ENTRUST

SECURING A WORLD IN MOTION

# nShield Solo XC F2

## Non-proprietary Security Policy for FIPS 140-2 Level 2

Version: 1.1.1

Date: 30/01/2024



nCipher Security Limited
Registered Office: One Station Square,
Cambridge, CB1 2GA, United Kingdom
Registered in England No. 11673268

# Contents

# 1 Introduction

## 1.1 Scope

This document defines the non-proprietary Security Policy enforced by the nShield Hardware Security Module, i.e. the Cryptographic Module, to meet with the security requirements in FIPS 140-2.

The following product hardware variants and firmware version(s) are in scope of this Security Policy.

| Variant name | Marketing model number | Firmware version |
|---|---|---|
| nShield Solo XC F2 | NC3025E-000 | 12.72.1<br>12.72.3 |

Table 1 Variants

All modules are supplied at build standard “A”

## 1.2 Security level

The Cryptographic Module meets overall FIPS 140-2 Security Level 2. The following table specifies the security level in detail.

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

Table 2 Security level of security requirements

## 1.3 Cryptographic module description

The nShield Hardware Security Module (HSM) is a multi-chip embedded Cryptographic Module as defined in FIPS 140-2, which comes in a PCI express board form factor protected by a tamper resistant enclosure, and performs encryption, digital signing, and key management on behalf of an extensive range of commercial and custom-built applications including public key infrastructures (PKIs), identity management systems, application-level encryption and tokenization, SSL/TLS, and code signing.

The Figure below shows the nShield Solo XC HSM.


Figure 1 nShield Solo XC

The cryptographic boundary is delimited in red in the images in the table below. It is delimited by the heat sink and the outer edge of the potting material on the top and bottom of the PCB.

The Cryptographic Module provides the following physical ports and interfaces, which remain outside of the cryptographic boundary:

- PCIe bus (data input/output, control input, status output and power). The services provided by the module are transported through this interface.
- Status LED (status output)
- Mode switch (control input)
- Clear button (control input)
- PS/2 serial connector for connecting a smartcard reader (data input/output).
- 14-way header (data input/output, control input, status output) which provides alternative connections for the mode switch, clear button, status LED and serial connector.
- Dual configuration switches (control input), are a set of two jumpers which enable the mode switch and enable the remote mode switching.
- Battery (power), providing power backup.

- Heat fan control signal.

The PCB traces coming from those connectors transport the signals into the module's cryptographic boundary and cannot be used to compromise the security of the module.

The top cover, heat fan and the battery are outside the module's cryptographic boundary and cannot be used to compromise the security of the module.

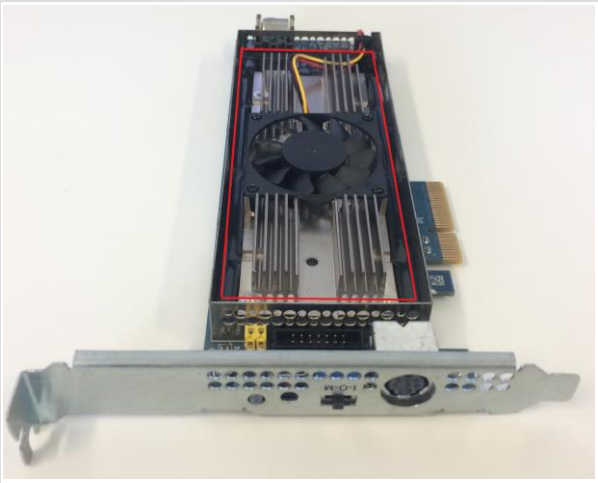
| | |
|---|---|
|  |  |

Table 3 Cryptographic module boundary

## 1.4 Operational environment

The FIPS 140-2 Operational Environment requirements are not applicable because the cryptographic module contains a limited operational environment.

# 2 Cryptographic Functionality

## 2.1 Security World overview

The security model of the module is based around the Security World concept for secure management of cryptographic keys.

A Security World includes:

- An Administrator Card Set (ACS), a set of Administrator smart cards used to perform administrative operations,
- Optionally, one or more Operator Card Sets (OCSs), a set or sets of Operator smart cards used to control access to application keys and to authorise certain operations,
- Optionally, a set of Softcards used to control access to application keys,
- Key Blobs, which contain cryptographic keys and their associated Access Control List (ACL), whose confidentiality and integrity are protected by approved algorithms. They are stored outside the Cryptographic Module.

## 2.2 Keys and Critical Security Parameters

The Cryptographic Module uses and protects the following keys and Critical Security Parameters (CSPs):

| CSP | Type | Description | Generation | Input | Output | Storage | Zeroization |
|---|---|---|---|---|---|---|---|
| **KRE - Recovery Confidentiality Key** | RSA 3072-bit | Key used to protect recovery keys (KR).<br>KTS cert#A1931 | DRBG | Load Blob - encrypted with LT | Make Blob - encrypted with LT | Ephemeral, stored in volatile RAM. | Initialize Unit |
| **KR - Recovery Key** | AES 256-bit | Key used to derive (using SP 800-108 KDF in counter mode) the keys Ke (AES 256-bit) and Km (HMAC-SHA256) that protect an archive copy of an application key.<br>• AES cert #A1931 | DRBG | Load Blob - encrypted with KRE | Make Blob - encrypted with KRE | Ephemeral, stored in volatile RAM. | Initialize Unit, Clear Unit, power cycle or reboot. |
| **Impath session keys** | AES 256-bit in CBC mode.<br>Integrity with HMAC SHA-256. | Used for secure channel between two modules. It consists of a set of four session keys used in an Impath session for encryption, decryption, MAC generation and MAC validation.<br>• AES cert #A1931<br>• HMAC cert #A1931 | 3072-bit DH key exchange | No | No | Ephemeral, stored in volatile RAM. | Clear Unit, new session, power cycle or reboot. |
| **KJSO - JSO key** | DSA 3072-bit | nShield Junior Security Officer key used with its associated certificate to perform the operations allowed by the NSO. | DRBG | Load Blob - encrypted with LT | Make Blob - encrypted with LT | Ephemeral, stored in | Destroy, Initialize Unit, Clear Unit, |

| CSP | Type | Description | Generation | Input | Output | Storage | Zeroization |
|---|---|---|---|---|---|---|---|
| | | • DSA cert #A1931 | | | | volatile RAM. | power cycle or reboot. |
| **KA - Application key** | AES 128, 192, 256 bits<br>TDES 168 bits<br>HMAC with key sizes >= 112 bits<br>RSA with key sizes >= 2048 bits<br>DSA, DH with key sizes >= 2048 bits<br>ECDSA, ECDH, EC MQV with curves:<br>• P-224, P-256, P-384, P-521<br>• K-233, K-283, K-409, K-571<br>• B-233, B-283, B-409, B-571<br>• Brainpool | Keys associated with a user to perform cryptographic operations, that can be used with one of the following validated algorithms:<br>• AES and KTS cert #A1931<br>• HMAC cert #A1931<br>• RSA cert #A1931<br>• DSA cert #A1931<br>• ECDSA cert #A1931<br>• Key Agreement (KAS) cert #A1931<br>• KBKDF cert #A1931<br>• KTS cert #A1931 | DRBG | Load Blob - encrypted with LT or KR | Make Blob - encrypted with LT or KR | Ephemeral, stored in volatile RAM. | Destroy, Initialize Unit, Clear Unit, power cycle or reboot |

| CSP | Type | Description | Generation | Input | Output | Storage | Zeroization |
|---|---|---|---|---|---|---|---|
| **KM - Module Key** | AES 256-bit | Key used to protect logical tokens and associated module Key Blobs.<br>• AES cert #A1931 | DRBG | Load Blob - encrypted with LT | Make Blob - encrypted with LT | Non-volatile memory | Initialize Unit |
| **KML - Module Signing Key** | DSA 3072-bit | Module Signing Key used by the module to sign key generation and module state certificates.<br>When the nShield module is initialized, it automatically generates this key that it uses to sign certificates using DSA with SHA-256. This key is only ever used to verify that a certificate was generated by a specific module.<br>• DSA cert #A1931 | DRBG | No | No | Non-volatile memory | Initialize Unit |
| **KNSO - NSO key** | DSA 3072-bit | nShield Security Officer key used for NSO authorisation and Security World integrity. Used to sign Delegation Certificates and to directly authorize commands during recovery operations<br>• DSA cert #A1931 | DRBG | Load Blob - encrypted with LT | Make Blob - encrypted with LT | Ephemeral, stored in volatile RAM. | Destroy, Initialize Unit, Clear Unit, power cycle or reboot. |
| **LT - Logical Token** | AES 256-bit | Key used to derive the keys that are used to protect token protected key blobs. Logical Tokens are split in shares (encrypted with Share Key) between one or more smartcards or a softcard, using the Shamir Secret Sharing scheme.<br>• AES cert #A1931 | DRBG | Read Share - encrypted with Share Key | Write Share - encrypted with Share Key | Ephemeral, stored in volatile RAM. | Destroy, Initialize Unit, power cycle or reboot |

| CSP | Type | Description | Generation | Input | Output | Storage | Zeroization |
|---|---|---|---|---|---|---|---|
| | | • KBKDF cert #A1931<br>• HMAC cert #A1931 | | | | | |
| **Share Key** | AES 256-bit | Protects a share when written to a smartcard or softcard. This key is used to derive (using SP 800-108 AES CTR KDF) the keys Ke (AES 256-bit) and Km (HMAC-SHA256) that wrap the share.<br>• AES cert #A1931<br>• KBKDF cert #A1931<br>• HMAC cert #A1931 | DRBG | No | No | Ephemeral, stored in volatile RAM. | N/A |
| **Remote Administration session keys** | AES 256-bit in CBC mode<br>Integrity with CMAC | Used for secure channel between the module and a smartcard. This is a set of four AES 256-bit session keys, namely $K_{m-e}$ (for encrypting data send to the smartcard), $K_{c-e}$ (for decrypting data from the smartcard), $K_{m-a}$ (for CMAC generation) and $K_{c-a}$ (for CMAC verification).<br>• AES cert #A1931 | ECDH P-521 key agreement | No | No | Ephemeral, stored in volatile RAM. | Clear Unit, new session, power cycle or reboot. |
| **KAL - Key Audit Logging** | DSA 3072-bit | Used for signing the log trail.<br>• DSA cert #A1931 | DRBG | No | No | Non-volatile memory | Initialize Unit |
| **DRBG internal state** | Hash_DRBG | The module uses the Hash_DRBG with SHA-256 compliant with SP800-90A.<br>• Hash DRBG cert #A1931 | Entropy source | No | No | Ephemeral, stored in | Clear Unit, power cycle or reboot. |

| CSP | Type | Description | Generation | Input | Output | Storage | Zeroization |
|---|---|---|---|---|---|---|---|
| | | | | | | volatile RAM. | |
| **DRBG entropy input** | 344 bits | Entropy input string used to initialize and re-seed the DRBG. | Entropy source | No | No | Ephemeral, stored in volatile RAM. | Clear Unit, power cycle or reboot. |

Table 4 CSP table

The following table describes the public keys handled by the module:

| Public Key | Type | Description | Generation | Input | Output | Storage |
|---|---|---|---|---|---|---|
| **Firmware Integrity key (KFI)** | ECDSA P-521 | Public key used to ensure the integrity of the firmware during boot. The module validates the signature before new firmware is written to non-volatile storage.<br>• ECDSA 805 | At Entrust | Firmware update | No | In firmware |
| **KJWAR** | ECDSA P-521 | Entrust root warranting public key for Remote Administrator Cards and Remote Operator Cards<br>• ECDSA cert #A1931 | At Entrust | Firmware update | None | Persistent storage in plaintext inside the module (EEPROM) |
| **Application keys public key** | See description | Public keys associated with private Application keys:<br>• RSA cert #A1931 | At creation of the | Load Blob - encrypted with LT | Key export | Stored in the key blob of the application key |

| Public Key | Type | Description | Generation | Input | Output | Storage |
|---|---|---|---|---|---|---|
| | | • DSA cert #A1931<br>• ECDSA cert #A1931<br>• Key Agreement (KAS) cert #A1931<br>• KTS cert #A1931 | application key | | | |
| **KJSO public key** | DSA 3072-bit | Public key associated to KJSO<br>• DSA cert #A1931 | At creation of the KJSO | Load Blob - encrypted with LT | Key export | Public key hash stored in the module persistent storage |
| **KNSO public key** | DSA 3072-bit | Public key associated to KNSO<br>• DSA cert #A1931 | At creation of the KNSO | Load Blob - encrypted with LT | Key export | Public key hash stored in the module persistent storage |
| **KML public key** | DSA 3072-bit | Public key associated to KML<br>• DSA cert #A1931 | At creation of KML | No | Key export | Public key hash stored in the module persistent storage |
| **KAL public key** | DSA 3072-bit | Public key associated to KAL<br>• DSA cert #A1931 | At creation of KAL | No | Included in the audit trail | Public key hash stored in the module persistent storage |
| **KRE public key** | RSA 3072-bit | Public key associated to KRE<br>• KTS cert #A1931 | At creation of the KNSO | Load Blob - encrypted with LT | Key export | Stored in a key blob |

| Public Key | Type | Description | Generation | Input | Output | Storage |
|---|---|---|---|---|---|---|
| **FET public key** | DSA 1024-bit | Feature Enable Tool (FET) public key used to verify FET certificates<br>• DSA cert #A1931 | At Entrust | Firmware update | No | Persistent storage in plaintext inside the module (EEPROM) |
| **Impath DH public key** | DH 3072-bit | Public key from peer used in the Impath DH key agreement.<br>• KAS cert #A1931 | No | Loaded with Cmd_ImpathKXFinish | No | Ephemeral, stored in volatile RAM. |
| **Remote Administration ECDH public key** | NIST P-521 | Public key from peer used in the Remote Administration ECDH key agreement.<br>• KAS cert #A1931 | No | Loaded with Cmd_DynamicSlotExchangeAPDUs | No | Ephemeral, stored in volatile RAM. |

Table 5 Public key table

# 2.3 Supported cryptographic algorithms

## 2.3.1 FIPS Approved or Allowed Algorithms

The following tables describe the Approved or allowed cryptographic algorithms supported by the Cryptographic Module.

| Cert # | Algorithm and mode | Standard | Key sizes | Use |
|---|---|---|---|---|
| A1931 | Advanced Encryption Standard (AES)<br>• ECB<br>• CBC<br>• CTR<br>• GCM | FIPS 197<br>SP800-38A<br>SP800-38D | 128 bits<br>192 bits<br>256 bits | Data encryption/decryption |
| A1931 | Triple-DES<br>*Note: The user is responsible to comply with the maximum use of the same key for encryption encryption operations, limited to 2^20 or 2^16, as defined in Implementation Guidance A.13 SP 800-67rev1 Transition.* | SP800-67 | 168 bits | Data encryption/decryption |
| A1931 | KTS<br>• AES Key Wrapping (AES KW)<br>• AES Key Wrapping with Padding (AES KWP)<br>• AES GCM | SP800-38F<br>SP800-38D | 128 bits<br>192 bits<br>256 bits | Key wrapping/unwrapping |
| A1931 | KTS<br>• RSA OAEP | SP 800-56Brev2 | 2048 bits<br>3072 bits<br>4096 bits | Key transport (encapsulation, un-encapsulation) |

| Cert # | Algorithm and mode | Standard | Key sizes | Use |
|---|---|---|---|---|
| | | | Caveat: Cert. #A1931 key establishment methodology provides between 112 and 152 bits of encryption strength. | |
| A1931 | RSA<br>• PKCS#1 v1.5<br>• PSS | FIPS 186-4 | 1024 bits (verification only)<br>2048 bits<br>3072 bits<br>4096 bits | Key generation<br>Signature generation and verification |
| A1931 | Elliptic Curve Digital Signature Algorithm (ECDSA) | FIPS 186-4 | • NIST P-224, P-256, P-384, P-521<br>• NIST K-233, K-283, K-409, K-571<br>• NIST B-233, B-283, B-409, B-571 | Signature generation and verification |
| A1931 | Digital Signature Algorithm (DSA) | FIPS 186-4 | L = 1024 bits, N = 160 bits (verification only)<br>L = 2048 bits, N = 224 bits<br>L = 2048 bits, N = 256 bits<br>L = 3072 bits, N = 256 bits | Signature generation and verification |
| A1931 | HMAC-SHA1<br>HMAC-SHA2 | FIPS 198-1 | >= 112 bits | MAC generation and verification |

| Cert # | Algorithm and mode | Standard | Key sizes | Use |
|---|---|---|---|---|
| A1931 | Advanced Encryption Standard (AES)<br>• CMAC | SP800-38B | 128 bits<br>192 bits<br>256 bits | MAC generation and verification |
| A1931 | KAS-FFC<br>• Diffie-Hellman (DH) | SP 800-56Arev3 | MODP-2048<br>MODP-3072<br>MODP-4096<br>MODP-6144<br>MODP-8192<br>FB<br>FC | Key establishment |
| A1931 | KAS-ECC<br>• Elliptic Curve Diffie-Hellman (ECDH) | SP 800-56Arev3 | • NIST P-224, P-256, P-384, P-521<br>• NIST K-233, K-283, K-409, K-571<br>• NIST B-233, B-283, B-409, B-571 | Key establishment |
| A1931 | KAS-ECC<br>• Elliptic Curve Menezes–Qu–Vanstone (ECMQV) | SP 800-56Arev3 | • NIST P-224, P-256, P-384, P-521<br>• NIST K-233, K-283, K-409, K-571 | Key establishment |

| Cert # | Algorithm and mode | Standard | Key sizes | Use |
|---|---|---|---|---|
| | | | • NIST B-233, B-283, B-409, B-571 | |
| A1931 | Key Based KDF (KBKDF):<br>• counter mode | SP 800-108 | n/a | Key derivation |
| A1931 | SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 | FIPS 180-4 | n/a | Message digest |
| A1931 | Hash-based DRBG | SP 800-90A | n/a | Random bit generation |
| ENT (P) | Hardware based entropy source<br>*This cryptographic module has been validated for compliance with NIST SP 800-90B. Based on noise source testing and analysis, the estimated minimum amount of entropy per the source output bit is about 0.915 bits. The overall amount of generated entropy meets the required security strength of 256 bits based on the entropy per bit and amount of entropy requested by the module.* | SP800-90B | n/a | Random bit generation |
| Vendor affirmed | CKG | SP800-133 | Symmetric keys are generated using the unmodified output of the approved DRBG. | Key generation |
| Bootloader | | | | |
| SHS 3130 | SHA-256, SHA-512 | FIPS 180-4 | n/a | Message digest |

| Cert # | Algorithm and mode | Standard | Key sizes | Use |
|---|---|---|---|---|
| ECDSA 805 | Elliptic Curve Digital Signature Algorithm (ECDSA) | FIPS 186-4 | NIST P-521 | Signature verification |

Table 6 Approved algorithms

| Cert # | Algorithm and mode | Standard | Key sizes | Use |
|---|---|---|---|---|
| A1931 | Elliptic Curve Digital Signature Algorithm (ECDSA) using non-NIST elliptic curves | FIPS 186-4<br>RFC 5639 | • brainpoolP224r1/P224t1 (112 bits of strength)<br>• brainpoolP256r1/P256t1 (128 bits of strength)<br>• brainpoolP320r1/P320t1 (160 bits of strength)<br>• brainpoolP384r1/P384t1 (192 bits of strength)<br>• brainpoolP512r1/P512t1 (256 bits of strength) | Signature generation and verification |
| A1931 | EC Diffie-Hellman using non-NIST elliptic curves | SP 800-56Arev3<br>RFC 5639 | • brainpoolP224r1/P224t1 (112 bits of strength)<br>• brainpoolP256r1/P256t1 (128 bits of strength)<br>• brainpoolP320r1/P320t1 (160 bits of strength) | Key establishment |

| Cert # | Algorithm and mode | Standard | Key sizes | Use |
|---|---|---|---|---|
| | | | • brainpoolP384r1/P384t1 (192 bits of strength)<br>• brainpoolP512r1/P512t1 (256 bits of strength) | |
| A1931 | EC MQV using non-NIST elliptic curves | SP 800-56Arev3<br>RFC 5639 | • brainpoolP224r1/P224t1 (112 bits of strength)<br>• brainpoolP256r1/P256t1 (128 bits of strength)<br>• brainpoolP320r1/P320t1 (160 bits of strength)<br>• brainpoolP384r1/P384t1 (192 bits of strength)<br>• brainpoolP512r1/P512t1 (256 bits of strength) | Key establishment |

Table 7 Allowed algorithms

### 2.3.2 Non-Approved Algorithms

The following table describes the non-approved cryptographic algorithms supported by the Cryptographic Module in non-Approved mode.

| Algorithm |
|---|
| **Symmetric encryption and decryption** |
| DES |

| Algorithm |
| --- |
| Two-key Triple DES encryption, MAC generation |
| AES GCM with externally generated IV |
| AES CBC MAC |
| Aria |
| Camellia |
| Arc Four (compatible with RC4) |
| CAST 256 (RFC2612) |
| SEED (Korean Data Encryption Standard) |
| **Asymmetric** |
| KTS-OAEP-basic with SHA-256 with key size less than 2048 bits |
| ElGamal (encryption using Diffie-Hellman keys) |
| KCDSA (Korean Certificate-based Digital Signature Algorithm) |
| RSA digital signature generation with SHA-1 or key size less than 2048 bits |
| DSA digital signature generation with SHA-1 or key size less than 2048 bits |
| ECDSA digital signature generation with SHA-1 or curves P-192, K-163 , B-163 |
| DH with key size p < 2048 bits or q < 224 bits, or non-compliant with SP800-56Arev3 |
| ECDH with curves P-192, K-163, B-163 or non-compliant with SP800-56Arev3 |
| EC MQV with curves P-192, K-163 or B-163, or non-compliant with SP800-56Arev3 |

| Algorithm |
| --- |
| Deterministic DSA compliant with RFC6979 |
| Ed25519 public-key signature |
| X25519 key exchange |
| ECIES encryption/wrapping and decryption/unwrapping |
| ECKA-EG key agreement |
| **Hash** |
| HAS-160 |
| MD5 |
| RIPEMD-160 |
| Tiger |
| **Message Authentication Codes** |
| HMAC with MD5, RIPEMD-160 and Tiger |
| HMAC with key size less than 112 bits |
| **Other** |
| PKCS#8 padding |
| EMV support:<br>Cryptogram (ARQC) generation and verification (includes EMV2000, M/Chip 4 and Visa Cryptogram Version 14, EMV 2004, M/Chip 2.1, Visa Cryptogram Version 10) |

| Algorithm |
| --- |
| Watchword generation and verification |
| Hyperledger client side KDF |

Table 8 Non-approved algorithms

# 3 Roles and Services

## 3.1 Roles

The Cryptographic Module supports the following roles:

- nShield Security Officer (NSO)
- Junior Security Officer (JSO)
- User

nShield Security Officer (NSO)

This role is represented by Administrator Card holders, which have access to KNSO and are responsible for the overall management of the Cryptographic Module.

To assume this role, an operator or group of operators need to present a quorum *m of N* of smartcards, and the KNSO Key Blob. Each operator is identified by its individual smartcard, which contains a unique logical token share.

Junior Security Officer (JSO)

This role is represented by either Administrator Card or Operator Card holders with a KJSO and an associated Delegation Certificate signed by KNSO, authorising a set of commands.

To assume this role, an operator or group of operators need to present a quorum *m of N* of smartcards and the associated Delegation Certificate. Each operator is identified by its individual smartcard or Softcard, which contains a unique logical token share.

User

This role is represented by Application key owners, which are authorised to perform approved services in the module using those keys.

To assume this role, an operator or group of operators need to present a quorum *m of N* of smartcards or a Softcard, and the Key Blob. Each operator is identified by its individual Smartcard or Softcard, which contains a unique logical token share.

## 3.2 Strength of authentication mechanisms

| Authentication mechanism | Type of authentication | Strength of Mechanism |
|---|---|---|
| Smartcard | Identity based | A logical token share stored in a Smartcard or Softcard is encrypted and MAC'ed. An attacker would need to guess the encrypted share value and the associated MAC in order to be able to load a valid Logical token share into the module. This requires, as a minimum, guessing a 256-bit HMAC-SHA256 value, which gives a probability of 2^-256. This probability is less than 10^-6.<br><br>The module can process around 2^16 commands per minute. This gives a probability of success in a one minute period of 2^-240, which is less than 10^-5. |
| Softcard | Identity based | |

Table 9 Strength of authentication table

## 3.3 Services

The following table describes the services provided by the Cryptographic Module and the access policy.

The Access column presents the access level given to the CSP, R for Read, W for Write, Z for Zeroise

| Service | Description | Authorized roles | Access | CSPs |
|---|---|---|---|---|
| **Big number operation**<br><br>Cmd_BignumOp | Performs an operation on a large integer. | Unauthenticated | - | None |
| **Make Blob**<br><br>Cmd_MakeBlob | Creates a Key blob containing the key. Note that the key ACL needs to authorize the operation. | User / JSO / NSO | W | KA, KRE, KR, KJSO, KM, KNSO, LT |
| **Buffer operations**<br><br>Cmd_CreateBuffer<br><br>Cmd_LoadBuffer | Mechanism for loading of data into the module volatile memory. The data can be loaded in encrypted form which can be decrypted inside the module with a key that has been previously loaded. | Unauthenticated | R | KA |

| Service | Description | Authorized roles | Access | CSPs |
|---|---|---|---|---|
| **Bulk channel**<br>Cmd_ChannelOpen<br>Cmd_ChannelUpdate | Provides a bulk processing channel for encryption / decryption, MAC generation / verification and signature generation / verification. | User | R | KA |
| **Check User Action**<br>Cmd_CheckUserAction | Determines whether the ACL associated with a key allows a specific operator defined action. | User / JSO / NSO | R | KNSO, KJSO; KA |
| **Clear Unit**<br>Cmd_ClearUnit | Zeroises all keys, tokens and shares that are loaded into the module. Will cause the module to reboot and perform self-tests. | Unauthenticated | Z | KA, KR, Impath keys, KJSO, remote administration session keys |
| **Set Module Key**<br>Cmd_SetKM | Allows a key to be stored internally as a Module key (KM) value. The ACL needs to authorize this operation. | NSO | W | KM |
| **Remove Module Key**<br>Cmd_RemoveKM | Deletes a given KM from non-volatile memory. | NSO | Z | KM |
| **Duplicate key handle**<br>Cmd_Duplicate | Creates a second instance of a Key with the same ACL and returns a handle to the new instance.<br>Note that the source key ACL needs to authorize this operation. | User / JSO / NSO | R | KA |
| **Enable feature**<br>Cmd_StaticFeatureEnable | Enables the service. This service requires a certificate signed by | Unauthenticated | - | None |

| Service | Description | Authorized roles | Access | CSPs |
|---|---|---|---|---|
| | the Master Feature Enable key. | | | |
| **Encryption / decryption**<br>Cmd_Encrypt<br>Cmd_Decrypt | Encryption and decryption using the provided key handle. | User | R | KA |
| **Erase from smartcard /softcard**<br>Cmd_EraseFile<br>Cmd_EraseShare | Removes a file or a share from a smartcard or softcard | NSO / JSO / User | - | None |
| **Format Token**<br>Cmd_FormatToken | Formats a smartcard or a softcard. | Unauthenticated | - | None |
| **File operations**<br>Cmd_FileCopy<br>Cmd_FileCreate<br>Cmd_FileErase<br>Cmd_FileOp | Performs file operations in the module. | NSO / JSO | - | None |
| **Firmware Authenticate**<br>Cmd_FirmwareAuthenticate | Reports firmware version, using a zero knowledge challenge response protocol based on HMAC.<br>The protocol generates a random value to use as the HMAC key. | Unauthenticated | - | None |
| **Force module to fail**<br>Cmd_Fail | Causes the module to enter a failure state. | Unauthenticated | - | None |
| **Foreign Token open**<br>Cmd_ForeignTokenOpen | Opens a channel for direct data access to a Smartcard<br>Requires Feature Enabled. | NSO / JSO | - | None |
| **Foreign Token command**<br>Cmd_ForeignTokenCommand | Sends an ISO-7816 command to a smartcard over the | Unauthenticated | - | None |

| Service | Description | Authorized roles | Access | CSPs |
|---|---|---|---|---|
| | channel opened by ForeignTokenOpen. | | | |
| **Firmware Update**<br>Cmd_Maintenance<br>Cmd_ProgrammingBegin<br>Cmd_ProgrammingBeginChunk<br>Cmd_ProgrammingLoadBlock<br>Cmd_ProgrammingEndChunk<br>Cmd_ProgrammingEnd<br>Cmd_ProgrammingGetKeyList | Perform a firmware update. Restricted service to Entrust signed Firmware. | Unauthenticated | R | KFI |
| **Generate prime number**<br>Cmd_GeneratePrime | Generates a random prime number. | Unauthenticated | R, W | DRBG internal state |
| **Generate random number**<br>Cmd_GenerateRandom | Generates a random number from the Approved DRBG. | Unauthenticated | R, W | DRBG internal state |
| **Get ACL**<br>Cmd_GetACL | Get the ACL of a given key. | User | R | KA |
| **Get key application data**<br>Cmd_GetAppData | Get the application data field from a key. | User | R | KA |
| **Get challenge**<br>Cmd_GetChallenge | Get a random challenge that can be used in fresh certificates. | Unauthenticated | R, W | DRBG internal state |
| **Get KLF2**<br>Cmd_GetKLF2 | Get a handle to the Module Long Term (KLF2) public key. | Unauthenticated | - | None |
| **Get Key Information**<br>Cmd_GetKeyInfo<br>Cmd_GetKeyInfoEx | Get the type, length and hash of a key. | NSO / JSO / User | R | KA |
| **Get module signing key**<br>Cmd_GetKML | Get a handle to the KML public key. | Unauthenticated | R | KML |
| **Get list of slot in the module**<br>Cmd_GetSlotList | Get the list of slots that are available from the module. | Unauthenticated | - | None |

| Service | Description | Authorized roles | Access | CSPs |
|---|---|---|---|---|
| **Get Logical Token Info**<br>Cmd_GetLogicalTokenInfo<br>Cmd_GetLogicalTokenInfoEx | Get information about a Logical Token: hash, state and number of shares. | NSO / JSO / User | R | LT |
| **Get list of module keys**<br>Cmd_GetKMList | Get the list of the hashes of all module keys and the KNSO. | Unauthenticated | R | KM, KNSO |
| **Get module state**<br>Cmd_GetModuleState | Returns unsigned data about the current state of the module. | Unauthenticated | - | None |
| **Get real time clock**<br>Cmd_GetRTC | Get the current time from the module Real Time Clock. | Unauthenticated | - | None |
| **Get share access control list**<br>Cmd_GetShareACL | Get the Share's ACL. | NSO / JSO / User | R | Share Key |
| **Get Slot Information**<br>Cmd_GetSlotInfo | Get information about shares and files on a Smartcard that has been inserted in a module slot. | Unauthenticated | - | None |
| **Get Ticket**<br>Cmd_GetTicket | Get a ticket (an invariant identifier) for a key. This can be passed to another client or to a SEE World which can redeem it using Redeem Ticket to obtain a new handle to the object. | NSO / JSO / User | - | None |
| **Initialize Unit**<br>Cmd_InitializeUnit<br>Cmd_InitializeUnitEx | Causes a module in the pre-initialization state to enter the initialization state. When the module enters the initialization state, it erases all Module keys (KM), the | Unauthenticated | Z | KA, KRE, KR, KJSO, KM, KAL, KML, KNSO, LT |

| Service | Description | Authorized roles | Access | CSPs |
|---|---|---|---|---|
| | module's signing key (KML), and the hash of the Security Officer's keys, HKNSO. It then generates a new KML and KM. | | | |
| **Insert a Softcard**<br>Cmd_InsertSoftToken | Allocates memory on the module that is used to store the logical token share and other data objects. | Unauthenticated | R | Share Key |
| **Remove a Softcard**<br>Cmd_RemoveSoftToken | Removes a Softcard from the module. It returns the updated shares and deletes them from the module's memory. | Unauthenticated | Z | Share Key |
| **Impath secure channel**<br>Cmd_ImpathGetInfo<br>Cmd_ImpathKXBegin<br>Cmd_ImpathKXFinish<br>Cmd_ImpathReceive<br>Cmd_ImpathSend | Support for Impath secure channel. Requires Feature Enabled. | NSO / JSO / User | R, W | KML, Impath keys |
| **Key generation**<br>Cmd_GenerateKey<br>Cmd_GenerateKeyPair | Generates a cryptographic key of a given type with a specified ACL. It returns a handle to the key. Optionally, it returns a KML signed certificate with the hash of the key and its ACL information. | Unauthenticated | R, W | KML, DRBG internal state, KA, KJSO, |
| **Key import**<br>Cmd_Import | Loads a plain text key into the module. | Unauthenticated | R | KA, KJSO |
| **Derive Key** | Performs key wrapping, | NSO / JSO / User | R, W | KA, KJSO |

| Service | Description | Authorized roles | Access | CSPs |
|---|---|---|---|---|
| Cmd_DeriveKey | unwrapping, transport and derivation. The ACL needs to authorize this operation. | | | |
| **Load Blob**<br>Cmd_LoadBlob | Load a Key blob into the module. It returns a handle to the key suitable for use with module services. | NSO / JSO / User | W | KA, KRE, KR, KJSO, KM, KNSO |
| **Load Logical Token**<br>Cmd_LoadLogicalToken | Initiates loading a Logical Token from Shares, which can be loaded with the Read Share command. | Unauthenticated | - | None |
| **Generate Logical Token**<br>Cmd_GenerateLogicalToken | Creates a new Logical Token with given properties and secret sharing parameters. | Unauthenticated | W | KM, LT, KJSO |
| **Message digest**<br>Cmd_Hash | Computes the cryptographic hash of a given message. | Unauthenticated | - | None |
| **Modular Exponentiation**<br>Cmd_ModExp<br>Cmd_ModExpCrt<br>Cmd_RSAImmedVerifyEncrypt<br>Cmd_RSAImmedSignDecrypt | Performs a modular exponentiation (standard or CRT) on values supplied with the command. | Unauthenticated | - | None |
| **Module hardware information**<br>Cmd_ModuleInfo | Reports detailed hardware information. | Unauthenticated | - | None |
| **No Operation**<br>Cmd_NoOp | No operation. | Unauthenticated | - | None |
| **Change Share Passphrase**<br>Cmd_ChangeSharePIN<br>Cmd_ChangeShareGroupPIN | Updates the passphrase of a Share. | NSO / JSO / User | R, W | Share Keys |
| **NVRAM Allocate** | Allocation in NVRAM. | NSO / JSO | - | None |

| Service | Description | Authorized roles | Access | CSPs |
|---|---|---|---|---|
| Cmd_NVMemAllocate | | | | |
| **NVRAM Free**<br>Cmd_NVMemFree | Deallocation from NVRAM. | NSO / JSO | - | None |
| **Operation on NVM list**<br>Cmd_NVMemList | Returns a list of files in NVRAM. | Unauthenticated | - | None |
| **Operation on NVM files**<br>Cmd_NVMemOp | Operation on an NVRAM file. | Unauthenticated | | None |
| **Key export**<br>Cmd_Export | Exports a key in plain text. | NSO / JSO / User | R | KA |
| **Pause for notifications**<br>Cmd_PauseForNotifications | Wait for a response from the module. | Unauthenticated | | None |
| **Read file**<br>Cmd_ReadFile | Reads data from a file on a Smartcard or Softcard. The ACL needs to authorize this operation. | NSO / JSO | - | None |
| **Read share**<br>Cmd_ReadShare | Reads a share from a Smartcard or Softcard. Once a quorum of shares have been loaded, the module re-assembles the Logical Token. | NSO / JSO / User | R | Share Keys, LT |
| **Send share to remote slot**<br>Cmd_SendShare | Reads a Share and encrypts it with the Impath session keys for transmission to the peer module. | NSO / JSO / User | R | Impath Keys, Share Keys |
| **Receive share from remote slot**<br>Cmd_ReceiveShare | Receives a Share encrypted with the Impath session keys by a remote module. | NSO / JSO / User | R | Impath Keys, Share Keys |
| **Redeem Ticket**<br>Cmd_RedeemTicket | Gets a handle in the current name space for the object referred to by a ticket | NSO / JSO / User | - | None |

| Service | Description | Authorized roles | Access | CSPs |
|---|---|---|---|---|
| | created by Get Ticket. | | | |
| **Remote Administration**<br><br>Cmd_DynamicSlotCreateAssociation<br>Cmd_DynamicSlotExchangeAPDUs<br>Cmd_DynamicSlotsConfigure<br>Cmd_DynamicSlotsConfigureQuery<br>Cmd_VerifyCertificate | Provides remote presentation of Smartcards using a secure channel between the module and the Smartcard. | NSO / JSO / User | R, W | Remote administration session keys |
| **Destroy**<br>Cmd_Destroy | Remove handle to an object in RAM. If the current handle is the only one remaining, the object is deleted from RAM. | Unauthenticated | Z | KA, KJSO, KNSO, LT |
| **Report statistics**<br>Cmd StatGetValues<br>Cmd_StatEnumTree | Reports the values of the statistics tree. | Unauthenticated | - | None |
| **Show Status**<br>Cmd_NewEnquiry | Report status information. | Unauthenticated | - | None |
| **Secure Execution Engine**<br>Cmd_CreateSEEWorld<br>Cmd_GetWorldSigners<br>Cmd_SEEJob<br>Cmd_SetSEEMachine<br>Cmd_TraceSEEWorld | Creation and interaction with SEE machines. | NSO / JSO | - | None |
| **Set ACL**<br>Cmd_SetACL | Replaces the ACL of a given key with a new ACL. The ACL needs to authorize this operation. | NSO / JSO / User | W | KA |
| **Set key application data**<br>Cmd_SetAppData | Writes the application information field of a key. | User | W | KA |

| Service | Description | Authorized roles | Access | CSPs |
|---|---|---|---|---|
| **Set NSO Permissions**<br>Cmd_SetNSOPerms | Sets the NSO key hash and which permissions require a Delegation Certificate. | NSO | - | None |
| **Set real time clock**<br>Cmd_SetRTC | Sets the Real-Time Clock value. | NSO / JSO | - | None |
| **Signature generation**<br>Cmd_Sign | Generate a digital signature or MAC value. | NSO / JSO / User | R | KA, KNSO, KJSO |
| **Sign Module State**<br>Cmd_SignModuleState | Returns a signed certificate that contains data about the current configuration of the module. | Unauthenticated | R | KML |
| **Signature verification**<br>Cmd_Verify | Verifies a digital signature or MAC value. | NSO / JSO / User | R | KA |
| **Write file**<br>Cmd_WriteFile | Writes a file to a Smartcard or Softcard. | NSO / JSO | - | None |
| **Write share**<br>Cmd_WriteShare | Writes a Share to a Smartcard or Softcard. | Unauthenticated | - | None |

Table 10 Service table

# 4 Physical Security

The product is a multi-chip embedded Cryptographic Module, as defined in FIPS 140-2. It is enclosed in a hard and opaque epoxy resin which meets the physical security requirements of FIPS 140-2 level 3.

Note: The module hardness testing was only performed at a single temperature and no assurance is provided for Level 3 hardness conformance at any other temperature.

To ensure physical security, the module should be inspected periodically for evidence of tamper attempts:

- Examine the entire PCIe board including the epoxy resin security coating for obvious signs of damage.
- Examine the heat sink on top of the module and also the potting which binds the edges of the heat sink for obvious signs of damage.
- Examine the smartcard reader and ensure it is directly plugged into the module or into the port provided by any appliance in which the module is integrated and the cable has not been tampered with.

The module has a clear button. Pressing this button puts the module into the self-test state, clearing all stored key objects, Logical Tokens and impath keys and running all self-tests. The long term security critical parameters, NSO's key, module keys and module signing key can be cleared by returning the module to the factory state.

# 5 Rules

This section describes how to accept, initialise and operate the module in the FIPS approved mode.

## 5.1 Delivery

The nShield Cryptographic Module is sent to the customers using a standard carrier service. After accepting the delivery of the module, the Crypto Officer shall perform a physical inspection of the module (refer to Physical Security). This inspection is done to ensure that the module has not been tampered with during transit. If the inspection results indicate that the module has not been tampered with, the Crypto Officer can then proceed with installation and configuration of the module.

The module must be installed and configured according to the User Guides and the Initialization procedures described below.

## 5.2 Initialization procedures

To configure the Cryptographic Module in FIPS approved mode, the following steps must be followed:

1. Put the module in pre-initialization mode.
2. Create a FIPS 140-2 level 2 compliant Security World using Entrust supplied utility `new-world`. Omitting the mode flag will create a Security World compliant with FIPS 140-2 Level 2.
3. Put the module in Operational mode.

An operator can verify that the module is configured in FIPS approved mode with the command line utility `enquiry`, which reports the following active modes:

```
active modes          UseFIPSApprovedInternalMechanisms AlwaysUseStrongPrimes
```

or

```
active modes          UseFIPSApprovedInternalMechanisms
```

## 5.3 Creation of new Operators

New User

To create a new User, the following steps must be followed:

1. Authenticate as NSO or JSO role.

2. Create a new Logical Token, LTU.
3. Split the LTU into one or more smartcards or a Softcard.
4. Generate a new Application key with the ACL configured so that the key can only be blobbed under LTU.
5. Generate a Key Blob for the Application key protected by LTU.
6. Give to the Operator the Key Blob, the Operator Cards or Softcard.

New Junior Security Officer (JSO)

To create a new JSO, the following steps must be followed:

1. Authenticate as NSO or JSO role.
2. Generate a new Logical Token, LTJSO.
3. Split LTJSO into one or more smartcards or Softcard.
4. Generate a new asymmetric key pair (KJSOpriv, KJSOpub):
    a. Set the ACL of KJSOpriv to allow Sign and UseAsSigningKey,
    b. Set the ACL of KJSOpub to allow ExportAsPlain
5. Generate a Key Blob for KJSOpriv protected by LTJSO
6. Export KJSOpub.
7. Create a Delegation Certificate signed by NSO or an already existing JSO, which includes KJSOpriv as the certifier and authorises the following actions
    a. OriginateKey, which authorises generation of new keys,
    b. GenerateLogToken, which authorises the creation of new Logical Tokens,
    c. ReadFile, WriteFile,
    d. FormatToken.
8. Give the Operator the Certificate, the Key Blob, the smartcards or Softcard.

# 6 Self tests

The Cryptographic Module performs power-up and conditional self-tests. It also supports power-up self-tests upon request by resetting the module, either by pressing the Clear button or by sending the Clear Unit command.

In the event of a self-test failure, the module enters an error state which is signalled by the SOS morse pattern flashing in the output LED. While in this state, the module does not process any commands.

## 6.1 Power-up self-tests

In the self-test state the module clears the RAM, thus ensuring any loaded keys or authorization information is removed and then performs the following:

- Power-up self-test on hardware components,
- Firmware integrity verification,
- Cryptographic self-tests as specified below.

| Algorithm | Description |
|---|---|
| **Boot Loader** | |
| SHA512 | Known Answer Test |
| ECDSA | Known Answer Test (verification only) with curve P-521 |
| **Firmware** | |
| AES | Known Answer Test: ECB encryption and decryption with 128, 192 and 256-bit keys |
| AES CMAC | Known Answer Test: 128-bit key |
| TDES | Known Answer Test: ECB encryption and decryption with 168-bit keys |
| TDES CBC MAC | Known Answer Test: 168-bit key |
| SHA1 | SHA1 KAT test, other size are tested along with KAT HMAC |
| HMAC with SHA1, SHA224, SHA256, SHA384, SHA512 | Known Answer Test |
| RSA | Known Answer Test: sign/verify, encrypt/decrypt with 2048-bit key<br>Pair-Wise consistency test: sign/verify |
| DSA | Known Answer Test: sign/verify with 2048-bit key<br>Pair-Wise consistency test: sign/verify |

| Algorithm | Description |
|---|---|
| ECDSA | Pair-Wise consistency test: sign/verify with curves P-224 and B-233 |
| Key Agreement | Shared Secret Computation Known Answer Test for DH |
| Key Agreement | Shared Secret Computation Known Answer Test for ECDH with curves P-384 and B-233 |
| One-step KDF | Known Answer Test with SHA-256 auxiliary function |
| Two-step KDF | Known Answer Test with HMAC-SHA256 auxiliary function |
| KBKDF | Known Answer Test |
| DRBG | Health Tests according to SP 800-90A |
| **Other** | |
| Entropy source | SP800-90B section 4.4 health tests: adaptive proportion test and repetition count test |

Table 11 Cryptographic algorithm self-tests

## 6.2 Conditional self-tests

The module performs pair-wise consistency checks when RSA, DSA and ECDSA keys are generated and the continuous test on the entropy source.

## 6.3 Firmware load test

The Cryptographic Module supports firmware upgrades in the field, with authenticity, integrity and roll-back protection for the code. Entrust provides signed firmware images with the Firmware Integrity Key.

The module performs the following actions before replacing the current image:

- Code signature verification with the public Firmware Integrity Key.
- Image decryption with the Firmware Confidentiality Key.
- Verification that the Version Security Number (VSN) of the new image is not less than the VSN of the current image.

Note: updating the firmware to a non-FIPS validated version of the firmware will result in the module operating in a non-Approved mode.

# Contact Us

| | |
|---|---|
| **Web site** | https://www.entrust.com |
| **Support** | https://nshieldsupport.entrust.com |
| **Email Support** | nShield.support@entrust.com |
| **Online documentation:** | Available from the Support site listed above. |

You can also contact our Support teams by telephone, using the following numbers:

### Europe, Middle East, and Africa

| | |
|---|---|
| United Kingdom: | +44 1223 622 444<br>One Station Square<br>Cambridge<br>CB1 2GA<br>UK |

### Americas

| | |
|---|---|
| Toll Free: | +1 833 425 1990 |
| Fort Lauderdale: | +1 954 953 5229<br>Sawgrass Commerce Center – A<br>Suite 130,<br>13800 NW 14 Street<br>Sunrise<br>FL 33323 USA |

### Asia Pacific

| | |
|---|---|
| Australia: | +61 9126 9070<br>World Trade Centre Northbank Wharf<br>Siddeley St<br>Melbourne VIC 3005<br>Australia |
| Japan: | +81 50 3196 4994 |
| Hong Kong: | +852 3008 4994<br>31/F, Hysan Place<br>500 Hennessy Road<br>Causeway Bay<br>Hong Kong |

To get help with
Entrust nShield HSMs

nShield.support@entrust.com

nshieldsupport.entrust.com

## ABOUT ENTRUST CORPORATION

Entrust keeps the world moving safely by enabling trusted identities, payments and data protection. Today more than ever, people demand seamless, secure experiences, whether they're crossing borders, making a purchase, accessing e-government services or logging into corporate networks. Entrust offers an unmatched breadth of digital security and credential issuance solutions at the very heart of all these interactions. With more than 2,500 colleagues, a network of global partners, and customers in over 150 countries, it's no wonder the world's most entrusted organizations trust us.

ENTRUST

SECURING A WORLD IN MOTION