Security Policy: µMACE

Cryptographic module for the Motorola Solutions CRYPTR Micro which is used in the ES400 phone for LTE systems

Version: R01.00.26

Date: April 10, 2012

Table of Contents

1. Introduction

1.1. Scope

This Security Policy specifies the security rules under which the µMACE must operate. In addition to the security requirements derived from FIPS 140-2 are those imposed by Motorola. These rules, in total, define the interrelationship between the:

- Module Operators,
- Module Services, and
- Critical Security Parameters (CSPs).

1.2. Definitions

ALGID	Algorithm Identifier
CBC	Cipher Block Chaining
CFB	Cipher Feedback
CKR	Common Key Reference
CSP	Critical Security Parameter
DES	Data Encryption Standard
DRBG	Deterministic Random Bit Generator
ECB	Electronic Code Book
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
ECMQV	Elliptic Curve Menezes-Qu-Vanstone
IKE	Internet Key Exchange
IPSec	Internet Protocol security
ISAKMP	Internet Security Association and Key Management Protocol
IV	Initialization Vector
KLK	Key Loss Key
KPK	Key Protection Key
KVL	Key Variable Loader
LED	Light-emitting diode
LFSR	Linear Feedback Shift Register
PEK	Password Encryption Key
RAM	Random Access Memory
RNG	Random Number Generator

1.3. Overview

The µMACE provides secure key management and data encryption for the Motorola Solutions CRYPTR Micro which is used in the following broadband and LTE Systems:

- ES400 phone

1.4. µMACE Implementation

The µMACE is implemented as a single chip cryptographic module as defined by FIPS 140-2.

1.5. µMACE Hardware / Firmware Version Numbers

The µMACE has the following FIPS validated hardware and firmware version numbers:

Table 1: FIPS Validated Version Numbers

FIPS Validated Cryptographic Module Hardware Kit Numbers	FIPS Validated Cryptographic Module Firmware Version Numbers
AT58Z04	R01.00.04

1.6. µMACE Cryptographic Boundary

The µMACE in the block diagram below provides data security services required by the CRYPTR Micro product. The module is a single µMACE processor with the set of interfaces shown in the diagram below.

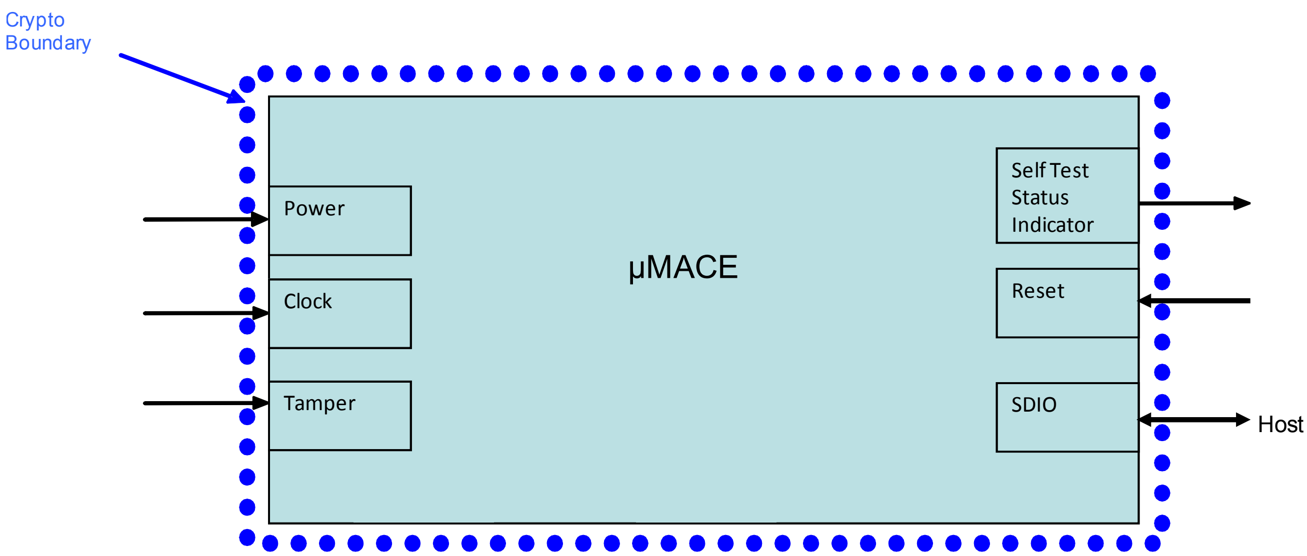

Figure 1: µMACE Block Diagram

The Crypto Boundary is drawn around the µMACE as shown below.

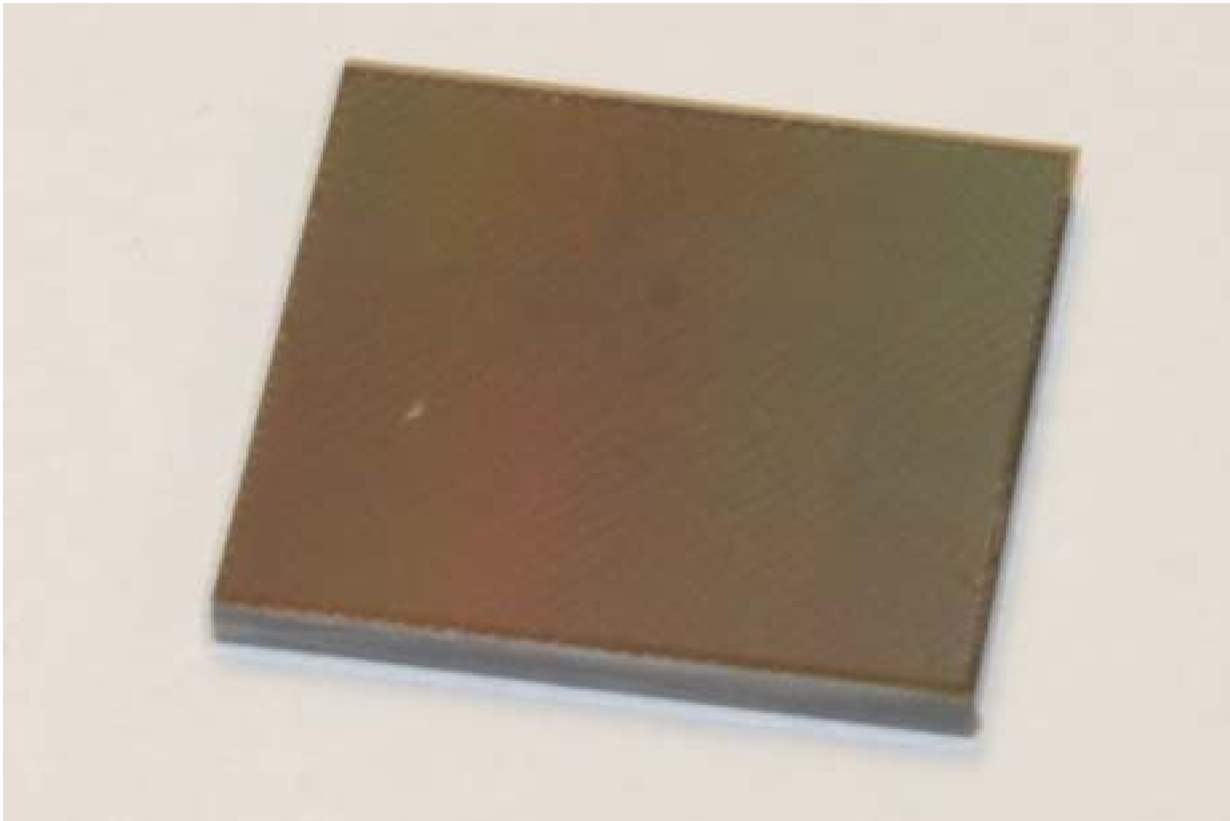

Figure 2: µMACE

1.7. Ports and Interfaces

The µMACE provides the following physical ports and logical interfaces:

Table 3: Ports and Interfaces

Physical Port	Qty	Logical interface definition	Description
Power	1	Power Input	This interface powers all circuitry. This interface does not support input / output of CSP's.
Clock	1	Control Input	Clock Input. This interface does not support input / output of CSP's.
Tamper	1	Control Input	Tamper Input. This interface does not support input / output of CSP's.
Self-test Status Indicator	1	Status Output	This interface provides status output to indicate all power-up self-tests completed successfully.
Reset	1	Control Input	This interface forces a reset of the module.
SDIO Interface	1	Control Input Status Output Data Output Data Input	Provides an interface for factory programming and execution of SDIO commands. All CSPs exchanged over this interface are always encrypted when operating in FIPS 140-2 Level 3 mode.

2. FIPS 140-2 Security Levels

The µMACE can be configured to operate at FIPS 140-2 overall Security Level 3. The table below shows the FIPS 140-2 Level of security met for each of the eleven areas specified within the FIPS 140-2 security requirements.

Table 4: µMACE Security Levels

FIPS 140-2 Security Requirements Section	Validated Level at overall Security Level 3
Cryptographic Module Specification	3
Module Ports and Interfaces	3
Roles, Services, and Authentication	3
Finite State Model	3
Physical Security	3
Operational Environment	N/A
Cryptographic Key Management	3
EMI / EMC	3
Self-Tests	3
Design Assurance	3
Mitigation of Other Attacks	N/A

3. FIPS 140-2 Approved Operational Modes

The µMACE can be configured to operate in a FIPS 140-2 Approved mode of operation and a non-FIPS Approved mode of operation. CSPs are not shared between FIPS Approved mode and non-FIPS Approved mode. The transition from a FIPS Approved mode to a non-FIPS Approved mode causes all CSPs to be zeroized. In response to the Version Query service request the module will return the following data which can be used to determine whether the module is operating at overall Security Level 3 or in a non-FIPS Approved mode.

FIPS Status Information Item ID	0x06
FIPS Status Information Item Length	0x01
FIPS Status Information Item Data	1 byte indicating the FIPS operating status of the module. The possible values are: - 0x00 – Not operating in a FIPS approved mode - 0x03 – Operating in a FIPS 140 Level 3 approved mode

The Version Query service can also be used to verify the firmware version matches an approved version listed on NIST's website: http://csrc.nist.gov/groups/STM/cmvp/validation.html

3.1. Configuration Settings for operation at FIPS 140-2 overall Security Level 3

Documented below are the actions and configuration settings required for the module to be used in a FIPS 140-2 Approved mode of operation at overall Security Level 3.

1. Disable Clear Key Import. The Module Configuration service is used to configure this parameter in the module. When this configuration setting is disabled, clear key import will be disallowed.
2. Disable Clear Key Export. The Module Configuration service is used to configure this parameter in the module. When this configuration setting is disabled, clear key export will be disallowed.
3. Disable Key Loss Key (KLK). The Module Configuration service is used to configure this parameter in the module.
4. Only Approved and Allowed algorithms installed. The module supports the following Approved algorithms:
    - AES-256 8-bit CFB8 (Cert. #1876) – used for symmetric encryption / decryption of keys and parameters stored in the internal database
    - AES-256 ECB (Cert. #1876) - for use with key wrap
    - AES-256 CBC (Cert. #1876) - for firmware upgrades
    - AES256 CTR (Cert. #1876) - for use with the SP800-90 DRBG
    - SHA-384 (Cert. #1619) – used for digital signature verification during firmware integrity test and firmware load test. Used for password hashing for internal password storage.
    - SP800-56A KAS (Cert. #28) – (key agreement; key establishment methodology provides 192 bits of encryption strength)
    - SP800-90 DRBG (Cert. #154) - used for IV and key generation.

- ECDSA-384 (Cert. #263) – used for digital signature verification during firmware integrity test and firmware load test, and for key generation and signature generation.

The module supports the following allowed algorithms:

- AES (Cert. #1876, key wrapping; key establishment methodology provides 256 bits of encryption strength) – used for key encryption

The following non-Approved algorithms and protocols are allowed within the Approved mode of operation:

- Non-deterministic Hardware Random Number Generator – used to provide random numbers used as Initialization Vectors (IV) and the seeds for the Approved DRBG

3.2. Non Approved Mode of Operation

A non-FIPS Approved mode of operation is transitioned to when any of the following is true:

1. Clear Key Import is enabled.
2. Clear Key Export is enabled.
3. KLK generation is enabled.

The module maintains FIPS mode status and will provide this upon operator request.

4. Crypto Officer and User Guidance

4.1. Administration of the µMACE in a secure manner (CO)

The µMACE requires no special administration for secure use after it is set up for use in a FIPS Approved manner. To do this, configure the module as described in Section 3 of this document.

Note that all keys will be zeroized after the Program Update service has completed.

4.2. Assumptions regarding User Behavior (CO)

The µMACE has been designed in such a way that no special assumptions regarding User Behavior have been made that are relevant to the secure operation of the unit.

4.3. Approved Security Functions, Ports, and Interfaces available to Users

µMACE services available to the User role are listed in section 8.2.

No Physical Ports or Logical Interfaces are directly available to the µMACE User, only indirectly through the host product in which the µMACE is installed.

4.4. User Responsibilities necessary for Secure Operation

To ensure the secure operation of the µMACE the operator should periodically check the module for evidence of tamper. It is recommended that the µMACE be checked for evidence of tamper every 6 months.

5. Security Rules

The µMACE enforces the following security rules.

5.1. FIPS 140-2 Imposed Security Rules

1. The µMACE inhibits all data output via the data output interface whenever an error state exists and during self-tests.
2. The µMACE logically disconnects the output data path from the circuitry and processes when performing key generation or key zeroization.
3. Authentication data (e.g. passwords) are entered in encrypted form. Authentication data is not output during entry.
4. Secret cryptographic keys are entered in encrypted form.
5. The µMACE does not support manual key entry.
6. The µMACE enforces Identity-Based authentication.
7. The µMACE supports a User role and a Crypto-Officer role. The module will verify the authorization of the operator to assume each role.
8. The µMACE re-authenticates an operator when it is powered-up after being powered-off.
9. The µMACE implements all firmware using a high-level language, except the limited use of low-level languages to enhance performance.
10. The µMACE protects secret keys and private keys from unauthorized disclosure, modification, and substitution.
11. The µMACE provides a means to ensure that a key entered into or stored within the module is associated with the correct entities to which the key is assigned. Each key in the µMACE is entered encrypted and stored with the following information:
    - Key Identifier – 16 bit identifier
    - Algorithm Identifier – 8 bit identifier
    - Key Type – Traffic Encryption Key or Key Encryption Key
    - Physical ID – Identifier indicating storage locations.

    Along with the encrypted key data, this information is stored in a key record that includes a CRC over all fields to protect against data corruption.
12. The µMACE denies access to plaintext secret and private keys contained within the module.
13. The Program Update service can be used to zeroize all plaintext cryptographic keys and other unprotected critical security parameters within the module.
14. The µMACE conforms to FCC 47 Code of Federal Regulations, Part 15, Subpart B, Unintentional Radiators, Digital Devices, Class B requirements.
15. The µMACE performs the following self-tests. Powering the module off then on will initiate the power up self-tests.
    - Power up and on-demand tests
      - Cryptographic algorithm test: A cryptographic algorithm test using a known answer is conducted for all cryptographic functions (e.g., encryption, decryption, authentication, random number generation, and hashing.) for each Approved algorithm listed below. The test passes if the final data matches the known data, otherwise it fails.
        - AES-256 (8-bit CFB, ECB, CBC, and CTR modes) encrypt / decrypt
        - SHA-384

- SP800-56A KAS (ECDH and ECMQV)
- SP800-90 DRBG
- ECDSA-384 (key generation)

- Firmware integrity test: A digital signature is generated over the code when it is built using SHA-384 and ECDSA-384 and is stored with the code upon download into the module. When the module is powered up the digital signature is verified. If the digital signature matches, then the test passes, otherwise it fails.
- Critical functions test: the module performs a read/write test of the internal RAM at each power up.

- Conditional tests
  - Firmware load test: A digital signature is generated over the code when it is built using SHA-384 and ECDSA-384. Upon download into the module, the digital signature is verified. If the digital signature matches, then the test passes, otherwise it fails.
  - Continuous Random Number Generator test: The continuous random number generator test is performed on all RNGs supported by the module (SP800-90 DRBG and NDRNG). For each RNG, an initial value is generated and stored upon power up. This value is not used for anything other than to initialize comparison data. A successive call to any one of the RNGs generates a new set of data, which is compared to the comparison data. If a match is detected, this test fails; otherwise the new data is stored as the comparison data and returned to the caller.
  - Pair-wise consistency test (for public and private keys used to perform the calculation and verification of digital signatures): The ECDSA Public and Private Generated Signature Key pair is tested by the calculation and verification of a digital signature. If the digital signature cannot be verified, the test fails.

16. The µMACE toggles the Self-test Indicator interface within 2 seconds of power-up to indicate the Firmware Integrity Test, Firmware Load Test, Cryptographic Algorithm Test, and Critical Functions Test have completed successfully. The µMACE enters the Critical Error state and does not toggle the Self-test Indicator interface if the Firmware Integrity Test, Firmware Load Test, Cryptographic Algorithm Test, or Critical Functions Test fails. The Critical Error state may be exited by powering the module off then on.
17. The µMACE enters the Critical Error state and outputs a message over the SDIO interface to indicate the Continuous Random Number Generator Test and Pair-wise Consistency tests have failed. The Critical Error state may be exited by powering the module off then on.
18. The µMACE does not perform any cryptographic functions while in an error state.

6. Identification and Authentication Policy

The µMACE supports a User role and a Crypto-Officer role.

The Crypto-Officer and User roles are authenticated with passwords. The Crypto-Officer and User passwords are initialized to a default value during manufacturing and are sent in encrypted form to the module for authentication. After authenticating, the Crypto-Officer and User passwords may be changed at any time.

Table 5: Roles and Authentication

Role	Authentication Type	Authentication Mechanism	Strength of Authentication
Crypto-Officer	Identity-Based	Identity: a 4-byte identifier is used to identify the identity and role. The µMACE supports a single identity. Crypto-Officer Password: a 14-32 character ASCII password is authenticated to gain access to all Crypto-Officer services.	Since the minimum password length is 14 ASCII printable characters and there are 95 ASCII printable characters, the probability of a successful random attempt is 1 in 95 ^ 14 or 1 in 4,876,749,791,155,298,590,087,890,625. The module limits the number of authentication attempts in one minute to 15. The probability of a successful random attempt during a one-minute period is 15 in 95 ^ 14 or 1 in 3.25117e+26.
User	Identity-Based	Identity: a 4-byte identifier is used to identify the identity and role. The µMACE supports a single identity. User Password: a 14-32 character ASCII password is authenticated to gain access to all User services.	Since the minimum password length is 14 ASCII printable characters and there are 95 ASCII printable characters, the probability of a successful random attempt is 1 in 95 ^ 14 or 1 in 4,876,749,791,155,298,590,087,890,625. The module limits the number of authentication attempts in one minute to 15. The probability of a successful random attempt during a one-minute period is 15 in 95 ^ 14 or 1 in 3.25117e+26.

7. Physical Security Policy

The µMACE is a production grade, single-chip cryptographic module as defined by FIPS 140-2 and is designed to meet Level 3 Physical Security.

The µMACE is covered with a hard opaque metallic coating that provides evidence of attempts to tamper with the module. Tampering with the module will cause it to enter a lock-up state in which no crypto services will be available.

No maintenance access interface is available.

8. Access Control Policy

8.1. µMACE Supported Roles

The µMACE supports two (2) roles. These roles are defined to be the:

- User Role
- Crypto-Officer Role

8.2. µMACE Services Available to the User Role.

- Validate User Password: Validate the current User password used to identify and authenticate the User role via the SDIO interface. Successful authentication will allow access to crypto services allowed for the User. Available in both FIPS and non-FIPS mode.
- Change User Password: Modify the current password used to identify and authenticate the User Role via the SDIO interface. Available in both FIPS and non-FIPS mode.
- Algorithm List Query: Provides a list of algorithms over the SDIO interface. Available in both FIPS and non-FIPS mode.
- Logout User Role: Logs out the User. Available in both FIPS and non-FIPS mode.
- Export Key Variable: Transfer encrypted key variables (KEKs, TEKs) out of the module over the SDIO interface. Available in both FIPS and non-FIPS mode.
- Import Key Variable: Receive encrypted key variables (KEKs, TEKs, and ECMQV Private Static Key) over the SDIO interface. Available in both FIPS and non-FIPS mode.
- Generate Key Variable: Auto-generate KEKs, TEKs, ECDH Public and Private Values, Public and Private Generated Signature Keys, ECMQV Public Static Key, ECMQV Public and Private Generated Ephemeral Keys, ECDH Shared Secret, and the KPK within the module. Available in both FIPS and non-FIPS mode.
- Delete Key Variable: Delete KEKs, TEKs, ECDH Public and Private Values, ECDH Public and Private Generated Signature Keys, ECMQV Public and Private Static Keys, ECMQV Public and Private Generated Ephemeral Keys, and ECDH Shared Secret. Available in both FIPS and non-FIPS mode.
- Edit Key Variable: Edit KEKs and TEKs managed by the module. Available in both FIPS and non-FIPS mode.
- Key Check: Validate the correctness of a Key based on algorithm properties. Available in both FIPS and non-FIPS mode.
- Encrypt: Encrypt plaintext data to be transferred over the SDIO interface. Available in both FIPS and non-FIPS mode.
- Decrypt: Decrypt ciphertext data received over the SDIO interface. Available in both FIPS and non-FIPS mode.
- Transfer Key Variable: Internally transfer key variables (KEKs, TEKs) between volatile and non-volatile memory. Available in both FIPS and non-FIPS mode.
- Generate Signature: Generate a Signature and output result over SDIO interface. Available in both FIPS and non-FIPS mode.
- Generate Hash: Generate a hash and output result over SDIO interface. Available in both FIPS and non-FIPS mode.
- Perform Key Agreement Process: Perform a key agreement process to create a key in volatile memory. Available in both FIPS and non-FIPS mode.
- Generate Random Number: Generate random data using the SP800-90 DRBG and

output result over SDIO interface. Available in both FIPS and non-FIPS mode.

- Key Query: Retrieve the metadata for a given key present in the module. Available in both FIPS and non-FIPS mode.

8.3. µMACE Services Available to the Crypto-Officer Role.

- Program Update: Update the module firmware via the SDIO interface. All keys (stored in RAM and non-volatile memory) and CSPs are zeroized during a Program Update. Available in both FIPS and non-FIPS mode.
- Validate Crypto-Officer password: Validate the current Crypto-Officer password used to identify and authenticate the Crypto-Officer role via the SDIO interface. Successful authentication will allow access to services allowed for the Crypto Officer. Available in both FIPS and non-FIPS mode.
- Change Crypto-Officer password: Modify the current password used to identify and authenticate the Crypto-Officer Role via SDIO interface. Available in both FIPS and non-FIPS mode.
- Extract Action Log: Exports a history of actions over the SDIO interface. Available in both FIPS and non-FIPS mode.
- Logout Crypto-Officer Role: Logs out the Crypto-Officer. Available in both FIPS and non-FIPS mode.
- Configure Module: Perform configuration of the module (e.g. time configuration, enable/disable clear key import, enable/disable red keyfill, etc.). Available in both FIPS and non-FIPS mode.

8.4. µMACE Services Available Without a Role.

- Perform Self-Tests: Performs module self-tests comprised of cryptographic algorithms test and firmware test. Initiated by a transition from power off state to power on state. Available in both FIPS and non-FIPS mode.
- Version Query: Provides module firmware version number and FIPS status over the SDIO interface. Available in both FIPS and non-FIPS mode.

8.5. Critical Security Parameters (CSPs) and Public Keys

Table 6: CSP Definition

CSP Identifier	Description
SP800-90 seed	This is a 384-bit seed value used within the SP800-90 DRBG. The seed is not stored but temporarily exists in volatile memory and is zeroized by power cycling the module. The seed is not entered into or output from the module. Entry - n/a Output - n/a Storage - in plaintext in volatile memory Zeroization - on power off or Program Update service request Generation - Non-deterministic Hardware Random Number Generator
SP800-90 internal state	This is the internal state of the SP800-90 DRBG during

CSP Identifier	Description
(“V” and “Key”)	initialization. The internal state is not stored but temporarily exists in volatile memory and is zeroized by power cycling the module. The internal state is not entered into or output from the module. Entry - n/a Output - n/a Storage – in plaintext in volatile memory Zeroization - on power off or Program Update service request Generation - Non-deterministic Hardware Random Number Generator
Key Protection Key (KPK)	This is a 256-bit AES key used to encrypt all other keys stored in non volatile memory. Generated internally using the SP800-90 DRBG. Stored in plaintext in non volatile memory. The KPK is not entered into or output from the module. Entry - n/a Output - n/a Storage – in plaintext in non-volatile memory Zeroization - on Program Update service request Generation - SP800-90 DRBG
Black Keyloading Key (BKK)	This is a 256-bit AES Key used for encrypting keys that are input into the module and output from the module via the SDIO interface. Stored unencrypted in RAM while in use; stored in plaintext in non-volatile memory and zeroized through the Program Update service. Also stored encrypted on the KPK in non volatile memory. The BKK is entered using the Program Update service (encrypted using AES-CBC) and is not output from the module. Entry - on Program Update service request Output - n/a Storage - in plaintext in non volatile memory and encrypted on KPK in non volatile memory Zeroization - on Program Update service request Generation - n/a
Image Decryption Key (IDK)	A 256-bit AES key used to decrypt downloaded images. The IDK is not output from the module. Entry - on Program Update service request Output - n/a Storage - in plaintext in non volatile memory Zeroization - on Program Update service request Generation - n/a
Traffic Encryption Keys (TEKs)	256-bit AES Keys used for enabling secure communication with target devices. TEKs are entered in encrypted form via the SDIO interface (AES Key Wrapping), and internally derived through key agreement (generated internally - n/a on entry encryption). TEKs entered through the SDIO interface are encrypted with the BKK. The TEKS are stored encrypted on

CSP Identifier	Description
	the KPK (AES256-CFB8) in non volatile memory. TEKs are stored in plaintext in RAM only as long as needed. TEKs are output from the module encrypted using KEKs (AES Key Wrapping). Entry – input encrypted with AES Key Wrap over the SDIO Interface Output – output encrypted with AES Key Wrap over the SDIO Interface Storage – stored encrypted on KPK with AES256-CFB8 in non volatile memory Zeroization - on Delete Key Variable and Program Update service requests Generation – internally derived through key agreement
Key Encryption Keys (KEKs)	256-bit AES Keys used for enabling secure communication with target devices. KEKs are entered in encrypted form via the SDIO interface (AES Key Wrapping). KEKs entered through the SDIO interface are encrypted with the BKK. The KEKS are stored encrypted on the KPK (AES256-CFB8) in non volatile memory. KEKs are stored in plaintext in RAM only as long as needed. KEKS are not output from the module. Entry – input encrypted with AES Key Wrap over the SDIO Interface Output - n/a Storage – stored encrypted on KPK with AES256-CFB8 in non volatile memory Zeroization - on Delete Key Variable and Program Update service requests Generation - n/a
User Password	The User Password is entered encrypted on the PEK (AES256-CFB8). The User Password is not stored in the module or output from the module. Entry – entered encrypted on the PEK with AES256-CFB8 Output - n/a Storage – a hash of the User Password is stored in non-volatile memory Zeroization – on Program Update service request Generation - n/a
Crypto-Officer Password	The Crypto Officer password is entered encrypted on the PEK (AES256-CFB8). After decryption the plaintext password is not stored but temporarily exists in volatile memory. The SHA-384 hash value of the plaintext password is stored encrypted on the PEK in non volatile memory. The SHA-384 hash of the decrypted password is compared with the SHA-384 hash value stored in non-volatile memory during password validation. Entry - entered encrypted on the PEK with AES256-CFB8 Output - n/a Storage - SHA-384 hash of the plaintext password is

CSP Identifier	Description
	encrypted on the PEK in non volatile memory Zeroization – on Program Update service requests Generation - n/a
Password Encryption Key (PEK)	This is a 256-bit AES Key used for decrypting passwords during password validation. Loaded via the Program Update service. Stored in plaintext in non-volatile memory and zeroized through the Program Update service. Also stored encrypted on the KPK in non volatile memory. The PEK is not output from the module. Entry - on Program Update service request Output - n/a Storage - in plaintext in non volatile memory; encrypted on the KPK in non volatile memory Zeroization - on Program Update service request Generation - n/a

CSP Identifier	Description
Elliptic Curve Diffie-Hellman Private value	Randomly generated internally by a Generate Key Variable service request using SP800-90 DRBG. Used to establish a shared secret over an insecure channel. Stored in plaintext in volatile memory. The Elliptic Curve Diffie-Hellman Private value is not entered into or output from the module. Entry - n/a Output - n/a Storage – in plaintext in volatile memory Zeroization - on Delete Key Variable or Program Update service requests and on power off Generation - on Generate Key Variable service request
ECDSA Private Generated Signature Key (PGSK)	Randomly generated internally by the Generate Key Variable service request using SP800-90 DRBG. Stored in non volatile memory encrypted on KPK; when in use it is in plaintext in RAM. The Private Generated Signature Key is not output from the module. Entry - n/a Output - n/a Storage – encrypted on KPK in non volatile memory Zeroization - on Delete Key Variable and Program Update service requests Generation - on Generate Key Variable service request
ECMQV Private Static Key	The ECMQV Private Static Key is entered over the SDIO Interface encrypted on the BKK (AES Key Wrapping). Used to establish a shared secret over an insecure channel. Stored encrypted with KPK in non volatile memory. The ECMQV Private Static Key is not output from the module. Entry – entered encrypted on the BKK with AES Key Wrap over the SDIO Interface Output - n/a Storage - encrypted on KPK in non volatile memory Zeroization - on Delete Key Variable and Program Update service requests Generation - n/a
ECMQV Private Generated Ephemeral Key (PGEK)	Randomly generated internally by the Generate Key Variable service request using SP800-90 DRBG. Used to establish a shared secret over an insecure channel. When in use it is in plaintext in RAM. The ECMQV Private Generated Ephemeral Key is not entered into or output from the module. Entry - n/a Output - n/a Storage - n/a Zeroization - on power off or Program Update service request Generation - SP800-90 DRBG
Elliptic Curve Diffie-Hellman Shared Secret	Generated internally by the Elliptic Curve Diffie-Hellman Algorithm. The Elliptic Curve Diffie-Hellman Shared Secret is output encrypted on a KEK (AES Key Wrapping) as part of the Diffie-Hellman key agreement protocol. Entry - n/a Output – output encrypted on a KEK with AES Key

CSP Identifier	Description
	Wrap over the SDIO interface Storage – in plaintext in volatile memory Zeroization - on power off or Program Update service request Generation - internally by the Elliptic Curve Diffie-Hellman Algorithm

Table 7: Public Keys

Key	Description
ECDSA Public Programmed Signature Key	A 384-bit ECDSA public key used to validate the signature of the firmware image being loaded before it is allowed to be executed. Stored in non volatile memory. Loaded during manufacturing and as part of the boot image during a Program Update service. The Public Programmed Signature Key is not output from the module. Entry - on Program Update service request Output - n/a Storage - in plaintext in non volatile memory Zeroization - on Program Update service request Generation - n/a

Elliptic Curve Diffie-Hellman Public value	Generated internally by the Generate Key Variable service request. Used to establish a shared secret over an insecure channel. Stored in plaintext in volatile memory. The Elliptic Curve Diffie-Hellman Public value is generated internally and is output as part of the Diffie-Hellman key agreement protocol. Entry - n/a Output - in plaintext over the SDIO interface Storage - in plaintext in volatile memory Zeroization - on Delete Key Variable and Program Update service requests and on power off Generation - on Generate Key Variable service request
ECDSA Public Generated Signature Key	Generated internally by the Generate Key Variable service request. A 384-bit ECDSA key used to validate signatures. Stored in plaintext in non volatile memory. The ECDSA Public Generated Signature Key is output from the module in plaintext over the SDIO interface. Entry - n/a Output - in plaintext over the SDIO interface Storage - in plaintext in non volatile memory Zeroization - on Delete Key Variable and Program Update service requests Generation - on Generate Key Variable service request
ECMQV Public Static Key	The ECMQV Public Static Key is generated internally by the Generate Key Variable service request. Used to establish a shared secret over an insecure channel. Stored in plaintext in non volatile memory. The ECMQV Public Static Key is output in plaintext from the module over the SDIO interface. Entry - n/a Output - SDIO Interface in plaintext Storage - in plaintext in non volatile memory Zeroization - on Delete Key Variable and Program Update service requests Generation – on Generate Key Variable service request
ECMQV Public Generated Ephemeral Key	Generated internally by the Generate Key Variable service request. Used to establish a shared secret over an insecure channel. Stored in plaintext in non volatile memory. The ECMQV Public Generated Ephemeral Key is output from the module in plaintext over the SDIO interface. Entry - n/a Output - SDIO Interface in plaintext Storage - n/a Zeroization - on power off or Program Update service request Generation - on Generate Key Variable service request
Remote Party Diffie-Hellman Ephemeral Public Key	Input in plaintext over the SDIO interface. Used to establish a shared secret over an insecure channel. Stored in plaintext in volatile memory. Entry – in plaintext over SDIO interface Output – n/a Storage - in plaintext in volatile memory Zeroization - on Delete Key Variable and Program Update service requests and on power off

	Generation – n/a
Remote Party ECMQV Ephemeral Public Key	Input in plaintext over the SDIO interface. Used to establish a shared secret over an insecure channel. Stored in plaintext in volatile memory. Entry – in plaintext over SDIO interface Output – n/a Storage - in plaintext in volatile memory Zeroization - on Delete Key Variable and Program Update service requests and on power off Generation – n/a
Remote Party ECMQV Static Public Key	Input in plaintext over the SDIO interface. Used to establish a shared secret over an insecure channel. Stored in plaintext in volatile memory. Entry – in plaintext over SDIO interface Output – n/a Storage - in plaintext in volatile memory Zeroization - on Delete Key Variable and Program Update service requests and on power off Generation – n/a

8.6. CSP Access Types

Table 8: CSP Access Types

CSP Access Type	Description
C – Check CSP	Checks status of the CSP.
D – Decrypt CSP	Decrypts entered KEKs and TEKs using the BKK during CSP entry over the SDIO interface. Decrypts entered passwords using the PEK during entry over the SDIO interface.
E – Encrypt CSP	Encrypts KEKs and TEKs prior to output over the SDIO interface using a KEK or BKK.
G – Generate CSP	Generates KPK, SP800-90 seed, SP800-90 seed key, Elliptic Curve Diffie-Hellman private key or ECMQV private key.
S – Store CSP	Stores KPK in plaintext in non volatile memory. Stores plaintext BKK, PEK, or IDK in volatile and non-volatile memory (encrypted except IDK). Stores SHA-384 Hash of the Crypto-Officer password in non volatile memory (encrypted on PEK).
U – Use CSP	Uses CSP internally for encryption / decryption services.
Z – Zeroize CSP	Zeroizes CSP.

Table 9: CSP versus CSP Access

Service	CSP															Role		
	SP800-90 seed	SP800-90 internal state	PEK	TEKs	KEKs	KPK	BKK	IDK	User Password	Crypto-Officer Password	ECDH Private Value	ECDSA PGSK	ECMQV Private Static Key	ECMQV PGEK	ECDH Shared Secret	User Role	Crypto-Officer Role	No Role Required
1. Program Update			z, s	z	z	z	z, s	u, z, s		z	z	z	z				√	
2. Validate Crypto-Officer Password			u							d, u, z							√	
3. Change Crypto-Officer Password			u							d,u, z, s							√	
4. Validate User Password			u			d			d, u, z							√		
5. Change User Password			u			d, s, e,g			d, u, z							√		
6. Extract Action Log																	√	
7. Version Query																		√
8. Algorithm List Query																√		
9. Logout User Role														z		√		
10. Logout Crypto-Officer Role																	√	
11. Export Key Variable				d, e, u	d, e, u	u	u								d, e, u	√		
12. Import Key Variable				d,e, s, u	d,e, s, u	u	u				u	u	d,e, s, u	u		√		
13. Generate Key Variable	u	u		e, g, s	e, g, s	u					e, g, s	e, g, s		e, g, s		√		
14. Delete Key Variable				z	z						z	z	z	z		√		
15. Edit Key Variable				d,e, u, s	d,e, u, s	u					d, s	d, s	d, s	d		√		
16. Key Check				c	c	u					c	c	c	c		√		
17. Encrypt				u	u	u	u									√		
18. Decrypt				u	u	u	u				u	u	u	u		√		
19. Perform Self-Tests																		√
20. Transfer Key				d,e, u, s	d,e, u, s	u					d,e, u, s	d,e, u, s	d,e, u, s			√		

Variable																		
21. Generate Signature						u					d, u	d, u	d, u	d, u		√		
22. Generate HASH						u					d, u	d, u	d, u	d, u		√		
23. Perform Key Agreement						u					d, u	d, u	d, u	d, u	u	√		
24. Configure Module																	√	
25. Random number generation	u															√		
26. Key Query				d	d	u					d	d	d	d		√		

9. Mitigation of Other Attacks Policy

The µMACE is not designed to mitigate any specific attacks outside of those required by FIPS 140-2.