizations to determine whether the implemented cryptographic module satisfies the stated security policy.
- It allows individuals and organizations to determine whether the described capabilities, the level of protection, and access rights provided by the cryptographic module meet their security requirements.

2. Cryptographic Module Specification

2.1.Module description

The Cryptographic Module (CM) is a multi-chip standalone software-hybrid module. The cryptographic services provided by the module are:

- Data encryption / decryption utilizing symmetric ciphers, i.*e.*Triple-DES, and AES algorithms.
- Random number generation based on a SP 800-90A CTR DRBG with a 128-bit AES derivation function.
- Computation of hash values, i.*e.* SHA-1, SHA-256.
- Message authentication utilizing HMAC-SHA1, HMAC-SHA256, AES CMAC, hashing algorithms.
- Concurrent operations of hashing and ciphering using AES CCM.

Table 2-1: Summary of FIPS and non FIPS algorithm in CM

FIPS Compliant	Implemented Algorithms	Kernel Invokes As (.cra_driver_name)
AES-128 CBC, AES-256 CBC	encryption, decryption	qcrypto-cbc-aes
AES-128 ECB, AES-256 ECB	encryption, decryption	qcrypto-ecb-aes
AES-128 CTR, AES-256 CTR	encryption, decryption	qcrypto-ctr-aes
AES-128 CCM, AES-256 CCM[1]	encryption, decryption (with message authentication code)	qcrypto-ccm-aes
AES-128 XTS, AES-256 XTS	encryption, decryption (with message authentication code)	qcrypto-xts-aes
Triple-DES CBC	encryption, decryption	qcrypto-cbc-3des
Triple-DES ECB	encryption, decryption	qcrypto-ecb-3des
SHA-1	Hashing	qcrypto-sha1
SHA256	Hashing	qcrypto-sha256
HMAC-SHA-1	message authentication code	qcrypto-hamc-sha1
HMAC-SHA-256	message authentication code	qcrypto-hamc-sha256
AES-CMAC[2]	message authentication code	n/a
DRBG	random number generation	n/a
Non-FIPS Compliant	**Implemented Algorithms**	**Kernel Invokes As (.cra_driver_name)**
DES CBC	encryption, decryption	qcrypto-cbc-des
DES ECB	encryption, decryption	qcrypto-ecb-des
AEAD-SHA-1 AES CBC[1]	encryption, decryption (with message authentication code)	qcrypto-aead-hmac-sha1-cbc-aes
AEAD-SHA-1 AES CTR[1]	encryption, decryption (with message authentication code)	qcrypto-aead-hmac-sha1-ctr-aes
AEAD-SHA-1 DES CBC[1]	encryption, decryption (with message authentication code)	qcrypto-aead-hmac-sha1-cbc-des

[1] Only available in the kernel interface.
[2] Only available in the user interface.

AEAD-SHA-1 Triple-DES CBC[1]	encryption, decryption (with message authentication code)	qcrypto-aead-hmac-sha1-cbc-Triple-DES
Kasumi	encryption, decryption (with message authentication code)	n/a
snow-3g	encryption, decryption	n/a
HW RNG	Random number generation	n/a

1. Crypto 5 hardware also provides Over The Air (OTA) f8/f9 algorithms as defined by the 3GPP forum (non-FIPS compliance).
2. Caveat: AES counter mode uses a 128 bit counter. The counter will roll over after 2^128 blocks of encrypted data.

2.1.1.Hardware description

The hardware portion of the cryptographic module is the Crypto 5 Core v5.3.0, which resides in Snapdragon 810 processors (https://www.qualcomm.com/products/snapdragon/processors/810). The QTI Crypto 5 Core v5.3.0 provides a series of algorithms (as listed in Table 2-1) implemented in the device hardware.

2.1.2.Software description

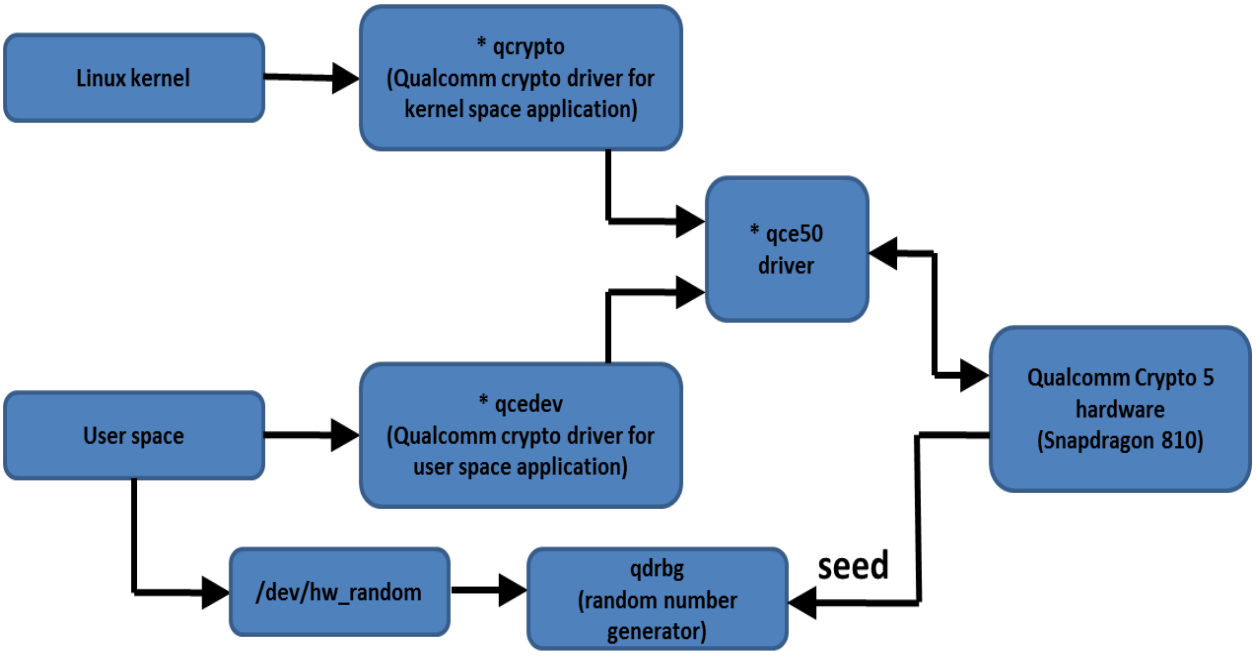
The software portion of the cryptographic module consists of the QTI crypto engine (qce50) driver V5.f3-64 that provides common services for accessing the hardware Crypto 5 Core v5.3.0 through the following three clients:

- qcrypto driver (module provided for accessing CM by kernel space apps)
- qcedev driver (module provided for accessing CM by user space apps)
- qdrbg provides random numbers to user space apps

The qcedev, qcrypto and qceXX executables are contained within the qcrypto_module.ko kernel module and the qdrbg executable is contained within the qdrbg_module.ko kernel module.

All actual cryptographic functions, with the exception of the DRBG, are implemented within the hardware.

Figure 1: Cryptographic modules depicting the main driver modules, qcrypto, qce50, qcedev, and qdrbg containing services to access the QTI Crypto 5 hardware



The qceXX software driver is independent of the platform. The qceXX (where XX is 50 for the current major version) driver provides the common services for hardware crypto access to the two drivers listed above. The qdrbg module implements an SP 800-90A compliant DRBG obtaining its seed from the Crypto 5 hardware.

The software application (user space), *qfintverify*, provides integrity checking for itself and the cryptographic software drivers.

2.1.3.Module Validation Level

The module is intended to meet requirements of FIPS 140-2 security level 1 overall. The following table shows the security level claimed for each of the eleven sections that comprise the validation:

Table 2-2: Security Levels

FIPS 140-2 Sections	Security Level				
	N/A	1	2	3	4
Cryptographic Module Specification		✔			
Cryptographic Module Ports and Interfaces		✔			
Roles, Services and Authentication		✔			
Finite State Model		✔			
Physical Security		✔			
Operational Environment		✔			
Cryptographic Key Management		✔			
EMI/EMC				✔	

FIPS 140-2 Sections	Security Level				
	N/A	1	2	3	4
Self Tests		✔			
Design Assurance		✔			
Mitigation of Other Attacks	✔				

The QTI Cryptographic Module is classified as a multi-chip standalone hybrid module for FIPS 140-2 purposes. The logical cryptographic boundary for the module includes the CM software backed by hardware cryptography to support hardware-based cryptographic algorithms.

Table 2-3 describes the platform where the cryptographic module has been tested:

Table 2-3: Tested Platforms

Module Name	Hardware Version/Test Platform	Software Version	O/S (Android) Version
QTI Cryptographic Module on Crypto 5 Core V5.3.0	Qualcomm Snapdragon 810 processor	5.f3-64	Android: 5.0

Table 2-4 describes the software that comprises the cryptographic module. Note that the qcedev, qcrypto and qceXX executables are contained within the qcrypto_module.ko kernel module and the qdrbg executable is contained within the qdrbg_module.ko kernel module.

Table 2-4: Tested Software

Software	Change ID in GIT Version Control System	Description
qcrypto_module.ko	I6600f352a8404c6391584f298fa482bc66309a14	Allows access to the CM from both user mode (via /dev/qce) and kernel mode (via the Linux kernel crypto API)
qfintverify	Ib7105a495492199542fa43cdfbc8125060e7df6c Ia286e86508c9258518f8b8d4f2d9dc0fa961c28d Ibf1db89060b944150d1a916645bfc886c80c6c5b	Performs integrity checks
qdrbg_module.ko	I6600f352a8404c6391584f298fa482bc66309a14	Random number generator

In addition to the configurations tested by the laboratory, vendor affirmed testing was performed using the same cryptographic hardware and software (e.g., the QTI Cryptographic Module on Crypto 5 Core v5.3.0 with the Software version 5.f3-64) on the following configuration:

- Android 5.0 on the Qualcomm Snapdragon 808 processor
- Android 6.0 on the Qualcomm Snapdragon 808 processor
- Android 6.0 on the Qualcomm Snapdragon 810 processor

2.2.Description of Approved modes

The CM supports a FIPS approved mode and a non-approved mode. All CSPs are kept separate between the two modes. The services available in each mode are specified in table 2-1. When a request for a non-approved mode service is received, the CM switches to non-approved mode, services the request, and immediately switches back to the approved mode.

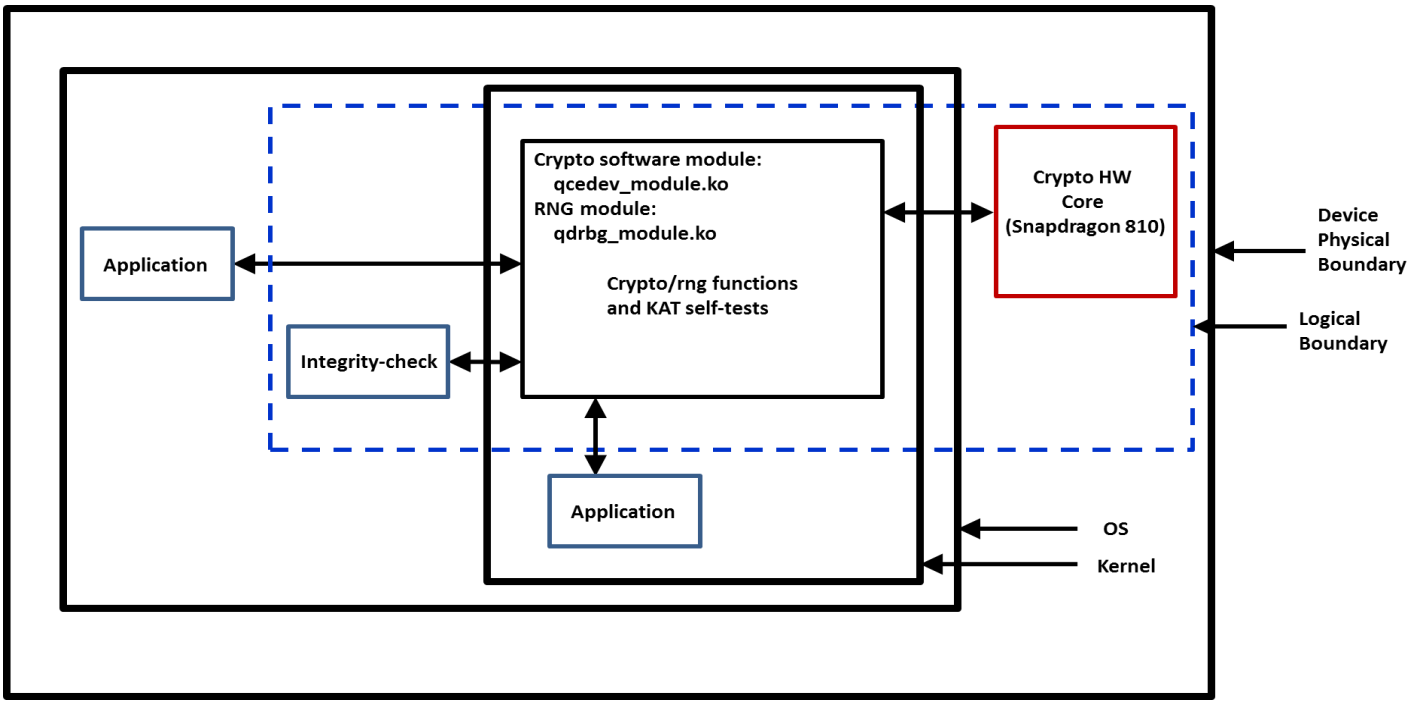
The CM is placed into the approved mode by performing power up self-tests consisting of a KAT self-test for each algorithm in the approved list and an integrity test. If either test fails, none of the cryptographic functions are available.

Table 2-1 provides a summary of all security functions (both FIPS compliant and FIPS non-compliant). Table 4-2, 4-3 and table 4-4 illustrate the role and corresponding services available to the Crypto officer and User.

2.3.Cryptographic Module Boundary

The physical boundary of the module is the physical boundary of the device that contains the module. Consequently, the embodiment of the module is a Multi-chip standalone cryptographic module.
In the following diagram, the bidirectional arrows depict the flow of the status, control and data. The logical boundary of the cryptographic module (in **blue**) overlaps both user space and kernel space.

Figure 2: Cryptographic Boundary



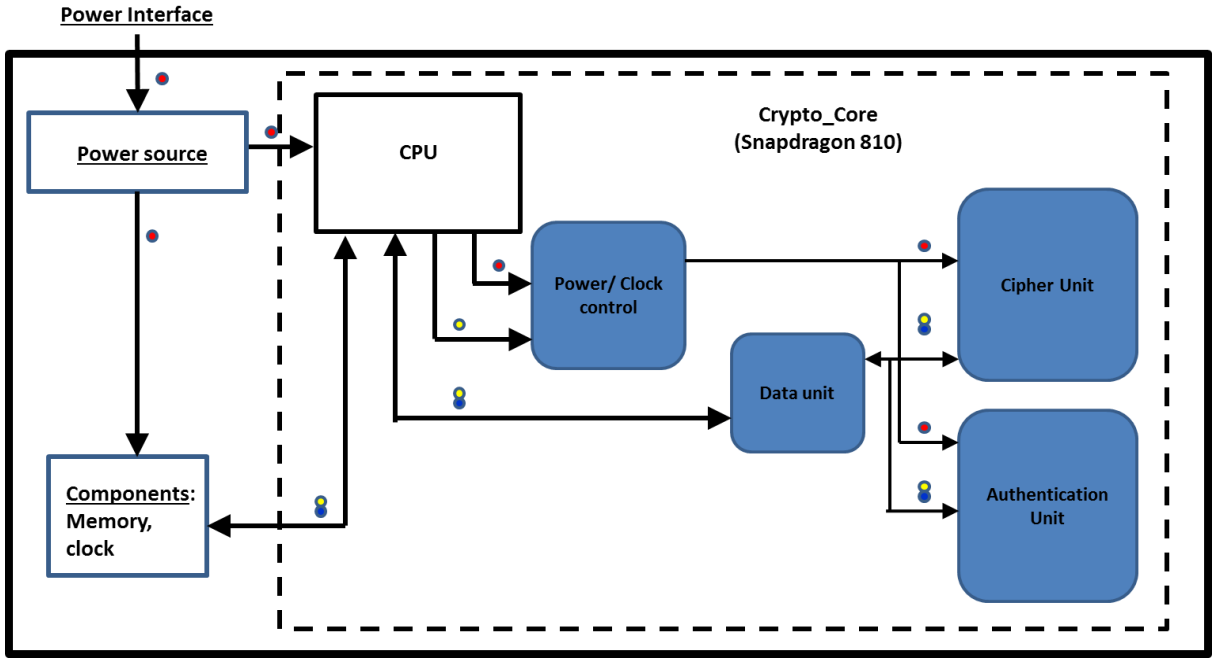
The operations within the logical security boundary (the blue dotted region) use the ciphers from the hardware, Crypto Core (see *Figure 3*) that is included in the logical boundary.

2.3.1.Hardware Block Diagram

The bidirectional arrows depict the flow of the status, control and data. Software drivers pass the parameters to the hardware and receive the results from the hardware.

The CSPs, such as the encryption key, are given to the APIs. The hardware components which are not part of the crypto core pass the Critical Security Parameter (CSP) to crypto core. Therefore, they act as “pipes” without interaction with the CSPs.

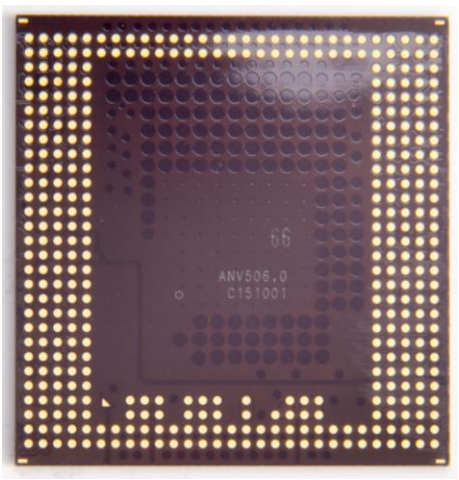
Figure 3: Hardware Block Diagram



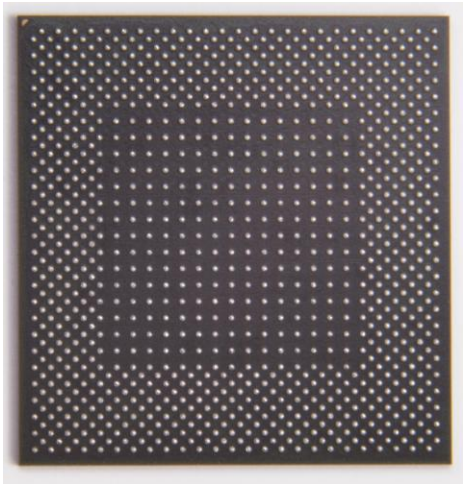
: Power in
: Command/ data/ CSP in
: Data/ status/ CSP out

The CSPs are passed via memory and processed by the Cryptographic Module. All parameters to the hardware are provided via memory from qce50.

Figure 4: Qualcomm Snapdragon 810 processor



Back view



Front view

3.Cryptographic Module Ports and Interfaces

Table 3-1 Ports and interfaces

FIPS Interface	Ports
Data Input	API input parameters
Data Output	API output parameters
Control Input	API function calls
Status Output	API return codes, kernel log messages
Power Input	Physical power connector

As indicated in Table 3-1, all status ports and control ports are directed through the interface of the module's logical boundary, which is the software APIs of the module. .

The User or Crypto Officer interacts with the CM in two distinct ways:

1. Powering on the CM
2. The application services (API's) invoked by users

For the application services, the logical interfaces of the module are the C language Kernel Programming Interfaces (KPIs). In detail these interfaces are the following:

- Data input and data output are provided in the variables passed in the KPI and callable service invocations, generally through caller-supplied buffers.
- Control inputs which control the mode of the module are provided through dedicated parameters.
- Status output is provided in return codes and through messages. Documentation for each KPI lists possible return codes. A complete list of all return codes returned by the C language KPIs within the module is provided in the header files and the KPI documentation. Messages are documented also in the KPI documentation. Return codes will be in the header files.

Once the kernel initializes the cryptographic module and the self-tests complete successfully, all cryptographic functions are made available to kernel services and user space. If any of the modules KAT fails, the module self-test causes the system to panic (see Section 9.1 for more details). To recover from a KAT failure a reboot of the kernel is required to reset the failure status. If the integrity tests fail the module causes the kernel to panic, thereby shutting down the kernel and the CM. The only way to recover from an integrity test failure is to reinstall the software and reboot the kernel.

Caller-induced or internal errors do not reveal any sensitive material to callers. Cryptographic bypass capability is not supported by the module. The CM ensures that there is no means to obtain data from the cryptographic module by performing key zeroization. There is no means to obtain sensitive information from the cryptographic module.

4.Roles, Services and Authentication

4.1.Roles

The module supports two roles: a Crypto Officer role and a User role. The module does not support user identification or authentication that would allow the module to distinguish between the two supported roles.

The Crypto Officer role is a purely administrative role that does not involve the use of cryptographic services. The role is not explicitly authenticated but assumed implicitly on implementation of the module's installation and initialization.

The User role has access to all of the module's services. The role is not explicitly authenticated, but assumed implicitly on access of any of the non-Crypto Officer services. An operator is implicitly in the User or Crypto Officer role based upon the service chosen. If any of the User-specific services are called, then the operator is in the User role; otherwise the operator is in the Crypto Officer role.

The customer who configures and installs the CM is considered to be a Crypto Officer. Additionally, a user powering up and initializing the device is assuming the Crypto Officer role. There is no enforced separation between the Crypto Officer and the User roles. As stated previously, the User or Crypto Officer role is based on the service or operation performed.

Table 4-1 Roles

Role	Services (see Table 4-2)
User	Utilization of services of the module
Crypto Officer	Installation and initialization of the module.

4.2.Services

The crypto module does not provide a bypass capability through which some cryptographic operations are not performed or where certain controls implemented during normal operation are not enforced.

Services and sub-services are split between the software and hardware portions of the software hybrid module as follows:

- DRBG: software implementation using the cipher primitives from hardware
- all other ciphers: hardware where the software driver is used for proper alignment of data and invocation of hardware
- selftest / integrity test: software

The following tables (Table 4-2 and Table 4-3) illustrate the role and corresponding services of the Crypto Officer and User. Additionally, the Component column indicates the component that implements the service with ‘H’ indicating hardware and ‘S’ indicating software.

Table 4-2 Approved Services

Service	Roles		CSP	Component	Modes	Is FIPS Approved? If Yes Cert #	Access (Read, Write, Execute)	Standard
	User	CO						
Symmetric Algorithms								
AES encryption and decryption	✓		AES Symmetric key (128, 256 bit)	H	CBC, ECB, CTR, XTS, CCM (only available on the qcrypto interface (kernel-kernel))	Cert. 3164	Read/Write	FIPS 197 SP 800-38 [A,C,E]
Triple-DES	✓		Triple DES Symmetric key (168 bits)	H	CBC, ECB	Cert. 1802	Read/Write	FIPS 46-3 SP 800-38A
Hash Functions								
SHA-1	✓		None	H	N/A	Cert. 2617	Read	FIPS 180-4
SHA-256	✓		None	H	N/A	Cert. 2617	Read	FIPS 180-4
Message Authentication Codes (MACs)								
HMAC-SHA1	✓		HMAC-SHA1 key (key length at least 112 bits)	H	N/A	Cert. 1992	Write	FIPS 198-1
HMAC-SHA-256	✓		HMAC-SHA256 key with a length of at least 112 bits	H	N/A	Cert. 1992	Write	FIPS 198-1

Service	Roles		CSP	Component	Modes	Is FIPS Approved? If Yes Cert #	Access (Read, Write, Execute)	Standard
	User	CO						
AES-CMAC	✓		AES Symmetric key (128, 256 bit)	H	CMAC (only available on the qcedev interface (user-kernel))	Cert. 3164	Write	SP 800-38B
Random Number Generation								
DRBG 800-90A	✓		Seed, Nonce	S	CTR DRBG with AES 128 core and derivation function	Cert. 655	Read	SP 800-90A
Miscellaneous								
Module installation		✓	None	N/A	N/A	N/A	Read/Write	N/A
Self Tests	✓		HMAC-SHA-256 key for integrity test	S	N/A	Cert. 1992	Read/Execute	N/A
Zeroization	✓		All CSPs	S	N/A	N/A	Read/Write/Execute	N/A
Query status	✓		None	S	N/A	N/A	Read	N/A

Table 4-3 Non-Approved Services

Service	Roles		CSP	Modes	Access (Read, Write, Execute)	Standard
	User	CO				
Symmetric Algorithms						
DES	✓		DES Symmetric key (56 bits)	CBC, ECB	Read/Write	n/a

Service	Roles		CSP	Modes	Access (Read, Write, Execute)	Standard
	User	CO				
AEAD-SHA-1 AES[3]	✓		AES Symmetric key (128 bits), HMAC-SHA1 key (key length at least 112 bits)	CBC, CTR	Read/Write	n/a
AEAD-SHA-1 DES[3]	✓		DES Symmetric key (56 bits), HMAC-SHA1 key (key length at least 112 bits)	CBC	Read/Write	n/a
AEAD-SHA-1 Triple-DES[3]	✓		Triple-DES Symmetric key (168 bits), HMAC-SHA1 key (key length at least 112 bits)	CBC	Read/Write	n/a
Kasumi	✓		Symmetric key (128 bits)	n/a	Read/Write	n/a
Snow-3g	✓		Symmetric key (128, 256 bits)	n/a	Read/Write	n/a
Random Number Generation						
HW RNG	✓		Entropy source, seed	Combination of True RNG and HASH-DRBG	Read	

[3] Also produces a Message Authentication Code

5.Physical Security

5.1.Type

The QTI Cryptographic Module is a software-hybrid module that operates on a multi-chip standalone platform which conforms to the Level 1 requirements for physical security. The hardware portion of the cryptographic module is a production grade component. The cryptographic module must be used in a commercial off the shelf (COTS) mobile device. The mobile device shall be comprised of production grade components with standard passivation (a sealing coat applied over the chip circuitry to protect it against environmental and other physical damage) and a production grade enclosure that completely surrounds the cryptographic module.

6.Operational Environment

6.1.Applicability

The operating system shall be restricted to a single operator mode of operation.
The procurement, build and configuring procedure are controlled. The module is installed into a commercial off-the-shelf (COTS) mobile device by the customer. Therefore the operational environment is considered non-modifiable.

6.2.Debugger

The use of a kernel debugger as well as a JTAG interface for debugging is prohibited.

7.Cryptographic Key Management

7.1.Random Number Generation

The random number generation uses a hardware/software hybrid solution. Hardware is used to collect random bits as the entropy seed for the software module to generate FIPS140-2 compliant random numbers.

The software deterministic random bits generator (DRBG) gets entropy and nonce from the hardware RNG. The software DRBG used to generate pseudo random numbers is an SP 800-90A compliant CTR DRBG with an AES 128 core using a derivation function without prediction resistance. It does not process a personalization string or an additional input string. The implementation performs a continuous self-test, a health check, and a poweron self-test.

The DRBG accepts 128 bits of entropy input and 64 bits of nonce as the seed from the underlying HW RNG. A re-seed process is applied to the DRBG, after it generates 2^31 blocks of data.

When the DRBG is instantiated, it runs a self-test with a set of test vectors, and also runs a health check test to verify that the instantiation function and generation function is able to handle any incorrect parameter inputs, such as the input data length being a negative number, etc. The DRBG implements a continuous self-test verifying the random number generation. The output of the hardware RNG is subject to a continuous self-test.

The DRBG self-test and run-time test performs the following steps:

(1) Instantiate DRBG.
(2) Reseed.
(3) Generate random bits (do not output).
(4) Generate random bits (as outputs).
(5) Un-Instantiate.

The self-test compares the output bits with pre-set expected output bits, from pre-set test vectors. In the run-time test, the output bits will be output to client for further verification by the client.

7.2.Key/CSP Generation Management

The module does not perform key generation for any of its approved algorithms. However, a user may use the modules DRBG to obtain key material to pass into the module. Keys are passed in from clients via algorithm APIs.

There is no manual seeding of the DRBG functions. For additional information about the DRBG refer to Section 7.1.

The CM does not provide any asymmetrical algorithms. Manual key entry or key output capabilities are not provided. All CSPs can only be exchanged between the CM and the calling application or kernel code through the appropriate API calls which all occur within the physical boundary of the device and therefore may be passed from the application to the module in the clear.

Application and kernel code pass references to keys and similar sensitive information to the CM using API calls. It is the applications responsibility to destroy this information using FIPS Pub 140-2 compliant procedures. The CM itself does not destroy externally stored keys and secrets since it does not own these CSPs. Keys are not stored within the CM; however intermediate key material

may reside in memory as clear data. All intermediate key storage and other CSPs stored or created by the CM are zeroized when processing completes.

7.3.Zeroization

Services can be called either by the kernel or the application running in the user space. When the service is called by the kernel, the kernel must initially obtain a cipher handle in which all data including CSPs are located. When complete, the kernel calls the Zeroization service which overwrites all CSPs with zeros before it releases the handle.

When the service is called by the application running in the user mode, the IOCTL switches the control from user mode to the kernel mode and the zeroization is handled by the kernel upon the completion of the service in the same way as described above. After zeroization, the IOCTL returns the control to the calling application in user mode.

8.Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)

The CM was tested for EMI/EMC and found to be compliant with FCC Part 15 subpart B (Household and Medical devices).

Lab Name:
UL Verification Services Inc.
47173 Benica Street
Freemont, CA 94538 U.S.A
NVLAP Lab Code 200065-0

9.Powerup Tests

Powerup self-tests consist of software integrity tests and known-answer tests of algorithm implementations. The module integrity test is automatically performed during loading. If any of these tests fail, the module will terminate the loading process. The module cannot be used in this state. To recover from the error state, re-initialization is possible by doing a reboot to set it to power on state.

Powerup self-tests are performed when the CM is initialized. Initialization is invoked by an application that is human user independent. All self-tests are performed as a single atomic action that has two possible results: success or failure. If the result is success, the CM becomes operational, if it is failure, the CM enters an error state and cryptographic functions cannot be performed.

FIPS 140-2 explicitly allows that the on-demand test can be fulfilled with a power cycle of the module. Hence, a power cycle and its associated poweron self-test is the methodology used to perform the "on-demand" tests.

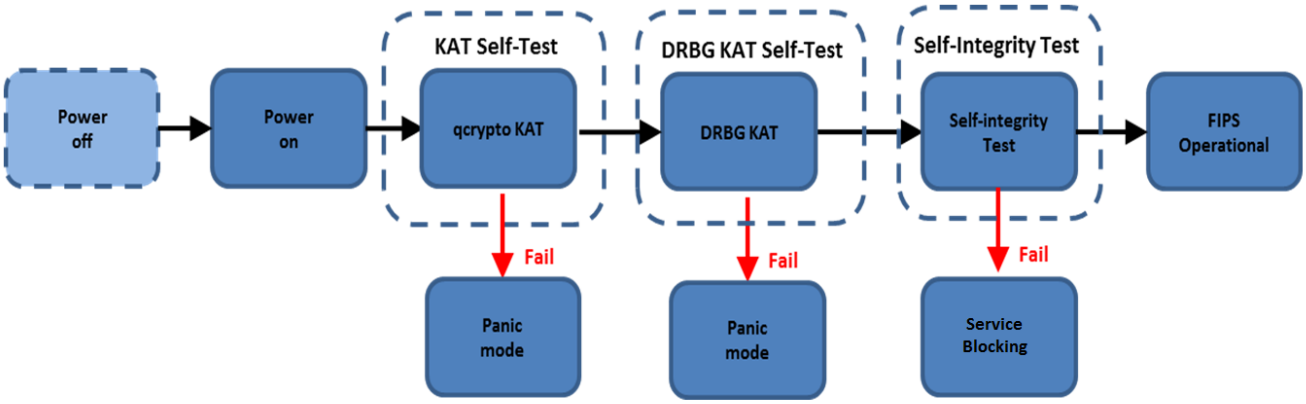
The KAT self-tests are performed during loading by the probe functions which are part of the module init phase in the qcrypto module. If any of the tests fail, the module will cause the system to panic and all functionality to stop.

The DRBG KAT is performed after the successful completion of the KAT self-test If the test fails, the module will cause the system to panic shutting down both the kernel and the CM.

The integrity test is performed after successful completion of the DRBG KAT by a user-space app which is executed in the Linux kernel initialization phase. The *qfintverify* performs integrity tests on the *qcrypto* and *qdrbg* modules and also on itself. All cryptographic requests other than those made by the integrity test module are blocked during the integrity test. The integrity test uses cryptographic functions in the CM to perform the integrity test. If the integrity test of any component fails, all the services are blocked and no crypto operations are available. If the test passes the CM will unblock the cryptographic requests. This causes the CM to enter FIPS operational mode.

Figure 5 shows the CM powerup test flow.

Figure 5 Crypto module finite state powerup tests flow



9.1.Self-tests

The CM implements self-tests to ensure proper functioning of the module Implemented self-tests include powerup self-tests and conditional self-tests. A continuous random number generator test is implemented as a conditional self-test. The powerup self-tests include known answer tests (KAT) and integrity tests of the CM binaries.

There are four values for a global flag used to drive the powerup self-tests:

- 1 - Known answer tests have succeeded
- 2 - Integrity tests have succeeded
- 3 - All tests have succeeded and FIPS 140-2 mode is enabled
- 255 - This is the initial value and each successful stage of the poweron self-tests will change it to one of the previous values. If the integrity test fails the flag is set to this value that causes all cryptographic functions to be unavailable.

The flag value is initialized to 255 which will cause all requests to the CM to be blocked. The powerup self-tests proceed as follows:

- The qcrypto module runs its KATs. If successful then the flag value is set to “1” and success is logged to dmesg. If the KATs fail the code panics the kernel causing the kernel and therefore the entire cryptographic module to shut down.
- The DRBG KAT is performed. If successful the flag value is set to “2” and a success is logged to dmesg. If the KAT fails the code panics the kernel causing the kernel and therefore the entire cryptographic module to shut down.
- The qfinteverify binary performs an integrity test on itself and the kernel image that verifies the integrity of the CM binaries. If unsuccessful, the code panics the kernel causing the kernel and therefore the entire cryptographic module to shut down. If successful, the flag value is set to 3 which causing the CM to no longer block and the CM enters its FIPS 140-2 operational state and the success is logged to dmesg.

9.2.Cryptographic algorithm tests (known answer tests)

Table 9-1 Powerup Tests

Algorithm	**Test**
AES encryption	KAT
AES decryption	KAT
Triple-DES encryption	KAT
Triple-DES decryption	KAT
SHA1	KAT
SHA256	KAT
HMAC-SHA-1	KAT
HMAC-SHA-256	KAT
AES-CMAC	KAT

Algorithm	Test
DRBG	KAT

9.3.Integrity Test Summary

1. During the build process The HMAC SHA-256 of the (qcrypto_module.ko and qdrbg_module.ko images and the HMAC SHA-256 of the integrity application (*qfintverify* executable) are created and stored in relevant files. These files are made available in a read only partition on the target which are:
   - /sbin/qcrypto_module.hmac
   - /sbin/qdrbg_module.hmac
   - /sbin/qfintverify.hmac

   a. The key that is required to create the HMAC SHA-256 signatures is hard-coded in the Makefile of the integrity test project. The HMACs are generated during build compilation time. The key is also hard coded in the program *qfintverify.c* to allow generation of the HMAC SHA-256 values during bootup.

2. On the target: The user-space application *qfintverify* is run as a service when the device boots up.
   The integrity test has three parts:
   - qcrypto_module.ko verification

   a. The program reads raw bytes from /sbin/qcrypto_module.ko
   b. The qcrypto_module.ko is read and stored in a temporary buffer
   c. This temporary buffer is sent to qcyrpto through the standard Linux kernel crypto APIand the HMAC SHA-256 signature of the buffer is created using QCOM hardware
   d. In qfinteverify, the created HMAC digest is compared to the HMAC digest that is stored in /sbin/qcrypto_module.hmac.
   e. If the HMAC digests match, the test will proceed to perform the qdrbg_module.ko verification otherwise it enters the error state.

   ii. qdrbg_module.ko verification
   a. The program reads raw bytes from /sbin/qdrbg_module.ko.
   b. The qdrbg_module.ko data are stored in a temporary buffer.
   c. This temporary buffer is sent to qdrbg through standard Linux kernel crypto API and the HMAC SHA-256 signature of the buffer is created using QCOM hardware.
   d. In qfinteverify, the created HMAC digest is compared to the HMAC digest that is stored in qdrbg_module.hmac.
   e. If the HMAC digests match, the test will proceed to perform the integrity application verification otherwise it enters the error state.

   - self-integrity kernel module verification

   a. The module reads raw bytes from the application /sbin/qfintverify which is stored in a temporary buffer
   b. The temporary buffer is sent to qcedev via an IOCTL and the HMAC SHA-256 signature of the buffer is created using hardware.
   c. In qfinteverify, the created HMAC is compared with the HMAC that is stored in /sbin/qfinteverify.hmac.
   d. If this portion of the integrity test passed, the CM proceeds to perform the DRBG KAT tests, otherwise it enters the error state.

3. The integrity test application (*qfintverify*) is run once every bootup .

If the HMACs match, the integrity tests are considered a success and the DRBG KAT is invoked. If any part of the integrity test fails, the code panics the kernel causing the kernel and therefore the cryptographic module to shut down.. Both the CM binary integrity verification and the self integrity module test have to pass in order for the whole integrity test to succeed.

9.4.Conditional Tests

Each time entropy is requested from the hardware RNG, a 256 bit block is returned. The block from the first request for entropy is not used but rather stored for comparison purposes. From that point when a request for entropy is made, the 256 bit block returned is compared to the previous 256 bit block that was stored. If the comparison indicates the blocks are equal the test fails, and the CM goes into a panic mode forcing the kernel and the CM to shut down.

Similarly, when a request is made for random data from the DRBG, a 128 bit block is generated. The first request is not returned but stored for comparison purposes. When additional requests are made, the current 128 bit block is compared to the stored 128 bit block. If the comparison shows that the blocks are the same, the test fails, the CM causes the system to panic and all functionality stops. Otherwise, when then blocks are different, the conditional test passes and the DRBG will continue to output random bits to its client.

The conditional test also runs during device bootup and the DRBG driver initialization phase. After successful completion of the DRBG continuous test, the FIPS global status flag is set to ”3” and the build is in FIPS 140 -2 mode.

Table 9-2 Conditional tests

Algorithm	**Test**
DRBG	Continuous test
HW RNG (non-FIPS compliant)	Continuous test

10.Design Assurance

10.1.Configuration Management

10.1.1.Software

GIT is used for software (Android) source control and configuration management. GIT is an open source distributed revision control and SCM system. Every GIT working directory is a repository with complete history and full version tracking capabilities, not dependent on network access or a central server. The software (Android) code is in open source domain and is maintained by the Code Aurora Forum (CAF).

10.1.2.Hardware

ClearCase, a version control system from IBM/Rational, is used to manage the revision control of the hardware code (Verilog code) and hardware documentation. The ClearCase version control system provides version control, workspace management, parallel development support, and build auditing. The Verilog code is maintained within the QTI ClearCase database.

10.2.Delivery and Operation

The cryptographic module is delivered via GIT open source. When customers require urgent delivery of the code, a process known as the software bundle archive (SBA) can be utilized.

The following compilation flags must be defined (set to y) in order to generate FIPS compliant code:

1. The FIPS_ENABLE flag, in kernel/drivers/crypto/Kconfig

If the above mentioned flags are not defined, the generated code is not FIPS compliant.

10.3.Guidance

10.3.1.Crypto Officer Guidance

To enable the QTI Cryptographic Module into the system, at the kernel build process, the Crypto Officer must ensure that the following compilation flag is defined (set to true):

- FIPS_ENABLE is defined in the kernel/drivers/crypto/Kconfig file. This ensures the build process will include all FIPS related binaries.

Then in the build process, the two kernel modules (qcrypto_module.ko and qdrbg_module.ko), integrity test app (qfinteverify) and their HMAC files (qcrypto_module.hmac, qdrbg_module.hmac, and qfinteverify.hmac) will be generated and installed in the boot.img.

In the bootup phase, the Crypto Officer can use “insmod” command in an Android init script file to load the 2 kernel modules sequentially in Android kernel init phase:

1) insmod /sbin/qcrypto_module.ko
2) insmod /sbin/qdrbg_module.ko

And then use “start” command to start the integrity test program as a service:

3) start qfintverify

The order of these “insmod” commands must be preserved, and they must be executed before any mount operations on disk partitions.

After these kernel modules are loaded, the poweron KAT self-tests and integrity tests will start to execute as described in section 9.

The Crypto Officer shall ensure that the User guidance (Section 11) is available to cryptographic module users.

11. User Guidance

11.1.Application Programming Interfaces

There are two interfaces into the CM, one from the kernel space and one from the user space.

11.2.User Space APIs

Status
The status of the cryptographic module is available via the following IOCTL: (Include File: linux/qcedev.h)

```
QCEDEV_IOCTL_QUERY_FIPS_STATUS_IOR(QCEDEV_IOC_MAGIC, 11, fips_status)
```

The returned value in fips_status is mapped by enum fips_status will present the current FIPS status as one of the following:

```
FIPS140_STATUS_NA = 0      (Non FIPS build)
FIPS140_STATUS_PASS = 3    (All tests passed and crypto engine is available)
FIPS140_STATUS_FAIL = 0xFF (One of the tests failed, crypto engine is unavailable
```

Random Numbers
To obtain a random number from the DRBG read data from the hardware random number device file /dev/hw_random.

The following example is how the UNIX dd command could be used to read 32000 bytes of random data from the DRBG:

dd if=/dev/hw_random of=file bs=32 count=1000

Cryptographic Functions
The following cryptographic functions are available via an IOCTL interface:

- AES-128 (ECB, CBC, CTR, and XTS mode)
- AES-256 (ECB, CBC CTR, and XTS mode)
- AES CMAC
- Triple-DES (CBC, ECB)
- SHA1/SHA256
- SHA1/SHA256 HMAC

All requests for service are synchronous. The invoking process will be put to sleep, waiting for completion of the request.

Source and destination buffers must point to the same location. Use of different locations for the source and destination buffers is not supported. For performance reasons, buffers must use memory allocated using PMEM thereby allowing the memory to be shared between user space and the kernel. Additionally, when in AES CTR mode and issuing either an encryption or decryption request, a non-zero byte offset is not supported.

Prior to invoking the ioctl, the application is required to open a file descriptor to the qcedev device as in the following example:

```
fd = open("/dev/qce", O_RDWR);
```

The user may then call the following ioctls with the file descriptor as the first parameter and a pointer to one of the following data structures (as appropriate) for the second parameter:

```
qcedev_cipher_op_req – for cipher operations
qcedev_sha_op_req – for hash/HMAC functionality
```

The following Cipher IOCTLs are available to the user space applications:

- QCEDEV_IOCTL_ENC_REQ – for encrypting data
- QCEDEV_IOCTL_DEC_REQ – for decrypting data

The second parameter is a pointer to the qcedev_cipher_op_req structure defined in qcedev.h:

The following hashing/HMAC IOCTLs are available to user space applications:

- QCEDEV_IOCTL_SHA_INIT_REQ – initialize a hash/HMAC request
- QCEDEV_IOCTL_SHA_UPDATE_REQ – for updating hash/HMAC
- QCEDEV_IOCTL_SHA_FINAL_REQ – for ending the hash/HMAC request
- QCEDEV_IOCTL_GET_SHA_REQ – retrieve the hash/HMAC for a data packet of a known size
- QCEDEV_IOCTL_GET_CMAC_REQ – for retrieving MAC (using the AES CMAC algorithm) for data packet of a known size

The second parameter to the IOCTL call is a pointer to the qcedev_sha_op_req structure defined in qcedev.h:

The IOCTLs and the associated data request structures are defined in:

```
kernel/include/uapi/linux/qcedev.h
```

It is the callers' responsibility to zeroize the 'req' data structure upon completion of the cipher operation.

11.3.Kernel Space APIs

In addition to the information presented below for the kernel space APIs, the Linux kernel now provides official documentation of the kernel crypto API. This documentation is available at:

https://www.kernel.org/doc/htmldocs/crypto-API/index.html

The following concepts are presented to simplify describing the kernel space APIs.

Page Vectors:
A page is the fundamental unit of memory used by the kernel. Consider data that resides on a specific page, offset from the start of the page by a specific amount and of a particular length. The location of this data can be expressed as a page-based tuple or page vector: {page, offset, length}.

Additionally, an existing kernel data structure called a scatterlist is used to operate on arrays of discontiguous pages.

Transforms:
Transforms are objects that instantiate algorithms manage internal state and handle common implementation logic. A set of API transform wrappers are provided in the kernel to simplify transform use and allow the properties of the transform's underlying algorithm to be queried.

11.3.1.Block Cipher Algorithms

The cryptographic block ciphers provided by the CM to kernel space are only available asynchronously and include:

- AES 128 (CBC, ECB, CTR, CCM and XTS mode)
- AES 256 (CBC, ECB, CTR, CCM and XTS mode)
- Triple-DES (CBC, ECB)

The following pseudo-code demonstrates a typical calling sequence to the kernel to utilize the block cipher APIs from kernel space asynchronously:

```
tfm = crypto_alloc_ablkcipher("algorithm[4]", type, mask);
req = ablkcipher_request_alloc(tfm, gfp-flag);
ablkcipher_request_set_callback(req, transform-flag,
                                _callback_function_ptr, &result);
crypto_ablkcipher_setkey(tfm, key, keylength);
crypto_ablkcipher_request_set_crypt(req, src, dest, numbytes, iv);
crypto_ablkcipher_encrypt(req);
crypto_free_ablkcipher(tfm);
ablkcipher_request_free(req);
```

11.3.2.Hash/HMAC Algorithms

The cryptographic block ciphers provided by the CM to kernel space are only available asynchronously and include:
SHA-1/SHA-256
HMAC-SHA-1/HMAC-SHA-256

The following pseudo-code demonstrates a typical calling sequence to the kernel to utilize the hash/HMAC APIs from kernel space asynchronously:

```
tfm = crypto_alloc_ahash("algorithm", type, mask);
req = ahash_request_alloc(tfm, gfp-flag);
ahash_request_set_callback(req, transform-flag,
                        _callback_function_ptr, &result);
crypto_ahash_setkey(tfm, key, keylength);
crypto_ahash_request_set_crypt(req, src, dest, numbytes);
crypto_ahash_digest(req);
crypto_free_ahash(tfm);
ahash_request_free(req);
```

The functions and the associated data structures are defined in:
`linux/opensource/kernel/include/linux/crypto.h.`

11.3.3.Status

The exported variable "g_fips140_status" (that can be read by any kernel module) is mapped by enum fips_status will present the current FIPS status as one of the following:
FIPS140_STATUS_NA = 0 (Non FIPS build)

[4] Also produces a Message Authentication Code
iver_name for the algorithm specified in table 1 in section 2.1.

FIPS140_STATUS_PASS = 3 (All tests passed and the crypto engine is available)
FIPS140_STATUS_FAIL = 0xFF (One of the tests failed, and the crypto engine is unavailable)

12. Mitigation of Other Attacks

The Mitigation of Other Attacks security section of FIPS 140-2 is not applicable to the QTI Cryptographic Module.

13.Terms and Abbreviations

AES	Advanced Encryption Specification
CAVP	Cryptographic Algorithm Validation Program
CBC	Cipher Block Chaining
CE	Cryptographic Engine
CCM	Counter with Cipher Block Chaining-Message Authentication Code
CM	Cryptographic Module
CMT	Cryptographic Module Testing
CMVP	Cryptographic Module Validation Program
COTS	Commercial Off The Shelf
CO	Crypto Officer
CSP	Critical Security Parameter
DES	Data Encryption Standard
FIPS	Federal Information Processing Standards Publication
FSM	Finite State Model
HMAC	Hash Message Authentication Code
IOCTL	Input Output Control
KAT	Known Answer Test
MAC	Message Authentication Code
NIST	National Institute of Science and Technology
NVLAP	National Voluntary Laboratory Accreditation Program
O/S	Operating System
QTI	Qualcomm Technologies, Inc.
RNG	Random Number Generator
SBA	Software Bundle Archive
SCM	Source Code Management
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
SLA	Service Level Agreement
SOF	Strength of Function
TDES	Triple DES
TFM	Transform
UI	User Interface