# mToken CryptoID

## Hardware Version: SCC-X

## Firmware Version: 3.11

**FIPS 140-2 Non-Proprietary Security Policy**

Version 1.4

Last Update: 2016-04-07

Prepared by:
atsec information security corporation
9130 Jollyville Road, Suite 260
Austin, TX 78759
www.atsec.com

## Copyrights and Trademarks



## Table of Contents



 

# 1. Introduction

This document is the non-proprietary FIPS 140-2 Security Policy for the mTokenCryptoID cryptographic module. It contains specific rules under which the module must operate and describes how this module meets the requirements as specified in FIPS PUB 140-2 (Federal Information Processing Standards Publication 140-2) for a Security Level 3 module.

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

In this document, the terms “token”, “cryptographic module” or “the module” are used interchangeably to refer to the mTokenCryptoID.

## 1.1 Purpose of the Security Policy

There are three major reasons that a Security Policy is needed:

- To provide a specification of the cryptographic security that will allow individuals and organizations to determine whether a cryptographic module, as implemented, satisfies a stated Security Policy.
- To describe to individuals and organizations the capabilities, protection, and access rights provided by the cryptographic module, thereby allowing an assessment of whether the module will adequately serve the individual or organizational security requirements.

## 1.2 Target Audience

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

- Those specifying cryptographic modules
- Administrators of the cryptographic module(s)
- Users of the cryptographic module(s)

# 2. Cryptographic Module Specification

The following section describes the cryptographic module and how it conforms to the FIPS 140-2 specification in each of the required areas.

## 2.1. Description of Module

The module mToken CryptoID is designed based on a secure smartcard chip that utilizes the built-in mCOS to communicate with a General Purpose Computer (GPC) via the USB interface in a “plug-and-play” manner. The mToken CryptoID implements the USB Circuit Cards Interface Device (CCID) protocol to communicate with the host application running on a computer device. The Application Protocol Data Unit (APDU) command-respond pairs transferred via the USB interface for communication is compatible with ISO/IEC 7816-4 standards. It provides the security services needed to interact with the Public Key Infrastructure (PKI) applications, including Digital Signature Generation/Verification for online authentication and Data Encryption/Decryption for online transactions. Here is the list of main functions provided by the module:

- Transportation Management
- File System Management
- Secret PIN Management
- Access Control Management
- Session Key Management
- Key Pair Management
- Data Digest Management
- Algorithm Management
- Algorithm Hardware Engine

The module is validated as a multi-chip standalone hardware module against FIPS 140-2 at an overall Security Level 3. The table below shows the security level claimed for each of the eleven sections that comprise the FIPS 140-2:

| FIPS 140-2 Sections | Security Level | | | | |
|---|---|---|---|---|---|
| | N/A | 1 | 2 | 3 | 4 |
| Cryptographic Module Specification | | | | X | |
| Cryptographic Module Ports and Interfaces | | | | X | |
| Roles, Services and Authentication | | | | X | |
| Finite State Model | | | | X | |
| Physical Security | | | | X | |
| Operational Environment | X | | | | |
| Cryptographic Key Management | | | | X | |
| EMI/EMC | | | | X | |
| Self Tests | | | | X | |
| Design Assurance | | | | X | |
| Mitigation of Other Attacks | X | | | | |

*Table 1: Security Levels*

The cryptographic boundary of the module is defined as the entire mToken CryptoID.

The physical boundary of the cryptographic module is defined as the opaque enclosure surrounding the token device as shown in the following picture.

*Figure 1: mToken CryptoID*

The mToken CryptoID provides two designs: one is plastic and the other is metal. Both designs of the module and all the colors listed in the Figure 1 have been tested and passed the Level 3 Physical Security Testing in accordance with FIPS 140-2.

For the plastic design, the weight of the module is 7g and the dimensions are approximately 55mm * 16.6mm * 7mm.

For the metal design, the weight of the module is 8g and the dimensions are approximately 56mm * 17mm * 8mm.

The module components within the logical boundary of the mToken CryptoID are specificed in the table below:

| Component Type | Part Number/File Name/Version |
|---|---|
| Hardware | AisinoChip Electronics SCC-X |
| Firmware | mTokenCryptoID_V3.11_B20151111.bin |
| Documentation | mToken CryptoID FIPS 140-2 Non-Proprietary Security Policy (Version 1.4) |
| | mToken CryptoID FSM (Version 0.4) |
| | mToken CryptoID Product Design (Version 2.6) |
| | mToken CryptoID APDU Specification (Version 1.7) |
| | mToken CryptoID HRNG Design (Version 1.1) |
| | mToken CryptoID Configuration Management Plan (Version 1.2) |

*Table 2: mToken CryptoID Cryptographic Module Components*

## 2.2. Modes of Operation

The mToken CryptoID supports FIPS mode (i.e., Approved mode of operation) and non-FIPS mode (i.e., non-Approved mode of operation).

The mode of operation is configured by the Issuer during initialization prior to being distributed to the end users. By default, the module enters in non-FIPS mode after the firmware is loaded on the token and powered on for the first time. The module is initialized by the Issuer by calling the **FormatDevice** APDU command with “bCipherWorkMode” parameter. If the “bCipherWorkMode” is 0, the module is initialized and configured in non-FIPS mode; if the “bCipherWorkMode” is 1, the module is initialized and configured in FIPS mode. Once the module is initialized, the Issuer can switch the mode of operation by calling the **ChangeWorkingMode** APDU command with “P1” parameter. If “P1” is 1, the module is configured to operate in FIPS mode; if “P1” is 0, the module is configured to operate in non-FIPS mode. After the successful completion of the power-up self-tests, the module enters FIPS mode or non-FIPS mode automatically based on the value of the “P1” parameter.

When the module operates in FIPS mode, the steady green LED is on and the red LED is off. The module returns “01” from the calling **GetData** APDU command which indicates that the module is in FIPS mode, and the module returns “0000000000000000” to the calling **GetHealthStatus** APDU command which indicates that all self-tests have completed successfully. Only Approved services that use the Approved or Allowed algorithms are allowed in FIPS mode which are listed in Table 9: Approved Services. Please refer to Table 6: LED status for the details of the LED status indicator of the module.

The module implements the Approved algorithms listed in the following table:

| Algorithms | Keys/CSPs | Usage | Standards | CAVS certificates |
|---|---|---|---|---|
| AES<br>• ECB<br>• CBC | 128-, 192- and 256-bit keys | Encryption and Decryption | FIPS 197 | #3582 |
| AES KW | 128-, 192- and 256-bit keys | Key Wrapping | SP 800-38F | #3582 |
| Triple-DES<br>• ECB<br>• CBC | 3-key Triple-DES key | Encryption and Decryption | SP 800-67 | #1994 |
| Triple-DES TKW | 3-key Triple-DES Key | Key Wrapping | SP800-38F | #1994 |
| SHA-256, SHA-384, SHA-512 | N/A | Hashing | FIPS 180-4 | #2944 |
| Hash-based DRBG with SHA-256 | Entropy Input String, seed, V and C values | Random Number Generation | SP 800-90A | #921 |

| | | | | |
|---|---|---|---|---|
| ECDSA<br>• SHA-256<br>• SHA-384<br>• SHA-512 | ECDSA Public and Private key pair according to P-256 and P-521 curves | Key Pair Generation, Public Key Verification, Signature Generation and Signature Verification | FIPS 186-4 | #728 |
| RSA<br>• SHA-256<br>• SHA-384<br>• SHA-512 | RSA Public and Private key pair with 2048 bits modulus size | Appendix B.3.3 Key Generation, PKCS#1 v1.5 Signature Generation and Signature Verification | FIPS 186-4 | #1843 |
| HMAC<br>• SHA-256<br>• SHA-384<br>• SHA-512 | At least 112 bits keys | Message Integrity | FIPS 198-1 | #2282 |
| CMAC with AES | 128-, 192- and 256-bit keys | Message Integrity | SP 800-38B | #3582 |
| CMAC with Triple-DES | 3-key Triple-DES key | Message Integrity | SP 800-38B | #1994 |
| EC Diffie-Hellman<br>• SHA-256<br>• SHA-384<br>• SHA-512 | ECDSA Public and Private key pair according to P-256 and P-521 curves | Key Derivation Function and Key Agreement Scheme Full Validation (as specified in Section 5.6.2.4 and/or 5.6.2.5) | CAVS tested for SP 800-56A Revision 1 | KAS #70 |
| | | SP 800-56A ECC CDH Primitive (Section 5.7.1.2) Component Test | Vendor Affirmed for SP 800-56A Revision 2 | CVL #610 |

*Table 3: Approved Algorithms*

When the module operates in non-FIPS mode, the green LED is off and the steady red LED is on. The module returns “00” from the calling **GetData** APDU command which indicates that the module is in FIPS mode, and the module returns “0000000000000000” to the calling **GetHealthStatus** APDU command which indicates that the self-test have completed successfully. The **GetData** APDU command returns an error when the module is uninitialized. Both Approved services and non-Approved services listed in Table 9: Approved Services and Table 10: Non-Approved Services are allowed in non-FIPS mode.

The module implements the non-Approved algorithms listed in the following table:

| Algorithms | Usage |
|---|---|
| SHA-1 | Hashing, not CAVS tested |
| 1024 bits RSA | Non-compliant key size for Digital Signature Generation and Verification, Key Wrapping |
| 2048 bits RSA | RSA-based Key Wrapping using SP 800-56B Encryption and Decryption Primitives (Section 7.1) |

| | |
|---|---|
| HMAC-SHA-1 | Message Authentication Code, not CAVS tested |
| 2-key Triple-DES<br>• ECB<br>• CBC | Encryption and Decryption, Key Wrapping<br>(The Decryption and Key Unwrap are allowed for legacy-use purpose after 2015, but have not been CAVS tested.) |
| CMAC with 2-key Triple-DES | Message Authentication Code |
| NDRNG | Allowed to be used in FIPS mode to seed the SP 800-90A DRBG |

*Table 4: non-Approved Algorithms*

When the mToken CryptoID is switching between FIPS mode and non-FIPS mode, the file structure of the module is erased including the keys stored in the Key File (KF) and Elementary File (EF). The module must be unplugged and plugged back into the GPC. The module will then re-initialize the file structure and user accounts. Thus, the keys/CSPs (i.e., DRBG seed, Public-Private key pairs, PINs and session keys) used for cryptographic operations by the end users are not shared between the modes of operation.

## 2.3. Cryptographic Module Block Diagrams

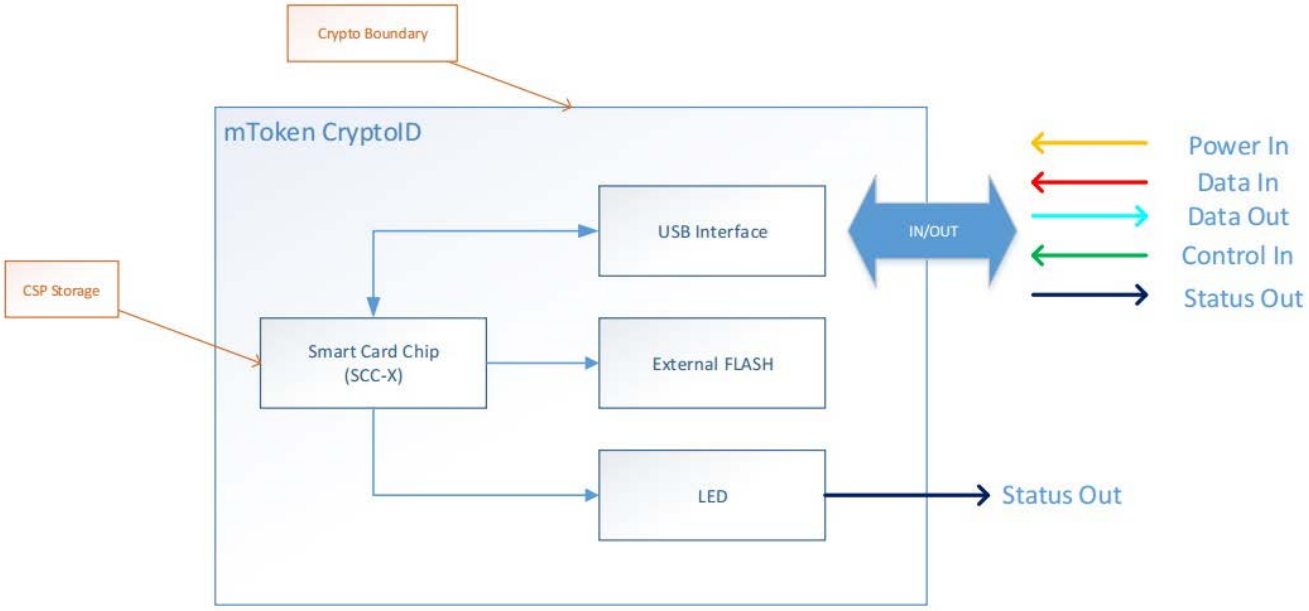


*Figure 2: Logical Block Diagram of the mToken CryptoID*

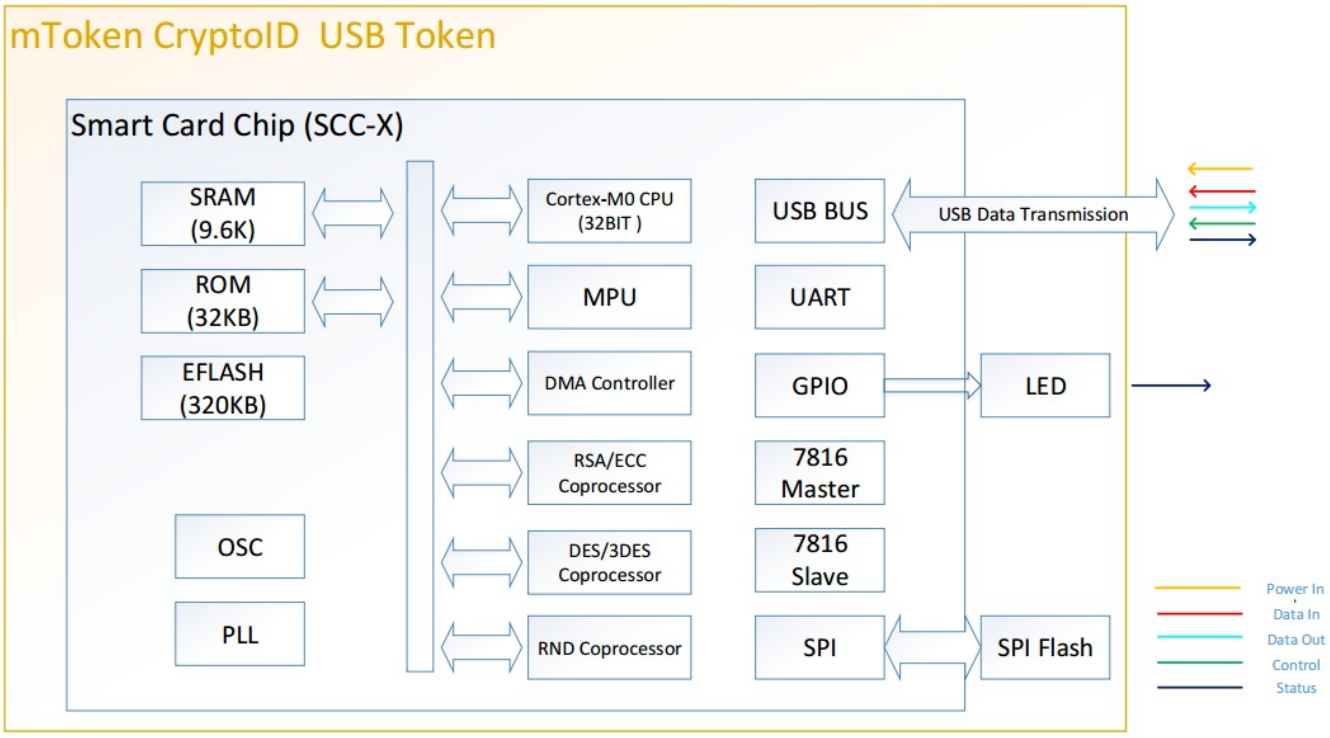


*Figure 3: Hardware Block Diagram of the mToken CryptoID and the Smart Card Chip*

# 3.Cryptographic Module Ports and Interfaces

The following table provides the mapping between the FIPS interfaces, physical ports, and the logical interfaces:

| FIPS Interfaces | Physical Ports | Logical Interfaces |
|---|---|---|
| Data Input | Data pins of the USB port | The input data field of the command APDU to USB Token |
| Data Output | Data pins of the USB port | The output data field of the response APDU from USB Token |
| Control Input | Data pins of the USB port | The command APDU header consisting of the CLA, INS, P1, P2 and Lc Fields |
| Status Output | Data pins of the USB port<br>LED | SW1, SW2<br>LED status indicates the current working mode and Error state |
| Power Input | Power pins of the USB port | N/A |

*Table 5: Ports and Interfaces*

The following table depicts the LED configurations for the corresponding FIPS states:

| FIPS state | Green LED | Red LED |
|---|---|---|
| Power Off | OFF | OFF |
| Self-Test | ON | ON |
| FIPS mode | ON | OFF |
| FIPS mode in Error state | FAST BLINK (200ms on, 200ms off) | OFF |
| Switching from FIPS mode to non-FIPS mode | SLOW BLINK (1000ms on, 1000ms off) | OFF |
| Non-FIPS mode | OFF | ON |
| Non-FIPS mode in Error state | OFF | FAST BLINK (200ms on, 200ms off) |
| Switching from non-FIPS mode to FIPS mode | OFF | SLOW BLINK (1000ms on, 1000ms off) |

*Table 6: LED status*

# 4. Roles, Services, and Authentication

The file system of the mTokenCryptoID supports multiple levels of directory structure. There are four types of components: Master File (MF), Dedicated File (DF), Elementary File (EF) and Key File (KF). The MF is the root directory of the entire file system and there is a KF associated with it. They are created and initialized during manufacturing. The DF is the directory file that contains child directory files or EFs. If the DF is associated with a KF, this type of DF is called an Application DF (ADF). The EF is the file that contains the data. There are four types of EFs supported by the module: Binary File, Linear Variable Record File, Public Key File, and Private Key File. The KFs are the files that store the authentication data (i.e., KEY_MC, KEY_EA, KEY_IA and KEY_PIN).

Identity-Based Authentication is required to authenticate the operator accessing the module and to verify that the operator is authorized to assume the requested role and perform the services within that role. Operators can authenticate themselves by providing the Main Control Key (KEY_MC) or a correct PIN with the associated user account identifier, PIN ID. The module supports up to eight user accounts and each user account is assumed with one type of operator.

Each user account has its own Role ID for access control to the module. The Role ID can be categorized into three types of operators. The operator is authorized to access certain DFs based on the permission set for each DF and the operator's Role ID. The role of the operator is implicitly selected by the operator based on the requested services.

The module does not support concurrent operators.

The details of the roles, services, and authentication mechanisms are given in the following sections.

## 4.1. Roles

The mTokenCryptoID supports two types of roles: Crypto Officer and User. The following table describes the authorized roles for operators:

| Roles | Description |
|---|---|
| Crypto Officer (CO) | Performs a set of cryptographic operations including initialization, configuration, and management functions (e.g., module initialization, input/output of cryptographic keys and CSPs, and file structure creation). |
| User | Performs general security services including cryptographic operations and other Approved security functions. |

*Table 7: Type of Roles*

The mTokenCryptoID supports three types of operators: Issuer, Admin, and User. The following table describes the types of operators:

| Operators | Description |
|---|---|
| Issuer | The module only supports one Issuer. The user account for the Issuer is initialized during manufacturing at the MF level of the file structure. The Issuer can create new DF levels of the file structure under the MF, set the module in FIPS or non-FIPS mode, and create other user accounts for Admin or User operators under the DF. |

| | |
|---|---|
| Admin | The module supports up to three Admin operators. The user accounts for the Admin operators are created by the Issuer at the DF level of the file structure. The Admin can unlock the Admin or User PIN and create, read, and update the Admin files. |
| User | The module supports up to four User operators. The user accounts for the User operators are created by the Issuer at the DF level of the file structure. The User can perform cryptographic operations, read and update User files, and change the User PIN. |

*Table 8: Type of Operators*

Upon successfully operator's authentication, the role of the operator is implicitly selected by the operator based on the requested services. Please refer to Table 9: Approved Services and Table 10: Non-Approved Services for the details of the services allowed for each role.

## 4.2. Services

In FIPS mode, the module only supports Approved services. In non-FIPS mode, the module supports both Approved and non-Approved services. The Approved services are listed in Table 9: Approved Services and the non-Approved Services are listed in Table 10: Non-Approved Services.

The following table provides the Approved services available in FIPS mode and non-FIPS mode (Note: "CO" stands for Crypto Officer, "R" stands for Read, "W" stands for Write, and "E" stands for Execute):

| Services | Operators | | | Keys/ CSPs | Description | Access (R/W/E) | Roles | |
|---|---|---|---|---|---|---|---|---|
| | Issuer | Admin | User | | | | CO | User |
| ***Issuer Command*** | | | | | | | | |
| FormatDevice | ✓ | | | KEY_MC, KEY_EA, KEY_IA | Initialize and format the file system. | W | ✓ | |
| ChangeWorkingMode | ✓ | | | KEY_MC | Change working mode: FIPS or Non FIPS. | W | ✓ | |
| InstallPIN | ✓ | | | KEY_MC, KEY_PIN | Install a new KEY_PIN into KF of the current DF. | W | ✓ | |
| ***General Command*** | | | | | | | | |
| DRBGReseed | ✓ | ✓ | ✓ | KEY_PIN, KEY_MC, Entropy Input, seed, V and C values | SP 800-90A DRBG reseeding which the entropy input is collected from the NDRNG. | W | | ✓ |

| Services | Operators | | | Keys/ CSPs | Description | Access (R/W/E) | Roles | |
|---|---|---|---|---|---|---|---|---|
| | Issuer | Admin | User | | | | CO | User |
| ***Session Key Management Command*** | | | | | | | | |
| SessionKeyKAS for an Approved algorithm | | ✓ | ✓ | KEY_PIN, ECDSA Keys, Key Derived Material, Session Key | EC Diffie-Hellman (P-256/P-521) KAS and KDF: Generate a Session Key for an Approved algorithm (i.e., 3-key Triple-DES, AES, CMAC with 3-key Triple-DES, CMAC with AES, HMAC-SHA-256, HMAC-SHA-384 and HMAC-SHA-512.) | RWE | | ✓ |
| GenSessionKey for an Approved algorithm | | ✓ | ✓ | KEY_PIN, Session Key, Entropy Input, seed, V and C values | SP 800-90A DRBG random engine: Generate a Session Key for an Approved algorithm (i.e., 3-key Triple-DES, AES, CMAC with 3-key Triple-DES, CMAC with AES, HMAC-SHA-256, HMAC-SHA-384 and HMAC-SHA-512.) | RW | | ✓ |
| DestroySessionKey | | ✓ | ✓ | KEY_PIN, Session Key | Destroy Session Key. | W | | ✓ |
| GetSessionInfo | | ✓ | ✓ | KEY_PIN, Session Key | Get the Session Key's algorithm. | R | | ✓ |
| ***File Command*** | | | | | | | | |
| Create File | ✓ | ✓ | ✓ | KEY_MC, KEY_PIN | Create a DF or EF. | W | | ✓ |
| Delete File | ✓ | ✓ | ✓ | KEY_MC, KEY_PIN | Delete a DF or EF. | W | | ✓ |
| ReadBinary | | ✓ | ✓ | KEY_PIN | Read binary file data. | R | | ✓ |

| Services | Operators | | | Keys/ CSPs | Description | Access (R/W/E) | Roles | |
|---|---|---|---|---|---|---|---|---|
| | Issuer | Admin | User | | | | CO | User |
| UpdateBinary | | ✓ | ✓ | KEY_PIN | Update data into binary file. | W | | ✓ |
| AppendRecord | | ✓ | ✓ | KEY_PIN | Append new record to record file. | RW | | ✓ |
| ReadRecord | | ✓ | ✓ | KEY_PIN | Read record data of record file. | R | | ✓ |
| UpdateRecord | | ✓ | ✓ | KEY_PIN | Update data into a record of the record file. | RW | | ✓ |
| Get Data | | ✓ | ✓ | KEY_PIN | Get device information or get the specified tag data. (Depends on the tag ID, the authentication is required for 0x4000-0XFFFF according to the “Read” access control of EF 3F01.) | R | | ✓ |
| Put Data | | ✓ | ✓ | KEY_PIN | Set device information or save the specified tag data. (Depends on the tag ID, the authentication is required for 0x4000-0XFFFF according to the “Write” access control of EF 3F01.) | W | | ✓ |
| ***Secret PIN management and authentication Command*** | | | | | | | | |
| ChangeSecretKey | ✓ | ✓ | ✓ | KEY_MC, KEY_PIN, KEY_EA, KEY_IA | Change the specified KEY_MC/ KEY_IA/ KEY_PIN/ KEY_EA. | W | | ✓ |
| UnblockSecretKey | ✓ | ✓ | | KEY_MC, KEY_PIN, KEY_EA | Unblock the specified Keys.<br>Admin can unblock KEY_PIN; Issuer can unblock KEY_EA under MF. | W | ✓ | |

| Services | Operators | | | Keys/ CSPs | Description | Access (R/W/E) | Roles | |
|---|---|---|---|---|---|---|---|---|
| | Issuer | Admin | User | | | | CO | User |
| ***Algorithm Command*** | | | | | | | | |
| SymImportSessionKey (3-key Triple-DES or AES) | | ✓ | ✓ | KEY_PIN, Session Keys | Import a Session Key that is wrapped by a 3-key Triple-DES or AES Session Key. | RW | | ✓ |
| SymExportSessionKey (3-key Triple-DES or AES) | | ✓ | ✓ | KEY_PIN, Session Keys | Export a Session Key that is wrapped by a 3-key Triple-DES or AES Session Key. | RW | | ✓ |
| SymCryptInit (3-key Triple-DES or AES) | | ✓ | ✓ | KEY_PIN, Session Key | Symmetric algorithm initialization. | R | | ✓ |
| SymEncryptUpdate (3-key Triple-DES or AES) | | ✓ | ✓ | KEY_PIN, Session Key | Continuously encrypt data part. | R | | ✓ |
| SymEncryptFinal (3-key Triple-DES or AES) | | ✓ | ✓ | KEY_PIN, Session Key | Encrypt the final data part and finish the operation. | R | | ✓ |
| SymDecryptUpdate (3-key Triple-DES or AES) | | ✓ | ✓ | KEY_PIN, Session Key | Continuously decrypt data part. | R | | ✓ |
| SymDecryptFinal (3-key Triple-DES or AES) | | ✓ | ✓ | KEY_PIN, Session Key | Decrypt the final data part and finish the operation. | R | | ✓ |
| MacInit (3-key Triple-DES CMAC, AES CMAC, HMAC-SHA-256, HMAC-SHA-348 or HMAC-SHA-512) | | ✓ | ✓ | KEY_PIN, Session Key | Initialize MAC Calculation. | R | | ✓ |
| MacUpdate (3-key Triple-DES CMAC, AES CMAC, HMAC-SHA-256, HMAC-SHA-348 or HMAC-SHA-512) | | ✓ | ✓ | KEY_PIN, Session Key | MAC calculation update. | R | | ✓ |

| Services | Operators | | | Keys/ CSPs | Description | Access (R/W/E) | Roles | |
|---|---|---|---|---|---|---|---|---|
| | Issuer | Admin | User | | | | CO | User |
| MacFinal (3-key Triple-DES CMAC, AES CMAC, HMAC-SHA-256, HMAC-SHA-348 or HMAC-SHA-512) | | ✓ | ✓ | KEY_PIN, Session Key | MAC calculation finish. | R | | ✓ |
| AsymGenKeypair (2048 bits RSA or ECDSA, and *bUsage=1 or 2) | | ✓ | ✓ | KEY_PIN, RSA or ECDSA Public and Private Keys | Generate asymmetric key pairs. | W | | ✓ |
| AsymWrapImportPub (3-key Triple-DES or AES, and *bUsage=1 or 2) | | ✓ | ✓ | KEY_PIN, Session Key | Import a public key that is wrapped by a 3-key Triple-DES or AES Session Key. | RW | | ✓ |
| AsymImportPub (*bUsage=1 or 2) | | ✓ | ✓ | KEY_PIN, RSA or ECDSA Public Key | Import a Public key in plaintext. | W | | ✓ |
| AsymWrapImportPri (3-key Triple-DES or AES, and *bUsage=1 or 2) | | ✓ | ✓ | KEY_PIN, Session Key, RSA or ECDSA Private Key | Import a private key that is wrapped by a 3-key Triple-DES or AES Session Key. | RW | | ✓ |
| AsymWrapExportPub (3-key Triple-DES or AES) | | ✓ | ✓ | KEY_PIN, Session Key, RSA or ECDSA Public Key | Export a public key wrapped with a 3-key Triple-DES or AES Session Key. | RW | | ✓ |
| AsymExportPub (2048 bits RSA or ECDSA) | | ✓ | ✓ | KEY_PIN, RSA or ECDSA Public Key | Export a Public key in plaintext. | R | | ✓ |

| Services | Operators | | | Keys/ CSPs | Description | Access (R/W/E) | Roles | |
|---|---|---|---|---|---|---|---|---|
| | Issuer | Admin | User | | | | CO | User |
| AsymWrapExportPri (3-key Triple-DES or AES) | | ✓ | ✓ | KEY_PIN, Session Key, RSA or ECDSA Private Key | Export a Private key that is wrapped by a 3-key Triple-DES or AES Session Key. | RW | | ✓ |
| AsymImportSessionKey (2048 bits RSA) | | ✓ | ✓ | KEY_PIN, RSA Private Key, 3-key Triple-DES Session Key | Import a 3-key Triple-DES Session Key that is wrapped by a Public Key with 2048 bits RSA. | R | | ✓ |
| AsymExportSessionKey (2048 bits RSA) | | ✓ | ✓ | KEY_PIN, RSA Public Key, 3-key Triple-DES Session Key | Export a 3-key Triple-DES Session Key that is wrapped by a Public Key with 2048 bits RSA. | R | | ✓ |
| AsymSign (2048 bits RSA or ECDSA) | | ✓ | ✓ | KEY_PIN, RSA or ECDSA Private Key | Asymmetric Private key signature operation with 2048 bits RSA or ECDSA. | R | | ✓ |
| AsymVerifySign (2048 bits RSA or ECDSA) | | ✓ | ✓ | KEY_PIN, RSA or ECDSA Public Key | Asymmetric Public key digital signature verification with 2048 bits RSA or ECDSA. | R | | ✓ |
| *Note: The following services do not require any authentication and the operator is not required to assume an authorized role according to FIPS 140-2 IG 3.1.* | | | | | | | | |
| ***General Command*** | | | | | | | | |
| GetFSMaxSpace | | | | N/A | Get max available space for file system. | R | | |
| GetChipID | | | | N/A | Get the smart card chip serial number. | R | | |
| GetHealthStatus | | | | N/A | Get self-test result: power up and on demand. | R | | |
| HealthCheck | | | | N/A | On demand self-test. | E | | |

| Services | Operators | | | Keys/ CSPs | Description | Access (R/W/E) | Roles | |
|---|---|---|---|---|---|---|---|---|
| | Issuer | Admin | User | | | | CO | User |
| ***Secure Message Command*** | | | | | | | | |
| SecureMessageKAS | | | | KEY_EA, ECDSA Keys, Key Derived Material, S_ENC, S_MAC | EC Diffie-Hellman (P-521) KAS and KDF to generate the Secure Message Keys (S_ENC and S_MAC) to establish a secure channel between the module and the host application.<br><br>Note: The **ExternalAuth** service is required before performing this service. | RWE | | |
| ***Secret PIN management and authentication Command*** | | | | | | | | |
| GetChallenge | | | | Entropy Input, seed, V and C | Get random challenge data from SP 800-90A DRBG. | W | | |
| ExternalAuth | | | | KEY_EA | External authentication with KEY_EA. | R | | |
| InternalAuth | | | | KEY_IA | Internal authentication with KEY_IA. | R | | |
| IssuerAuth | | | | KEY_MC | Verify Issuer's key. | R | | |
| Verify PIN | | | | KEY_PIN | Verify Admin or User PIN. | R | | |
| GetSecretKeyInfo | | | | N/A | Get the specified information (i.e., Algorithm ID, PIN type, Left retry times, access control and Unblock PIN ID) of KEY_MC/ KEY_IA/ KEY_PIN/ KEY_EA. | R | | |
| ClearSecureState | | | | N/A | Logoff current DF or MF. | W | | |

| Services | Operators | | | Keys/ CSPs | Description | Access (R/W/E) | Roles | |
|---|---|---|---|---|---|---|---|---|
| | Issuer | Admin | User | | | | CO | User |
| ***File Command*** | | | | | | | | |
| Select File | | | | N/A | Select a DF or EF and get the information. | R | | |
| Get Data | | | | N/A | Get device information or get the specified tag data. (Depends on the tag ID, the authentication is required for 0x4000-0XFFFF according to the “Read” access control of EF 3F01.) | R | | |
| Put Data | | | | N/A | Set device information or save the specified tag data. (Depends on the tag ID, the authentication is required for 0x4000-0XFFFF according to the “Write” access control of EF 3F01.) | W | | |
| ***Algorithm Command*** | | | | | | | | |
| HashInit (SHA-256, 384 or 512) | | | | N/A | Initialize a hash operation. | RW | | |
| HashUpdate (SHA-256, 384 or 512) | | | | N/A | Continuously hash data part. | RW | | |
| HashFinal (SHA-256, 384 or 512) | | | | N/A | Hash the final data part and get the hash value. | RW | | |

*Table 9: Approved Services*

The following table provides the Non-Approved services that only allow in non-FIPS mode:

| Service | Operators | | | Keys/ CSPs | Description | Access (R/W/E) | Roles | |
|---|---|---|---|---|---|---|---|---|
| | Issuer | Admin | User | | | | CO | User |
| ***Session Key Management Command*** | | | | | | | | |
| CreateSessionKey | | ✓ | ✓ | Session Key | Create a Session Key with plaintext key value. | RW | | ✓ |
| SessionKeyKAS for an non-Approved algorithm | | ✓ | ✓ | KEY_PIN, ECDSA Keys, Key Derived Material, Session Key | Generate CMAC with 2-key Triple-DES, HMAC-SHA-1 or 2-key Triple-DES Session Key with EC Diffie-Hellman (P-256/P-521) KAS and KDF. | RWE | | ✓ |
| GenSessionKey for an non-Approved algorithm | | ✓ | ✓ | KEY_PIN, Session Key, Entropy Input, seed, V and C | Generate CMAC with 2-key Triple-DES, HMAC-SHA-1 or 2-key Triple-DES Session Key with SP 800-90A DRBG random engine. | RW | | ✓ |
| ***Algorithm Command*** | | | | | | | | |
| SymImportSessionKey (2-key Triple-DES) | | ✓ | ✓ | KEY_PIN, Session Keys | Import a Session Key that is wrapped by a 2-key Triple-DES Session Key. | RW | | ✓ |
| SymExportSessionKey (2-key Triple-DES) | | ✓ | ✓ | KEY_PIN, Session Keys | Export a Session Key that is wrapped by a 2-key Triple-DES Session Key. | RW | | ✓ |
| SymCryptInit (2-key Triple-DES) | | ✓ | ✓ | KEY_PIN, Session Key | Symmetric algorithm initialization. | R | | ✓ |
| SymEncryptUpdate (2-key Triple-DES) | | ✓ | ✓ | KEY_PIN, Session Key | Continuously encrypt data part. | R | | ✓ |
| SymEncryptFinal (2-key Triple-DES) | | ✓ | ✓ | KEY_PIN, Session Key | Encrypt the final data part and finish the operation. | R | | ✓ |

| Service | Operators | | | Keys/ CSPs | Description | Access (R/W/E) | Roles | |
|---|---|---|---|---|---|---|---|---|
| | Issuer | Admin | User | | | | CO | User |
| SymDecryptUpdate (2-key Triple-DES) | | ✓ | ✓ | KEY_PIN, Session Key | Continuously decrypt data part. | R | | ✓ |
| SymDecryptFinal (2-key Triple-DES) | | ✓ | ✓ | KEY_PIN, Session Key | Decrypt the final data part and finish the operation. | R | | ✓ |
| MacInit (HMAC-SHA-1 or 2-key Triple-DES CMAC) | | ✓ | ✓ | KEY_PIN, Session Key | Initialize MAC Calculation. | R | | ✓ |
| MacUpdate (HMAC-SHA-1 or 2-key Triple-DES CMAC) | | ✓ | ✓ | KEY_PIN, Session Key | MAC calculation update. | R | | ✓ |
| MacFinal (HMAC-SHA-1 or 2-key Triple-DES CMAC) | | ✓ | ✓ | KEY_PIN, Session Key | Mac calculation finish. | R | | ✓ |
| AsymGenKeypair (1024 bits RSA or *bUsage=3) | | ✓ | ✓ | KEY_PIN, RSA Public and Private Keys | Generate asymmetric key pairs with 1024 bits RSA or bUsage=3*. | W | | ✓ |
| AsymWrapImportPub (2-key Triple-DES or *bUsage=3) | | ✓ | ✓ | KEY_PIN, Session Key, RSA or ECDSA Public Key | Import a public key that is wrapped by a 2-key Triple-DES Session Key or the public key usage is defined as bUsage=3*. | RW | | ✓ |
| AsymImportPub (*bUsage=3) | | ✓ | ✓ | KEY_PIN, RSA or ECDSA Public Key | Import a public key in plaintext where the public key usage is defined as bUsage=3*. | W | | ✓ |
| AsymWrapImportPri (2-key Triple-DES or *bUsage=3) | | ✓ | ✓ | KEY_PIN, Session Key, RSA or ECDSA Private Key | Import a private key that is wrapped by a 2-key Triple-DES Session Key or the private key usage is defined as bUsage=3*. | RW | | ✓ |

| Service | Operators | | | Keys/ CSPs | Description | Access (R/W/E) | Roles | |
|---|---|---|---|---|---|---|---|---|
| | Issuer | Admin | User | | | | CO | User |
| AsymWrapExportPub (2-key Triple-DES) | | ✓ | ✓ | KEY_PIN, Session Key, RSA or ECDSA Public Key | Export a public key wrapped with a 2-key Triple-DES Session Key. | RW | | ✓ |
| AsymWrapExportPri (2-key Triple-DES) | | ✓ | ✓ | KEY_PIN, Session Key, RSA or ECDSA Private Key | Export a private key that is wrapped by a 2-key Triple-DES Session Key. | RW | | ✓ |
| AsymImportSessionKey (1024 bits RSA) | | ✓ | ✓ | KEY_PIN, RSA Private Key, 2-key Triple-DES Session Key | Import a 2-key Triple-DES Session Key that is wrapped by other party Public Key with 1024 bits RSA. | R | | ✓ |
| AsymExportSessionKey (1024 bits RSA) | | ✓ | ✓ | KEY_PIN, RSA Public Key, 2-key Triple-DES Session Key | Export a 2-key Triple-DES Session Key that is wrapped by a Public Key with 1024 bits RSA. | R | | ✓ |
| AsymEnDecrypt | | ✓ | ✓ | KEY_PIN, RSA or ECDSA Public and Private Keys | Asymmetric encrypt/decrypt with 1024 or 2048 bits RSA. | R | | ✓ |
| AsymKeyOperation | | ✓ | ✓ | KEY_PIN, RSA Public and Private Keys | Asymmetric public key operation and private key operation with 1024 or 2048 bits RSA. | R | | ✓ |
| AsymSign (1024 bits RSA or 2048 bits RSA with external hash) | | ✓ | ✓ | KEY_PIN, RSA Private Key | Asymmetric private key sign operation with 1024 bits RSA or 2048 bits RSA where the hash is generated externally. | R | | ✓ |

| Service | Operators | | | Keys/ CSPs | Description | Access (R/W/E) | Roles | |
|---|---|---|---|---|---|---|---|---|
| | Issuer | Admin | User | | | | CO | User |
| AsymVerifySign (1024 bits RSA or 2048 bits RSA with external hash) | | ✓ | ✓ | KEY_PIN, RSA Public Key | Asymmetric public key digital signature verification with 1024 bits RSA or 2048 bits RSA where the hash is generated externally. | R | | ✓ |
| *Note: The following services do not require authentication.* | | | | | | | | |
| ***Algorithm Command*** | | | | | | | | |
| HashInit (SHA-1) | | | | N/A | Initialize a hash operation. | RW | | |
| HashUpdate (SHA-1) | | | | N/A | Continuously hash data part. | RW | | |
| HashFinal (SHA-1) | | | | N/A | Hash the final data part and get the hash value. | RW | | |

*Table 10: Non-Approved Services*

* bUsage is a input parameter to specify the usage of the private-public key pairs. Option 1 means signature generation and verification only, Option 2 means key wrapping only, and Option 3 means the key pair can be used for both signature generation and verification, and key wrapping which is not allowed in FIPS mode.

For details regarding service inputs, corresponding service outputs, status return codes and description of each service listed in Table 9: Approved Services and Table 10: Non-Approved Services, please refer to section 4 of “mToken CryptoID APDU Specification V1.7”.

## 4.3. Authentication

The mToken CryptoID supports Identity-Based Authentication to authenticate the operators.

The module provides three security layers to protect the authentication mechanism.

### 4.3.1. First Security Layer – Authentication at the Application Level

The module supports External Authentication and Internal Authentication for the authentication between the module and the application running on a host.

Both External Authentication and Internal Authentication are key-based authentication. External Authentication (**ExternalAuth**) with KEY_EA authenticates the host application running to the module. The module also supports Internal Authentication (**InternalAuth**) with KEY_IA which the module authenticates itself to the host application.

This layer of protection ensures that the module communicates with a legitimate host application which the keys KEY_EA and KEY_IA are only known between the Manufacture and the Distributor.

### 4.3.2. Second Security Layer – Security Channel Establishment

The module supports **SecureMessageKAS** which uses EC Diffie-Hellman with P-521 curve for KAS and KDF to derive the S_ENC and S_MAC keys to establish a secure channel. Then, the module uses the S_ENC key for AES-256 bits encryption and decryption and the S_MAC key for CMAC with AES-256 bits secured hashing to protect the data transfer between the module and the host application. The AES encryption and decryption provides confidentiality and the CMAC with AES secured hashing provides data integrity.

### 4.3.3. Third Security Layer – Operator's Authentication

As described in Table 8: Type of Operators, the module supports three types of operator: Issuer, Admin and User.

The module supports key-based authentication for the Issuer authentication and supports PIN-based authentication for the Admin and User authentication. The Issuer is authenticated by providing the Main Control Key (KEY_MC) with **IssuerAuth** APDU command. The Admin and User can authenticate themselves by providing their KEY_PIN (i.e., Admin PIN or User PIN) with **Verify PIN** APDU command. The operators can logoff logout by calling the **ClearSecureState** APDU command or by unplugging the module from the GPC.

## 4.4. Mechanism and Strength of Authentication

The module uses the "challenge-response" method for all types of authentication.

For key-based authentication, the host application calls the **GetChallenge** command to get a random value from the module that has the same length as the authenticated keys. The host application then encrypts the random value with the AES-256 bit symmetric algorithm and transmits the ciphertext to the module for verification. The module decrypts the ciphertext with the AES-256 bit symmetric algorithm and compares the result with the original random value from the **GetChallenge** command. If the results are the same, the authentication is completed successfully; otherwise, the authentication fails and the module will decrement the value pre-defined in "LeftRetryTimes". Once the value stored in "LeftRetryTimes" reaches 0, the key is locked.

For PIN-based authentication, the host application calls the **GetChallenge** command to get a random value from the module that has the same length as the authenticated keys. The host application then generates a SHA-256 hash value from the PIN provided by the operator. The random value is encrypted with the SHA-256 hash value of the PIN using a pre-defined symmetric algorithm. The PIN ID and ciphertext are transmitted to the module for verification. The module generates a SHA-256 hash value from the PIN based on the PIN ID, decrypts the ciphertext with the SHA-256 hash value using the pre-defined symmetric algorithm, and compares the result with the original random value from the **GetChallenge** command.

If the results are the same, the authentication is completed successfully; otherwise, the authentication fails and the module will decrement the “LeftRetryTimes” variable. Once the value stored in “LeftRetryTimes” reaches 0, the PIN is locked.

No visible display of the authentication data is allowed from the module. The host application running on the GPC interacting with the module obscures the authentication data during data entry. The authentication data is stored in the key files which can never be exported outside the mToken CryptoID. The current operator status is stored in the security context in memory. When the operator logs out or if another operator attempts to login, the security context will be cleared and the operator will be logged out automatically. The security context will also be cleared when the module is powered off.

The following table shows the strength of the operator’s authentication mechanism:

| Operator’s Authentication Mechanism | Strength |
|---|---|
| Key-Based Authentication:<br>KEY_MC with **IssuerAuth** APDU command. | The KEY_MC is a 256-bit AES key.<br>The probability that a random attempt will succeed using this authentication method is 1/2^256, which is less than 1/1,000,000. The probability that a random attempt will succeed over a one minute interval is less than 1/100,000. |
| PIN-Based Authentication:<br>SHA-256 hashed KEY_PIN (i.e., Admin PIN or User PIN) with the **Verify PIN** APDU command. | The KEY_PIN length must be between 6 and 32 characters with a combination of letters, numeric characters, and special characters (i.e., A-Z a-z 0-9 ~ ` ! @ # $ % ^ & * ( ) _ + \| - = \ { } [ ] : ; " ' < > , . ? / Tab and Space). This yields 96 choices per character. The probability of a successful random attempt is at most 1/96^6, which is less than 1/1,000,000.<br>Assuming 10 attempts per second via a scripted or automated attack, the probability of guessing the PIN with multiple attempts in a one minute period is 600/96^6, which is less than 1/100,000.<br>The module will lock an account after 15 consecutive failed authentication attempts; thus, the maximum number of attempts in a one minute period is 15. Therefore, the probability of success with multiple consecutive attempts in a one minute period is 15/96^6, which is less than 1/100,000. |

*Table* 11*: Operator’s Authentication Mechanism and Strength*

## 5. Physical Security

The module is a multi-chip standalone module and conforms to Level 3 requirements for physical security. The module hardness testing was performed at a single temperature (i.e., ambient temperature) and no assurance is provided for Level 3 hardness conformance at any other temperature. The module is composed 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 is housed in a sealed, hard plastic enclosure or metal enclosure that has no openings, vents or doors. It cannot be opened without damage.

## 6. Operational Environment

The module operates in a limited operational environment and does not implement a General Purpose Operating System. Once the firmware of the module is loaded on the mToken CryptoID, it cannot be modified or erased. The operational environment requirements do not apply to the module.

## 7. Cryptographic Key Management

All the cryptographic keys and CSPs are stored within the physical boundary of the module. The module has met the Physical Security Level 3 requirements which protect the sensitive information from disclosure, unauthorized modification, or substitution.

The following table provides a summary of the keys and CSPs that are employed by the module:

| Keys/CSPs | Name | Access | Type | Usage |
|---|---|---|---|---|
| KEY_MC | Main Control Key | Issuer | 256 bit AES Key | Issuer Authentication |
| KEY_EA | External Authentication Key | Anyone | 256 bit AES Key | Authenticate the host application running on a GPC |
| KEY_IA | Internal Authentication Key | Anyone | 256 bit AES Key | Authenticate the module itself to the host application running on a GPC |
| KEY_PIN | Admin PIN | Admin | 6-32 Characters which will be hashed by SHA-256 and converted into 128, 192, or a 256 bit AES Key, or a 168 bit Triple-DES Key | Admin or User Authentication |
| | User PIN | User | | |
| S_ENC | Data Encryption Key | Anyone | 256 bit AES Key | Secure Messaging |
| S_MAC | MAC key | Anyone | 256 bit AES CMAC Key | |

| | | | | |
|---|---|---|---|---|
| Session Keys | Crypto Session Key | Admin, User | 128, 192, or 256 bit AES Key, or 168 bit Triple-DES Key | Data Encryption, Decryption, Key Wrapping, Message Authentication Code |
| | Mac Session Key | Admin, User | 128, 192, or 256 bit AES Key, 168 bit Triple-DES Key, or at least a 112 bit HMAC Key | |
| RSA Private Key | RSA Private Key | Admin, User | 2048 bit modulus size of RSA Key Pairs | Signature Generation and Verification, Key Wrapping |
| RSA Public Key | RSA Public Key | Admin, User | | |
| ECDSA Private Key | ECDSA Private Key | Admin, User | P-256 or P-521 curve ECDSA Key Pairs | Signature Generation and Verification, EC Diffie-Hellman KAS and KDF |
| ECDSA Public Key | ECDSA Public Key | Admin, User | | |
| Entropy Input | DRBG Entropy Input | Issuer, Admin, User | At least 512 bits random values generated from NDRNG | Seeding the SP 800-90A DRBG |
| Seed | DRBG seed | Issuer, Admin, User | 440 bit value are generated internally by the Hash-based DRBG with SHA-256 | SP 800-90A-compliant random number generation |

*Table 12: Keys/CSPs*

Note: The Key Derived Material generated in the EC Diffie-Hellman KAS and KDF, and the V and C values generated in the SP 800-90A DRBG internal state are intermediate CSPs. Thus, they are not listed in the table above as the module does not support the input/output of intermediate values of key material or CSPs.

## 7.1. Random Number Generation

The mToken CryptoID uses a FIPS-Approved Deterministic Random Bit Generator (DRBG) based on SP 800-90A for random number generation, symmetric key generation, asymmetric key generation, and key establishment. The module implements a hash based DRBG (Hash_DRBG) which generates at least 256 bits random value per request.

At least 512 bits data is provided by the IC hardware-based nondeterministic random number generator (NDRNG) for seeding the Hash_DRBG implementation. The module also implements checks to ensure that two consecutive seeds are not identical before the later one is used as a valid input to seed the SP 800-90A DRBG.

## 7.2. Key/CSP Generation

The module supports the following Approved keys/key material generation methods in FIPS mode:

- FIPS 186-4 RSA and ECDSA Key Generation are used to generate the public and private key pair. The prime numbers for RSA and ECDSA Key Generation are generated from SP 800-90A DRBG.
- The SP 800-56A EC Diffie-Hellman Key Agreement Scheme and the Key Derivation Function are used to generate the Data Encryption Key (S_ENC), MAC key (S_MAC), and Session Keys (Crypto Session Key and Mac Session Key). The ECDSA key pair used in the EC Diffie-Hellman protocol is compliant with FIPS 186-4 ECDSA Key Generation.
- The Crypto Session Key and Mac Session Key are generated directly from the SP 800-90A DRBG.

## 7.3. Key/CSP Establishment

The mToken CryptoID supports the following Key Establishment methods:

- SP 800-56A rev2 EC Diffie-Hellman Key Agreement Scheme
- RSA-based Key Wrapping using SP 800-56B Encryption and Decryption Primitives (Section 7.1)
- SP 800-38F Symmetric Key Wrapping including KW mode for AES and TKW for Triple-DES

The keys generated from the EC Diffie-Hellman KAS and KDF are used in two services: Secure Messaging and Session Key Generation.

The module provides a Secure Messaging service which establishes the Secure Message Channel between the module itself and the host application running on a GPC for data transfer. The module uses EC Diffie-Hellman with the P-521 curve for key agreement and Single Step KDF with SHA-512 to derive a Data Encryption Key (S_ENC) and a MAC Key (S_MAC) that are used in Secure Messaging. All the data transfer between the module and the host application (including the APDU commands and the data field) are encrypted with the AES 256 bit S_ENC key in CBC mode. A message digest is also generated with the CMAC AES 256 bit S_MAC key for data integrity.

In addition to the Secure Messaging, the module supports a second layer of protection for the transfer data by providing the Session Key services. The module uses the P-256 or P-521 curve EC Diffie-Hellman for key agreement and KDF with HMAC to derive a Crypto Session Key and Mac Session Key. These Session Keys can be used for symmetric key wrapping to import/export RSA or ECDSA key pairs, symmetric key wrapping to import/export other Session Keys, data encryption and decryption.

Note: The module prohibits any service that uses RSA-based or symmetric key wrapping methods when the security strength of wrapping key is smaller than the wrapped key.

- The module supports RSA-based Key Wrapping to import and export Session Keys which RSA key wrapping provides 112 bits of encryption strength and non-compliant less than 112 bits of encryption strength;
- The module supports Triple-DES Key Wrapping to import and export Session Keys which Triple-DES key wrapping provides 112 bits of encryption strength and non-compliant less than 112 bits of encryption strength;
- The module supports AES Key Wrapping to import and export Session Keys which AES key wrapping provides between 128 and 256 bits of encryption strength.

## 7.4.Key/CSP Entry and Output

The keys are entered into the module or output from the module electronically via the Data field in the APDU command. The end users enter their PINs manually with a key loader (i.e., a keyboard from a host GPC). The APDU commands such as **InstallPIN**, **IssuerAuth**, **VerifyPIN**, **ChangeSecretKey,** and **UnblockSecretKey** require Secure Messaging such that all the sensitive authentication data is protected by the Secure Messaging service with a symmetric algorithm (i.e., AES with 256 bits). The APDU commands **SymImportSessionKey**, **SymExportSessionKey**, **AsymWrapImportPri**, **AsymWrapExportPri**, **AsymWrapImportPub**, **AsymWrapExportPub**, **AsymImportSessionKey,** and **AsymExportSessionKey** are used for importing and exporting the public keys, private keys, and Session keys by using an Approved or Allowed key wrapping method described in section 7.3.

The following table describes how the keys/CSPs are generated, imported, and exported from the module:

| Keys/CSPs | Key Generation | Key Entry | Key Output |
|---|---|---|---|
| KEY_MC | Generated externally during manufacturing. | Entered in plaintext during manufacturing. | Never exits the module. |
| KEY_EA | Generated externally during manufacturing. | Entered in plaintext during manufacturing. | Never exits the module. |
| KEY_IA | Generated externally during manufacturing. | Entered in plaintext during manufacturing. | Never exits the module. |
| KEY_PIN | Externally generated and initially installed with the **InstallPIN** APDU command by Issuer. | The PINs are manually entered via the keyboard from the computer that runs the host application. | Never exits the module. |

| | | |
|---|---|---|
| S_ENC<br>S_MAC | The S_ENC and S_MAC are derived with the Single-Step KDF (SP800-56A) from the Key Derived Material which is generated in the EC Diffie-Hellman (P-521) KAS. The hash function used in the KDF is SHA-512. | The keys are generated internally within the module and cannot be imported or exported out of the physical boundary of the module. |
| Session Keys | The session keys can be generated internally by the SP 800-90A DRBG or can be generated by the EC Diffie-Hellman Key Agreement Scheme and the Key Derivation Function to derive the Session keys. | The Session Keys are imported and exported using the other Session Keys or public-private key pair using one of the Approved or Allowed key wrapping methods such as SP 800-38F AES and Triple-DES key wrapping or RSA key wrapping. |
| RSA Private Key<br>RSA Public Key | Generated by the module with the FIPS 186-4 RSA Key Generation method. | The Private or Public Key is imported and exported using the Session Keys with the SP 800-38F AES and Triple-DES key wrapping method. |
| ECDSA Private Key<br>ECDSA Public Key | Generated by the module with the FIPS 186-4 ECDSA Key Generation method. | The Private or Public Key is imported and exported using the Session Keys with the SP 800-38F AES and Triple-DES key wrapping method. |
| Entropy Input | The entropy input is provided by the NDRNG in the hardware. | The entropy input is generated internally within the module and cannot be imported or exported out of the physical boundary of the module. |
| Seed | The seed is generated internally by the SP 800-90A DRBG. | The seed is generated internally within the module and cannot be imported or exported out of the physical boundary of the module. |

*Table 13: Keys/CSPs Generation, Entry, and Output*

## 7.5. Key/CSP Storage and Zeroization

The module stores the keys in the EFLASH memory embedded in the Smart Card chip. Data stored in the EFLASH is protected by the secure design of the Smart Card chip and the enclosure of the module.

The keys stored under the MF file structure can be accessed by the Issuer only. The keys stored under the DF and EF file structure can be accessed by the Issuer, Admin, or User based on the access right of the files.

The key zeroization is performed by filling the memory area with “0xFF”. It is performed immediately after the zeroization APDU commands are called.

The following table describes how the keys/CSPs are stored and zeroized in the module:

| Keys/CSPs | Storage | Zeroization |
|---|---|---|
| KEY_MC | Stored in plaintext in the KF under the MF. | Erased by **FormatDevice** APDU command. |

| | | |
|---|---|---|
| KEY_EA | Stored in plaintext in the KF under the MF. | Erased by **FormatDevice** APDU command. |
| KEY_IA | Stored in plaintext in the KF under the MF. | Erased by **FormatDevice** APDU command. |
| KEY_PIN | Stored in plaintext in the KF under the DF. | 1. Erased by **FormatDevice** APDU command.<br>2. Erased by **Delete File** APDU command.<br>The keys in RAM will be zeroized when deleting the key file. |
| S_ENC<br>S_MAC | Stored in plaintext in RAM. | 1. Erased from RAM during Power off.<br>2. Overwrote by deriving new S_ENC and S_MAC. |
| Session Keys | Stored in plaintext in RAM. | 1. Erased from RAM during Power off.<br>2. Erased by **DestroySessionKey** APDU command.<br>3. Erased from RAM when executing the **ClearSecureState** APDU command to log off.<br>4. Erased from RAM when executing the **Verify PIN** APDU command to switch to another user. |
| RSA Private Key | Stored in plaintext in the EF. | Erased by **Delete File** APDU command. The keys in RAM will be zeroized when deleting the key file. |
| RSA Public Key | Stored in plaintext in the EF. | Erased by **Delete File** APDU command. The keys in RAM will be zeroized when deleting the key file. |
| ECDSA Private Key | 1. Stored in plaintext in the EF.<br>2. Stored in plaintext in RAM for EC Diffie-Hellman. | Erased by **Delete File** APDU command. The keys in RAM will be zeroized when deleting the key file. |
| ECDSA Public Key | 1. Stored in plaintext in the EF.<br>2. Stored in plaintext in RAM for EC Diffie-Hellman. | Erased by **Delete File** APDU command. The keys in RAM will be zeroized when deleting the key file. |
| Entropy Input<br>Seed | The module uses the Entropy Input to generate the seed and stores the seed in plaintext in RAM. | 1. Erased from RAM during Power off.<br>2. The **Uninstantiate** function is called by other DRBG internal functions to clear the internal state of the DRBG including the V and C. |

*Table 14: Keys/CSPs Storage and Zeroization*

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

Lab Name: Bay Area Compliance Laboratories Corp. (Dongguan)

Test Firm Registration: 273710

FCC ID: 2AEXF201505LM

The mToken CryptoID conforms to the EMI/EMC requirements specified by 47 Code of Federal Regulations, Part 15, Subpart B, Unintentional Radiators, Digital Devices, Class B (i.e., for home use).

# 9. Self- Tests

The mToken CryptoID implements a number of self-tests to ensure that proper cryptographic algorithm calculations are implemented in the module. Self-tests include power-up self-tests and conditional tests.

If any self-test fails, the module enters into the Error state. When the module is in the Error state, the LEDs will blink quickly (flashing every 200ms) to indicate that the module is in the Error state. If any conditional test fails, the module will return an error code that matches the status code listed in section 6 of "*mToken CryptoID APDU Specification V1.4*". If the module enters the Error state from FIPS mode, the green LED will blink quickly; if the module enters the Error state from non-FIPS mode, the red LED will blink quickly. Please refer Table 6: LED status for the details of LED status indicator.

When the module is performing self-tests or in the Error state, all data output is prohibited and no cryptographic operation is allowed. No APDU command can be executed except **GetChipID**, **GetFSMaxSpace** and **GetHealthStatus**. In the Error state, the **GetHealthStatus** APDU command returns the value that indicates that the self-tests failed.

The module can recover from the Error state by being unplugged and plugged back into the host GPC to reinitiate the power-up self-tests.

## 9.1.Power-Up Self-Tests

When the mToken CryptoID is powered on, the power-up self-tests are executed automatically without any operator intervention. If the module completes the power-up self-tests successfully, the module will enter either FIPS mode or non-FIPS mode depending on the initial configuration of the module by the Manufacturer or Distributor. If the module enters FIPS mode, a steady green LED will be observed; if the module enters non-FIPS mode, a steady red LED will be observed. The module returns "0000000000000000" from the calling **GetHealthStatus** APDU command which indicates that the power-up self-tests have completed successfully.

If any power-up self-test fails, the module enters into the Error state. All data output is prohibited and no further cryptographic operation is allowed. The LEDs will blink quickly (flashing every 200ms) and the module will return an error code to indicate that the module is in the Error state.

The on-demand self-tests can be invoked by the **HealthCheck** APDU command or by unplugging the token and plugging it back in to reinitiate the power-up self-tests.

The module implements the following Known Answer Tests (KAT), Pair-wise Consistency Tests (PCT), and the Integrity Test during the power-up of the module:

| Algorithm | Test |
|---|---|
| AES | KAT: encryption and decryption are tested separately |
| Triple-DES | KAT: encryption and decryption are tested separately |
| CMAC with AES | KAT |
| CMAC with 3-key Triple-DES | KAT |
| HMAC-SHA-256 | KAT |
| HMAC-SHA-384 | KAT |
| HMAC-SHA-512 | KAT |
| SHA-256 | KAT |
| SHA-384 | KAT |
| SHA-512 | KAT |
| RSA | KAT: signature generation and verification are tested separately. Encryption and decryption are also tested separately. |
| ECDSA | PCT: signature generation and verification are tested separately |
| Hash-based DRBG | KAT |
| EC Diffie-Hellman | KAT: Primitive Z Computation Test |
| Single-Step KDF (Section 5.8.1.1 of SP 800-56A rev2) with SHA-256 | KAT |
| Single-Step KDF (Section 5.8.1.1 of SP 800-56A rev2) with SHA-384 | KAT |
| Single-Step KDF (Section 5.8.1.1 of SP 800-56A rev2) with SHA-512 | KAT |
| Error Detection Code (EDC) using SHA-256 | Integrity Test |

*Table 15: Power-Up Self-Tests*

The mToken CryptoID uses SHA-256 for the integrity test of its firmware. SHA-256 is an Approved hash algorithm that is provided by the module. The known SHA-256 hash value of the firmware is calculated in the factory as part of the binary of the firmware. When the module is powered-up, the module will generate the SHA-256 hash value of the firmware of the module. The module then compares the generated SHA-256 hash value and the known

hash value stored in the firmware binary. If the values do not match, the integrity test fails and the module enters the Error state.

## 9.2. Conditional Tests

The mToken CryptoID performs conditional tests when the module generates random numbers or Public/Private key pairs.

The module implements the following Pair-wise Consistency Tests (PCT) for Public-Private key pair generation and the Continuous Random Number Generator Test (CRNGT):

| Algorithm | Test |
|---|---|
| ECDSA | PCT: Key generation |
| RSA | PCT: Key generation |
| Hash-based DRBG | CRNGT |
| NDRNG | CRNGT |

*Table 16: Conditional Tests*

# 10. Design Assurance

## 10.1. Configuration Management

The module is maintained by SVN for versioning control.

The mToken CryptoID development team utilizes SVN, a software versioning and revision control system, to maintain the current and historical versions of files such as source code and design documentation that contribute to the formation of the module. Each version of the files is uniquely identified by the Revision number in the SVN tool. The Revision number is updated from the previous version to the latest version. SVN integrates several aspects of the software development process in a distributed development environment to facilitate project-wide coordination of development activities across all phases of the product development life cycle, which include:

- Configuration Management – the process of identifying, managing, and controlling software modules as they change over time
- Version Control – the storage of multiple versions of a single file along with details about each version
- Change Control – centralizes the storage of files and controls changes to the files through the process of checking files in and out

Please refer to Table 2: mToken CryptoID Cryptographic Module Components for the final version of the configuration items.

## 10.2. Crypto Officer/User Guidance

The security rules for module usage are listed below for the Crypto Officer and User roles:

- **SecureMessageKAS** service is required to be used to establish a secure channel between the module and the host application running on a GPC before requesting any service.
- According to FIPS 186-4, the RSA key pairs generated for digital signature shall not be used for other purposes such as Key Establishment (i.e., Key Wrapping). In addition, the RSA key pairs for Key Wrapping shall not be used for digital signature generation or verification.
- The operator should check the LED status indicator to determine the state of the module.
- The operators should logout when they finish using the module to maintain securely usage of the module. The operators can logout by calling the **ClearSecureState** APDU command or by unplugging the module from the GPC.

The security rules for module usage are listed below for the Crypto Officer role:

- When installing the KEY_MC and KEY_PIN the “LeftRetryTimes” field is required to be set to 15 so that the module will block the KEY_MC and KEY_PIN after 15 failed login attempts.
- The default value of KEY_MC is defined and exchanged between the Manufacturer and the Distributor as part of the contract. It is the Issuer responsibility to change the default key immediately once he/she receives the module. The Issuer can use the **ChangeSecretKey** APDU command to change the KEY_MC.
- The Issuer should call the **GetData** APDU command to ensure that the module is configured to operate in FIPS mode. If the module is not configured to operate in FIPS mode, the Issuer should refer to section 2.2 “Modes of Operation” for instructions on how to configure the module to operate in FIPS mode.
- The Manufacturer provides a secure delivery and distribution process to main the integrity of the module. The Distributor should inspect the exterior of the delivered package and the tamper tab on each box that contains the modules, and confirm the basic information matching with the received modules. Please refer to document “mToken CryptoID Configuration Management Plan” for more information.
- The Distributor is responsible to provide a secure delivery and distribution procedure to maintain the integrity of the module when the module is distributed to the end user.

The security rules for module usage are listed below for the User role:

- The default values of KEY_PIN (i.e., Admin PIN and User PIN) are required to be securely delivered to the end user by the Distributor to maintain the integrity of the module. It is the end user’s responsibility to change the default KEY_PIN immediately once they receive the module. The end user can use the **ChangeSecretKey** APDU command to change the KEY_PIN.

## 11. Mitigation of Other Attacks

No other attacks are mitigated.

## 12. Glossary and Abbreviations

| | |
|---|---|
| **AES** | Advanced Encryption Specification |
| **APDU** | Application Protocol Data Unit |
| **CAVS** | Cryptographic Algorithm Validation System |
| **CBC** | Cipher Block Chaining |
| **CCID** | Circuit Cards Interface Device |
| **CMAC** | Cipher-based Message Authentication Code |
| **CMVP** | Cryptographic Module Validation Program |
| **CO** | Crypto Officer |
| **CRNGT** | Continuous Random Number Generator Test |
| **CSP** | Critical Security Parameter |
| **DES** | Data Encryption Standard |
| **DF** | Dedicated File |
| **DRBG** | Deterministic Random Bit Generator |
| **EC** | Elliptic Curve |
| **ECB** | Electronic Codebook |
| **ECC CDH** | Elliptic Curve Cryptography Cofactor Diffie-Hellman |
| **ECDSA** | Elliptic Curve Digital Signature Algorithm |
| **EF** | Elementary File |
| **EMI** | Electromagnetic Interference |
| **EMC** | Electromagnetic Compatibility |
| **FCC** | Federal Communications Commission |
| **FIPS** | Federal Information Processing Standards |
| **FSM** | Finite State Model |
| **GPC** | General Purpose Computer |
| **HMAC** | Hash Message Authentication Code |
| **IC** | Integrated Circuit |
| **KAS** | Key Agreement Scheme |
| **KAT** | Known Answer Test |
| **KDF** | Key Derivation Function |
| **HMAC** | Hash Message Authentication Code |
| **KW** | Key Wrap |
| **KTS** | Key Transport Scheme |
| **LED** | Light-Emitting Diode |
| **MF** | Master File |
| **NDRNG** | Non-Deterministic Random Number Generator |
| **NIST** | National Institute of Science and Technology |
| **PCT** | Pairwise Consistency Test |
| **PIN** | Personal Identification Number |
| **PKI** | Public Key Infrastructure |
| **PUB** | Publication |
| **RSA** | Rivest, Shamir, Addleman |
| **SHA** | Secure Hash Algorithm |
| **SP** | Special Publications |
| **SVN** | Apache Subversion |
| **TDES** | Triple DES |
| **TKW** | Triple DES Key Wrap |
| **KW** | Key Wrap |

## 13. Reference

[1] FIPS 140-2 Standard,
http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

[2] FIPS 140-2 Implementation Guidance,
http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402IG.pdf

[3] FIPS 140-2 Derived Test Requirements,
http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402DTR.pdf

[4] FIPS 197, Advanced Encryption Standard (AES),
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[5] FIPS 180-4 Secure Hash Standard (SHS),
http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[6] FIPS 198-1 The Keyed-Hash Message Authentication Code (HMAC),
http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf

[7] FIPS 186-4, Digital Signature Standard,
http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

[8] NIST SP 800-67 Revision 1, Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher,
http://csrc.nist.gov/publications/nistpubs/800-67-Rev1/SP-800-67-Rev1.pdf

[9] NIST SP 800-90A, Recommendation for Random Number Generation Using Deterministic Random Bit Generators,
http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf

[10] NIST SP 800-38B, Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication,
http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf

[11] NIST SP 800-38F, Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping,
http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38F.pdf

[12] NIST SP 800-56A Revision 2, Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography,
http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar2.pdf

[13] NIST SP 800-56B Revision 1, Recommendation for Pair-Wise Key-Establishment Schemes Using Integer Factorization Cryptography,
http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Br1.pdf