nCipher

www.ncipher.com

# Security Policy

nToken

**Version:** **4.6**

**Date:** **29 October 2019**



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

# Contents

# 1 Purpose

nToken is a FIPS 140-2 level 2 module designed to protect a DSA key used to authenticate a host computer to an nShield Connect.

This authentication is made by signing a nonce message to prove to the nShield Connect that the session was instigated by a client running on the host. Though it inherits additional restricted HSM capabilities from the nShield family.

The nShield nToken Hardware Security Modules are defined as multi-chip embedded cryptographic modules as defined by FIPS PUB 140-2.

| Unit ID | Model Number | Real Time Clock (RTC) NVRAM | Secure Execution Environment (SEE) | Potting (epoxy resin) | EMC classification | Crypto Accelerator |
|---|---|---|---|---|---|---|
| nToken | nC2023E-000 | No | No | Yes | A | No |

All modules are now supplied at build standard “N” to indicate that they meet the latest EU regulations regarding ROHS.

The modules run firmware provided by nCipher Security. There is the facility for the factory to upgrade this firmware. In order to determine that the module is running the correct version of firmware they should use the **New Enquiry** service which reports the version of firmware currently loaded. The validated firmware version is 2.51.10-2 and 2.55.1-2.

The modules must be accessed by a custom written application. Full documentation for the nCore API can be downloaded from the nCipher web site.

The module can be connected to a computer running one of the following operating systems:

- Windows
- Solaris
- HP-UX
- AIX
- Linux x86 / x64

Windows was used to test the module for this validation.

| Section | Level |
|---|---|
| 1. Cryptographic Module Specification | 2 |
| 2. Cryptographic Module Ports and Interfaces | 2 |
| 3. Roles, Services, and Authentication | 2 |
| 4. Finite State Model | 2 |
| 5. Physical Security | 3 |
| 6. Operational Environment | N/A |
| 7. Cryptographic Key Management | 2 |
| 8. EMI/EMC | 3 |
| 9. Self-Tests | 2 |
| 10. Design Assurance | 2 |
| 11. Mitigation of Other Attacks | N/A |
| Overall FIPS Level | 2 |

## 1.1 Initializing the nToken

When the module is put in Initialisation mode using the Mode switch, it generates a random AES 256 bit key for use as a module key. This key is stored in the module's EEPROM encrypted under a Boardkey and is never revealed. This step is usually performed in the nCipher factory.

In order to enrol an nToken into an nShield Connect, the administrator runs the initunit and nTokenEnrol utilities on the host computer.

The following steps are performed:

1. The initunit utility calls the Generate Key service in the nToken module to generate a DSA key pair.
2. Once the key pair is generated inside the nToken module, the initunit utility exports the private key in a Key Blob protected by the module key (AES 256 bit), using the Make Blob service from the nToken module, and stores the Key Blob in the host's hard disk. The public key is exported in plaintext using the Export service and is also stored in the host computer's hard disk.
3. initunit exports a certificate containing the public key and the nToken's Electronic Serial Number to the nShield Connect.
4. nTokenEnrol is used to display the SHA-1 hash of the DSA public key on the host computer's display, to enable the administrator to key the value into the nShield Connect front panel and writes the public key and nToken ESN to a configuration file that is used to enrol the nToken into an nShield Connect.

During impath set-up, the hardserver attached to the nToken signs its setup message with the key. It also includes the public key in the message. Hence the nShield Connect can verify that the hardserver it's talking to is attached to the nToken it was told about.

## 1.2 Using the nToken

The nToken is used when a operator wishes to open a connection from a host application to an nShield Connect. When the operator attempts to open such a connection, the host server containing the nToken obtains a nonce from the nShield Connect and has the nToken sign a message containing this nonce to confirm the identity of the computer the application is running on. The host server sends this message to the nShield Connect. The nShield Connect verifies the signature with the nToken's public key and can then determine whether the host is authorized.

Although the nToken uses the same firmware image as nShield modules, nToken modules are factory configured so that they can only perform a limited subset of operations. See nCipher Master Feature Enable Key for more details.

The host server uses the Show Status service to determine which attached modules are nToken modules and which are nShields. If the operator requests a service that the nToken cannot perform, the server ensures the command is routed to an nShield and not an nToken.

# 2 Ports and Interfaces

The module has the following physical ports:

- 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)
- Clear button (control input)
- Mode switch traces (control input)

The New Enquiry command provides the status interface.

# 3 Roles

The module has three roles that are implicitly assumed for all services: unauthenticated, user and administrator.

## 3.1 Unauthenticated

Connections are initially unauthenticated.

An operator in the unauthenticated role does not have access to handles or tickets required to provide access to the CSPs of authenticated users.

## 3.2 User

The User role performs cryptographic operations using keys retrieved from key blobs.

To assume the User role, the operator loads a valid key blob, which is the authentication factor. If the key blob loads correctly, the module returns a **ObjectID**, which is a handle for the key ready to be used for cryptographic operations inside the module.

## 3.3 Administrator

The Administrator role is responsible for the module feature enabling capability. It corresponds to the FIPS crypto officer.

To assume the Administrator role, the operator presents a valid nCipher signed Feature Enable Certificate.

After the module has been feature enabled to operate as an nToken, there is no further requirement for the administrator role to interact with the module and all further services interaction is performed by Unauthenticated or User Role.

# 4 Services available to each role

This section describes all the services supported by the module. The role that is assumed when carrying out the service is identified.

Note: No additional initialization is required to operate in the Approved or non-Approved mode. The module can alternate service by service between Approved and non-Approved modes of operation based on the cryptographic functions selected for each service. For each Service, the Approved and non-Approved use of cryptographic functions is identified. For the list of Approved algorithms and key sizes, See Supported Algorithms on page 31.

| Key Access | Description |
|---|---|
| Create | Creates a in-memory object, but does not reveal value. |
| Erase | Erases the object from memory, smart card or non-volatile memory without revealing value |
| Export | Discloses a value, but does not allow value to be changed. |
| Report | Returns status information |
| Set | Changes a CSP to a given value |
| Use | Performs an operation with an existing CSP - without revealing or changing the CSP |

| Command / Service | Role | | | Description | Key/CS P access | Approved | |
|---|---|---|---|---|---|---|---|
| | Unauth | User | Admin | | | | Non-approved |
| Bignum Operation | Yes | Yes | Yes | Performs simple mathematical operations. | No access to keys or CSPs | None. | None. |
| Channel Open | handle, ACL | handle, ACL | handle, ACL | Opens a communication channel which can be used for bulk encryption, decryption, signing or hashing. | *Uses* a key object | Encryption/Decryption: AES all key sizes, Triple-DES 3-Key.<br>Signature: DSA and ECDSA with compliant key sizes.<br>Hash: SHA-1, SHA-256, SHA- 384, SHA-512<br>Message auth:<br>HMAC with SHA1, SHA-256, SHA-384, SHA-512 | Encryption/Decryption: Triple-DES 2-Key, DES, ArcFour, Aria, Camellia, SEED<br>Signature: RSA all key sizes; DSA and ECDSA with non-compliant key sizes; KCDSA.<br>Hash: HAS-160, MD5, RIPEMD160.<br>Message auth: HMAC with HAS-160, MD5 and RIPEMD160 |
| Channel Update | handle | handle | handle | Performs encryption, decryption, signing or hashing on a previously opened channel. The operation and key are specified in **ChannelOpen**. | *Uses* a key object | As above. | As above. |
| Clear Unit | Yes | Yes | Yes | Causes the module to reset and will trigger the Self-tests. Erases all loaded keys. Clear Unit does not erase long term keys, such as module keys, these are stored encrypted or wrapped. | *Erases* objects. | None. | None. |
| Create Buffer | No | cert | Yes | Allocates an area of memory to load data. If the data is encrypted, this service specifies the encryption key and IV used. The decrypt operation is performed by **LoadBuffer** | *Uses* a key object | Decryption: AES all key sizes, Triple-DES 3-Key. | Decryption: DES, ElGamal, ArcFour, Aria, Camellia, SEED |
| Decrypt | handle, ACL | handle, ACL | handle, ACL | Decrypts a cipher text with a stored key returning the plain text. | Uses a key object | Decryption: AES and Triple-DES all key sizes | Decryption: DES, ElGamal, ArcFour, Aria, Camellia, SEED |

| Command / Service | Role | | | Description | Key/CS P access | Approved | |
|---|---|---|---|---|---|---|---|
| | Unauth | User | Admin | | | | Non-approved |
| Derive Key | handle, ACL | handle, ACL | handle, ACL | Creates a new key object from a variable number of other keys already stored on the module and returns a handle for the new key. This service can be used to split, or combine, encryption keys. This service is used to wrapkeys according to the KDP so that a key server can distribute the wrapped key to micro-HSM devices.<br>The **DeriveKey** service does not provide key derivation in the sense understood by FIPS 140-2. | *Uses* a key object, create a new key object. | Key Establishment: AES all key sizes; Triple-DES 3-Key in CBC mode; DH, ECDH and ECMQV<br>using compliant key sizes.<br>Key Derivation: KDF AES-CTR | Key establishment: AES KW;<br>AES CMAC in CTR mode KDF (SP 800-108); Triple- DES 2-Key; RSA key encapsulation, DH, ECDH and ECMQV using non- compliant key sizes; PKCS #8, SSL/TLS master key derivation, XOR key split, DLIES (D/H plus Triple-DES or D/H plus AES), Aria, Arc Four, Camellia, DES, SEED |
| Destroy | handle | handle | handle | Removes an object, if an object has multiple handles as a result of **RedeemTicket** service, this removes the current handle. | *Erases* any key object. | None. | None. |
| Duplicate | handle, ACL | handle, ACL | handle, ACL | Creates a second instance of a key object with the same ACL and returns a handle to the new instance. | *Creates* a new key object. | None. | None. |
| Encrypt | handle, ACL | handle, ACL | handle, ACL | Encrypts a plain text with a stored key returning the cipher text. | *Uses* a key object | Encryption: AES all key sizes; Triple-DES 3-Key | Encryption:<br>Triple-DES 2-Key, DES, RSA, El Gamal, Aria, Arc Four, Camellia, SEED |
| Export | handle, ACL | handle, ACL | handle, ACL | If the unit is operating in FIPS level 2 mode this operation is only available for public keys – see Operating a level 2 module in FIPS mode on page 25. | *Exports* a public key object. | None. | None. |

| Command / Service | Role | | | Description | Key/CS P access | Approved | |
|---|---|---|---|---|---|---|---|
| | Unauth | User | Admin | | | | Non-approved |
| Feature Enable | No | No | Yes | Enables a service. This requires a certificate signed by the Master Feature Enable key. | *Uses* the public half of the Master Feature Enable Key | Signature verification: DSA 1024-bit | None. |
| Firmware Authenticate | Yes | Yes | Yes | Reports firmware version. Performs a zero knowledge challenge response protocol based on HMAC that enables a operator to ensure that the firmware in the module matches the firmware supplied by nCipher.<br>The protocol generates a random value to use as the HMAC key. | No access to keys or CSPs | N/A | HMAC with Tiger hash. |
| Generate Key | Yes | Yes | Yes | Generates a symmetric key of a given type with a specified ACL and returns a handle.<br>Optionally returns a certificate containing the ACL. | *Creates* a new symmetric key object.<br>*Sets* the ACL and Application data for that object. Optionally *uses* module signing key and *exports* the key generation certificate. | Key gen using DRBG: AES all key sizes; Triple-DES 3-Key | Key gen using DRBG: Triple-DES 2-Key, DES, ArcFour, Aria, Camellia, SEED |

| Command / Service | Role | | | Description | Key/CS P access | Approved | |
|---|---|---|---|---|---|---|---|
| | Unauth | User | Admin | | | | Non-approved |
| Generate Key Pair | Yes | Yes | Yes | Generates a key pair of a given type with specified ACLs for each half or the pair. Performs a pair wise consistency check on the key pair.<br>Returns two key handles.<br>Optionally returns certificates containing the ACL. | *Creates* two new key objects. *Sets* the ACL and Application data for those objects. Optionally *uses* module signing key and *exports* two key generation certificates | Key gen using DRBG: DSA and ECDSA all approved key sizes | Key gen using DRBG: RSA all key sizes; DSA and ECDSA with non-compliant key sizes; KCDSA all key sizes. |
| Get ACL | handle, ACL | handle, ACL | handle, ACL | Returns the ACL for a given handle. | *Exports* the ACL for a key object. | None | None |
| Get Application Data | handle, ACL | handle, ACL | handle, ACL | Returns the application information stored with akey. | *Exports* the application data of a key object. | None | None |
| Get Challenge | Yes | Yes | Yes | Returns a random nonce that can be used in certificates | No access to keys or CSPs | DRBG | None |
| Get Key Info | handle | handle | handle | Superseded by **GetKeyInfoExtended**, retained for compatibility. | *Exports* the SHA- 1 hash of a key object | Hash: SHA-1 | None |
| Get Key Info Extended | handle | handle | handle | Returns the hash of a key for use in ACLs | *Exports* the SHA- 1 hash of a key object | Hash: SHA-1 | None |
| Get Module Keys | Yes | Yes | Yes | Returns a hashes of all loaded module keys. | *Exports* the SHA- 1 hash of module keys. | Hash: SHA-1 | None |

| Command / Service | Role | | | Description | Key/CS P access | Approved | |
|---|---|---|---|---|---|---|---|
| | Unauth | User | Admin | | | | Non-approved |
| Get Module Long Term Key | Yes | Yes | Yes | Returns a handle to the public half of the module's signing key. this can be used to verify key generation certificates and to authenticate inter module paths. | *Exports* the public half of the module's long term signing key. | None | None |
| Get Module Signing Key | Yes | Yes | Yes | Returns the public half of the module's signing key. This can be used to verify certificates signed with this key. | *Exports* the public half of the module's signing key. | None | None |
| Get Slot List | Yes | Yes | Yes | Reports the list of slots available from this module. | No access to keys or CSPs | None | None |
| Get Ticket | handle | handle | handle | Gets a ticket - an invariant identifier - for a key. This can be passed to another client which can redeem it using **RedeemTicket** to obtain a new handle to the object, | *Uses* a key object. | None | None |
| Hash | Yes | Yes | Yes | Hashes a value. | No access to keys or CSPs | SHA-1, SHA-256, SHA-384, SHA-512 | HAS-160, MD5, RIPEMD-160, Tiger |
| Import | Yes | Yes | Yes | Loads a key and ACL from the host and returns a handle.<br>If the unit is operating in FIPS mode at level 2, this operation must only be used for public keys - see Operating a level 2 module in FIPS mode on page 25. | *Creates* a new key object to store imported key, *sets* the key value, ACL and App data. | None | None |

| Command / Service | Role | | | Description | Key/CS P access | Approved | |
|---|---|---|---|---|---|---|---|
| | Unauth | User | Admin | | | | Non-approved |
| Initialise Unit | init | init | init | Initialises the module, returning it to the factory state. This clears all loaded keys and all module keys and the module signing key. This can only be performed when the module is in initialisation mode, using the Mode selector.<br><br>It also generates a new KM0 and module signing key.<br><br>The only key that is not zeroized is the long term signing key. This key only serves to provide a cryptographic identity for a module that can be included in a PKI certificate chain. nCipher may issue such certificates to indicate that a module is a genuine nShield module. This key is not used to encrypt any other data.<br><br>If the Module signing key is believed to have been compromised, the module must be re-initialised via this service. | *Erases* all keys, *Creates* KM0 and KML | Key gen using DRBG: AES 256-bit and DSA 3072-bit | None |
| Load Blob | No | handle | handle | Loads a key that has been stored in a key blob. The operator must first have loaded the key used to encrypt the blob. | *Uses* module key, or archiving key, *creates* a new key object. | Decryption: AES 256-bit<br><br>Message auth: HMAC-SHA-1 | Key establishment: RSA, DLIES |

| Command / Service | Role | | | Description | Key/CS P access | Approved | |
|---|---|---|---|---|---|---|---|
| | Unauth | User | Admin | | | | Non-approved |
| Load Buffer | No | handle | handle | Loads signed data into a buffer.<br><br>Several load buffer commands may be required to load all the data, in which case it is the responsibility of the client program to ensure they are supplied in the correct order.<br><br>Requires the handle of a buffer created by **CreateBuffer**. | No access to keys or CSPs | None | None |
| Make Blob | No | handle, ACL | handle, ACL | Creates a key blob containing the key and returns it. The key object to be exported may be any algorithm. | *Uses* module key, archiving key, *exports* encrypted key object. | Encryption:AES 256-bit<br><br>Message auth: HMAC-SHA-1 | Key establishment: RSA, DLIES |
| Mod Exp | Yes | Yes | Yes | Performs a modular exponentiation on values supplied with the command. | No access to keys or CSPs | None | None |
| Mod Exp CRT | Yes | Yes | Yes | Performs a CRT modular exponentiation on values supplied with the command. | No access to keys or CSPs | None | None |
| Module Info | Yes | Yes | Yes | Reports low level status information about the module. This service is designed for use in nCipher's test routines. | No access to keys or CSPs | None | None |
| New Enquiry | Yes | Yes | Yes | Reports status information (Show Status). | No access to keys or CSPs | None | None |

| Command / Service | Role | | | Description | Key/CS P access | Approved | |
|---|---|---|---|---|---|---|---|
| | Unauth | User | Admin | | | | Non-approved |
| No Operation | Yes | Yes | Yes | Does nothing, can be used to determine that the module is responding to commands. | No access to keys or CSPs | None | None |
| Random number | Yes | Yes | Yes | Generates a random number for use in a application using the on-board random number generator.<br>There are separate services for generating keys.<br>The random number services are designed to enable an application to access the random number source for its own purposes - for example an on-line casino may use **GenerateRandom** to drive its applications. | *Uses*<br>DRBG<br>Key | DRBG | None |
| Random prime | Yes | Yes | Yes | Generates a random prime. This uses the same mechanism as is used for RSA and DH key generation. The primality checking conforms to ANSI X9.31. | *Uses* DRBG key | DRBG | None |
| Redeem Ticket | ticket | ticket | ticket | Gets a handle in the current name space for the object referred to by a ticket created by **GetTicket**. | *Uses* a key object | None | None |
| Set ACL | handle, ACL | handle, ACL | handle, ACL | Sets the ACL for an existing key. The existing ACL for the key must allow the operation. | *Sets* the Access Control List for a key object | None | None |

| Command / Service | Role | | | Description | Key/CS P access | Approved | Non-approved |
|---|---|---|---|---|---|---|---|
| | Unauth | User | Admin | | | | |
| Set Application Data | handle, ACL | handle, ACL | handle, ACL | Stores information with a key. | *Sets* the application data stored with a key object | None | None |
| Sign | handle, ACL | handle, ACL | handle, ACL | Returns the digital signature or MAC of plain text using a stored key. | *Uses* a key object | Signature: DSA and ECDSA with compliant key sizes.<br>Message auth: AES CMAC and AES GMAC all key sizes; Triple- DES 3-Key MAC;<br>HMAC with SHA1, SHA-256, SHA-384, SHA-512 | Signature: RSA all key sizes; DSA and ECDSA with non-compliant key sizes; KCDSA.<br>Message auth: Triple-DES 2-Key; HMAC with HAS-160, MD5, RIPEMD160, Tiger |
| Sign Module State | handle, ACL | handle, ACL | handle, ACL | Signs a certificate describing the modules security policy. | *Uses* the module signing key | Signature: DSA 3072-bit | None |
| Statistic Get Value | Yes | Yes | Yes | Reports a particular statistic. | No access to keys or CSPs | None | None |
| Statistics Enumerate Tree | Yes | Yes | Yes | Reports the statistics available. | No access to keys or CSPs | None | None |
| Verify | handle, ACL | handle, ACL | handle, ACL | Verifies a digital signature or MAC using a stored key. The key used is a key that is stored in the module. | *Uses* a key object. | Signature: DSA and ECDSA all key sizes.<br>Message auth: AES CMAC and AES GMAC all key sizes; Triple-DES MAC all key sizes; HMAC with SHA1, SHA-256, SHA-384, SHA-512 | Signature: RSA and KCDSA all key sizes.<br>Message auth: HMAC with HAS-160, MD5, RIPEMD160, Tiger |

## 4.1 Terminology

| Code | Description |
|---|---|
| No | The operator can not perform this service in this role. |
| yes | The operator can perform this service in this role without further authorization. |
| handle | The operator can perform this service if they possess a valid handle for the resource: key, channel, buffers.<br>The handle is an arbitrary number generated when the object is created. The handle for an object is specific to the operator that created the object. The ticket services enable an operator to pass an ID for an object they have created to another operator. |
| ACL | Access Control List. The operator can only perform this service with a key if the ACL for the key permits this service. The ACL may require that the operator present a certificate signed by a Security Officer or another key.<br>The ACL may specify that a certificate is required, in which case the module verifies the signature on the certificate before permitting the operation. |
| FE | Feature Enable. This service is not available on all modules. It must be enabled using the **FeatureEnable** service before it can be used. |
| ticket | The **RedeemTicket** command requires the ticket generated by **GetTicket**. |

# 5 Keys

For each type of key used by the nToken, the following section describes the access that an operator has to the keys.

nShield modules refer to keys by their handle, an arbitrary number, or by its SHA-1 hash.

## 5.1 Long term signing key

The nToken stores one 160-bit and one 256-bit random number in the EEPROM .

The 160-bit number is combined with a discrete log group stored in the module firmware to produce a DSA key. The 256-bit number is used as the private exponent of a ECDSA key using the NIST P521 curve.

It can be used to sign a module state certificate using the **SignModuleState** service and the public value retrieved by the non-cryptographic service **GetLongTermKey**.

This is the only key that is not zeroized when the module is initialised.

This key is not used to encrypt any other data. It only serves to provide a cryptographic identity for a module that can be included in a PKI certificate chain. nCipher may issue such certificates to indicate that a module is a genuine nCipher module.

## 5.2 Module signing key

When the nShield module is initialised it automatically generates a 3072-bit DSA key pair that it uses to sign certificates. Signatures with this key use SHA-256. The private half of this pair is stored internally in EEPROM and never released. The public half is revealed in plaintext, or encrypted as a key blob. This key is only ever used to verify that a certificate was generated by a specified module.

## 5.3 Module Keys

Module Keys are AES 256 bit or 3-Key Triple DES . The module generates the first Module Key $K_{M0}$ when it is initialised. This Module Key is guaranteed never to have been known outside this M0 module.

Module Keys can not be exported once they have been assigned as Module Keys. They may only be exported on a key blob when they are initially generated.

## 5.4 Key objects

Keys used for encryption, decryption, signature verification and digital signatures are stored in the module as objects in the object store in RAM. All key objects are identified by a random identifier that is specific to the operator and session.

All key objects are stored with an Access control List or ACL. The ACL specifies what operations can be performed with this key. Whenever an operator generates a key or imports a key in plain text they must specify a valid ACL for that key type. The ACL can be changed using the **SetACL** service. The ACL can only be made more permissive if the original ACL includes the **ExpandACL** permission.

Key objects may be exported as key blobs if their ACL permits this service. Each blob stores a single key and an ACL. The ACL specifies what operations can be performed with this copy of the key. The ACL stored with the blob must be at least as restrictive as the ACL associated with the key object from which the blob was created. When you load a key blob, the new key object takes its ACL from the key blob. Working key blobs are encrypted under the Module Key. Key objects may also be exported as key blobs under an archiving key. The key blob can be stored on the host disk.

Key objects can only be exported in plain text if their ACL permits this operation. An operator may pass a key reference to another operator using the ticketing mechanism. The **GetTicket** mechanism takes a key identifier and returns a ticket. This ticket refers to the key identifier - it does not include any key data. A ticket can be passed as a byte block to the other operator who can then use the **RedeemTicket** service to obtain a key identifier for the same object that is valid for their session. As the new identifier refers to the same object the second operator is still bound by the original ACL.

When the nToken is enrolled it generates a DSA key pair used for signature generation known as the Authentication key. This is a key object as described above.

## 5.5 Archiving keys

It is sometimes necessary to create an archive copy of a key, protected by another key. Keys may be archived using the following mechanisms.

Note: Established keys may not be used in an Approved mode if the keys were established with the help of a key establishment algorithm using a non-approved algorithm or a non-compliant key size. See Services available to each role on page 10, and see Supported Algorithms on page 31.

- Three-key Triple DES or AES keys (used for unwrapping legacy keys and wrapping public keys only).
- A combination of three-key Triple DES and RSA keys (non-approved).
- A key encapsulation mechanism using RSA (non-approved).
- A hybrid key encapsulation mechanism using Diffie Hellman (DLIES, non-approved).

When a key is archived in this way it is stored with its ACL.

When you generate or import the archiving key, you must specify the **UseAsBlobKey** option in the ACL. The archiving key is treated as any other key object.

When you generate or import the key that you want to archive you must specify the Archival options in the ACL. This options can specify the hash(es) of the allowed archiving key(s). If you specify a list of hashes, no other key may be used.

## 5.6 Firmware Integrity Key

All firmware is signed using a 3072-bit DSA key pair. Signatures with this key use SHA-256.

The module checks the signature before new firmware is written to flash. A module only installs new firmware if the signature decrypts and verifies correctly.

The private half of this key is stored at nCipher.

The public half is included in all firmware. The firmware is stored in flash memory when the module is switched off, this is copied to RAM as part of the start up procedure.

## 5.7 Firmware Confidentiality Key

All firmware is encrypted using AES to prevent casual decompilation. The encryption key is stored at nCipher's offices and is in the firmware.

The firmware is stored in flash memory when the module is switched off, this is copied to RAM as part of the startup procedure.

## 5.8 Master Feature Enable Key

For commercial reasons not all devices in the nShield family of HSMs offer all services. Certain services must be enabled separately. The nToken in particular is a very restricted member of the nShield family. In order to enable a service the operator presents a certificate signed by the Master Feature Enable Key.

The Master Feature Enable Key is a DSA key pair. The private half of this key pair is stored at nCipher's offices. The public half of the key pair is included in the firmware. The firmware is stored in flash memory when the module is switched off, this is copied to RAM as part of the start up procedure.

## 5.9 DRBG Key

DBRG stands for Deterministic Random Bit Generator.

The module uses the CTR_DRBG from SP800-90 with a 256-bit AES key. This key is seeded from the on board entropy source whenever the module is initialised and is reseeded according to SP800-90 with a further 1024 bits of entropy after every 2048-bytes of output.

This key is only ever used by the DRBG. It is never exposed outside the module.

The DRBG internal state is contained within the DRBG mechanism boundary and is not accessed by non-DRBG functions or by other instances of any DRBG.

**Note:** For CTR DRBG, the values of V and Key (SP 800-90) are the 'secret values' of the internal state.

# 6 Rules

## 6.1 Identification and authentication

Communication with the module is performed via a server program running on the host machine, using standard inter process communication, using sockets in UNIX operating systems, named pipes under Windows.

In order to use the module the operator must first log on to the host computer and start an nShield enabled application. The application connects to the hardserver, this connection is given a client ID, a 32-bit arbitrary number.

The User role is authenticated by presenting a valid key blob to the module.

The Administrator role is authenticated by presenting a signed Feature Enable certificate to the module.

Before performing any service the operator must present the correct authorization. Where several stages are required to assemble the authorization, all the steps must be performed on the same connection. The authorization required for each service is listed in the section Services available to each role on page 10.

### 6.1.1 Access Control

Module keys are stored in EEPROM in the module.

The key blob also contains an Access Control List that specifies which services can be performed with this key, and the number of times these services can be performed. These can be hard limits or per authorization limits. If a hard limit is reached that service can no longer be performed on that key.

All objects are referred to by handle. Handles are cross-referenced to ClientIDs. If a command refers to a handle that was issued to a different client, the command is refused. Services exist to pass a handle between **ClientID**s.

### 6.1.2 Access Control List

All key objects have an Access Control List (ACL). The operator must specify the ACL when they generate or import the key. The ACL lists every operation that can be performed with the key - if the operation is not in the ACL the module will not permit that operation. In particular the ACL can only be altered if the ACL includes the **SetACL** service. The ACL is stored with the key when it is stored as a blob and applies to the new key object created when you reload the blob.

The ACL can specify limits on operations - or groups of operations - these may be global limits or per authorization limits. If a global limit is exceeded then the key cannot be used for that operation again.

An ACL can also specify a certifier for an operation. In this case the operator must present a certificate signed by the key whose hash is in the ACL with the command in order to use the service.

An ACL can also specify a host service identifier. In which case the ACL is only met if the hardserver appends the matching Service name. This feature is designed to provide a limited level of assurance and relies on the integrity of the host, it offers no security if the server is compromised or not used.

ACL design is complex - operators will not normally need to write ACLs themselves. nCipher provide tools to generate keys with strong ACLs.

### 6.1.3 Object re-use

All objects stored in the module are referred to by a handle. The module's memory management functions ensure that a specific memory location can only be allocated to a single handle. The handle also identifies the object type, and all of the modules enforce strict type checking. When a handle is released the memory allocated to it is actively zeroed.

### 6.1.4 Error conditions

If the module cannot complete a command due to a temporary condition, the module returns a command block with no data and with the status word set to the appropriate value. The operator can resubmit the command at a later time. The server program can record a log of all such failures.

If the module encounters an unrecoverable error it enters the error state. This is indicated by the status LED flashing in the Morse pattern SOS. As soon as the unit enters the error state all processors stop processing commands and no further replies are returned. In the error state the unit does not respond to commands. Recorded error status codes may be queried without interaction with the module. The unit must be reset.

### 6.1.5 Security Boundary

The cryptographic boundary is the PCIecard.

### 6.1.6 Status information

The module has a status LED that indicates the overall state of the module.

The module also returns a status message in the reply to every command. This indicates the status of that command.

There are a number of services that report status information.

## 6.2 Operating a level 2 module in FIPS mode

To be operating in Level 2 FIPS Mode, only FIPS Approved cryptography can be used in FIPS Mode. Use of any Non-FIPS Approved algorithms, except for those for which the CMVP allowed in FIPS Mode (See Supported Algorithms Section), means that the module would not be operating in FIPS Mode.

In order to comply with FIPS mode the operator must not generate private or secret keys with the **ExportAsPlain** ACL entry; nor should they use the **Import** service to import such keys in plain text.

An operator can verify that a key was generated correctly using the **nfkmverify** utility supplied by nCipher. This utility checks the ACL stored in the key-generation certificate.

# 7 Physical security

All security critical components of the module are covered by epoxy resin.

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

## 7.1 Checking the module

To ensure physical security, make the following checks regularly:

- Examine the epoxy resin security coating for obvious signs of damage.

# 8 Strength of functions

## 8.1 Object IDs

The following analysis describes the strength of a session.

Connections are identified by a **ClientID**, a random 160 bit number.

Objects are identified by an **ObjectID** again this is a random 32 bit number.

In order to gain access to a key loaded by another operator a random 160 bit and a random 32 bit number would need to be guessed. There are $2^{192}$ possibilities therefore meets the 1 in a $10^6$ requirement.

The module can process about $2^{16}$ commands per minute - therefore the chance of succeeding within a minute is $2^{16} / 2^{192} = 2^{-176}$ which is significantly less that the required chance of 1 in $10^5$ (~$2^{-17}$)

## 8.2 Key Blobs

Key blobs are used to protect keys outside the module.

The blobs used for module key protected keys take a 256 bit AES key and a nonce and uses SHA-1 to derive a AES encryption key, used for encryption and a HMAC SHA-1 key, used for integrity. It provides a strength of 128 bits.

An attacker would be required to guess, at a minimum, the MAC value can have a key size of 160, 256 or 512 bits. At a minimum, this requires guessing a 160 bit HMAC key, which gives a probability of 2^-160. This probability is less than one in 1,000,000 that a random attempt will succeed and one in 100,000 that a random attempt will succeed or a false acceptance will occur during a one-minute period.

## 8.3 Feature Enable certificates

Feature enable certificates are signed using a 1024 bit DSA private key under the control of nCipher. The module performs a digital signature verification of the certificate, which provides a strength of 80 bits. Considering this strength, the probability is 2^-80, which is less than one in 1,000,000 that a random attempt will succeed and one in 100,000 that a random attempt will succeed or a false acceptance will occur during a one-minute period.

## 8.4 Firmware Images

Firmware images are signed using 3072-bit DSA with SHA-256, providing at least a 128-bit symmetric equivalent security level.

## 8.5 Impath authentication

The nToken module is used to authenticate the host computer by signing a nonce with a DSA 3072 or 1024 bit key . In the approved mode of operation, 1024-bit key size is not allowed.

## 8.6 Derived Keys

The nCore API provides a range of key derivation and wrapping options that an operator can choose to make use of in their protocols.

For any key, these mechanisms are only available if the operator explicitly enabled them in the key's ACL when they generated or imported the key.

The ACL can specify not only the mechanism to use but also the specific keys that may be used if these are known.

| Mechanism | Use | Notes |
|---|---|---|
| Key Splitting | Splits a symmetric key into separate components for export | Components are raw byte blocks. |
| PKCS8 wrapping | Encrypts a private key encoded inPKCS8 format using any supported symmetric algorithm. | Non-approved mode. |
| PKCS8 unwrapping | Decrypts a private key encoded inPKCS8 format using any supported symmetric algorithm. | Non-approved mode. |
| SSL/TLS master key derivation | Setting up a SSL/TLS session | Non-compliant. Theprotocols SSL, TLS shall not be used when operated in FIPS mode. In particular, none of the keys derived using this key derivation function can be used in the Approved mode. |
| Key Wrapping | Encrypts one key object with another to allow thewrapped key to beexported. | May use any supported encryption mechanism that accepts a byteblock.<br>The operator must ensure<br>that they chose a wrapping key that has an equivalent strength to the key being transported.<br>The operator must ensure<br>that they chose a wrapping key that has an equivalent strength to the key being transported. |

In Approved mode you can use key wrapping and key establishment mechanisms with all supported algorithms.

# 9 Self Tests

When power is applied to the module it enters the self test state. The module also enters the self test state whenever the unit is reset, by pressing the clear button or by sending the Clear Unit command.

In the self test state the module clears the main RAM, thus ensuring any loaded keys or authorization information is removed and then performs its self test sequence, which includes:

- An operational test on hardware components
- An integrity check on the firmware, verification of a SHA-1 hash
- Power-up self-test for the SP 800-90A DRBG
- Statistical tests on the entropy source (monobit, poker, runs and long runs tests)
- Known answer checks as required by FIPS 140-2:
    - SHA-1, SHA-1 HMAC, SHA-224 HMAC, SHA-256 HMAC, SHA-384 HMAC, SHA-512 HMAC,
    - DES, Triple-DES, AES,
    - AES CMAC, AES CBC MAC, Triple-DES CBC MAC,
    - DSA, RSA,
    - AES CTR based DRBG,
    - Primitive Z for DH and ECDH.
- ECDSA pair-wise consistency test
- Verification of a MAC on EEPROM contents to ensure it is correctly initialised.

The module also runs continuous random number generator tests on the NDRNG and the approved SP800-90A DRBG. If either fail, it enters the error state.

When firmware is updated, the module verifies a DSA signature on the new firmware image before it is written to flash.

The module also performs pairwise-consistency checks when generating asymmetric key-pairs.

In the error state, the module's LED repeatedly flashes the Morse pattern SOS, followed by a status code indicating the error. All other inputs and outputs are disabled.

Note that if the module's firmware is updated to a different version, this results in the loss of the current CMVP validation of the module.

# 10 Supported Algorithms

## 10.1 FIPS approved and allowed algorithms:

This section describes the Approved or allowed cryptographic algorithms supported by the Cryptographic Module.

### 10.1.1 Symmetric Encryption

**AES**

Certificate #2122

Key sizes: 128, 192, 256 bits.

ECB, CBC and GCM (internally generated IV).

**Triple-DES**

Certificate #1349

ECB and CBC modes.

Key sizes: 2-Key (112 bits, decryption only), 3-Key (168 bits).

After 2^28 encryption operations, the Triple-DES key must not be used for any further encryption operations.

### 10.1.2 Hashing and Message Authentication

**AES CMAC**

AES Certificate #2122

Key sizes: 128, 192, 256 bits.

**AES GMAC**

AES Certificate #2122

Key sizes: 128, 192, 256 bits.

**HMAC SHA-1, HMAC SHA-224, HMAC SHA-256, HMAC SHA-384 and HMAC SHA-512**

Certificate #1292

Key sizes greater or equal than 112 bits

**SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512**

Certificate #1844

**Triple-DES MAC**

Triple-DES Certificate #1349 vendor affirmed.

Key sizes: 2-Key (112 bits, MAC verification only), 3-Key (168 bits).

After 2^28 encryption operations, the Triple-DES key must not be used for any further encryption operations.

### 10.1.3 Signature

**DSA**

Certificate #664

FIPS 186-4: Signature generation and verification

Modulus 2048-bits Sub-group 224-bits SHA-224

Modulus 2048-bits Sub-group 256-bits SHA-256

Modulus 3072-bits Sub-group 256-bits SHA-256

**ECDSA**

Certificate #318

FIPS 186-4: Signature generation and verification.

The following curves are supported: P-224, P-256, P-384, P-521, K-233, K-283, K-409, K-571, B- 233, B-283, B-409 and B-571.

### 10.1.4 Key Establishment

**Diffie-Hellman**

Diffie-Hellman (CVL Cert. #27, key agreement; key establishment methodology provides 112 or 128 bits of encryption strength; non-compliant less than 112 bits of encryption strength)

Modulus 2048-bits, Sub-group 224-bits

Modulus 2048-bits, Sub-group 256-bits

Modulus 3072-bits, Sub-group 256-bits

**Elliptic Curve Diffie-Hellman**

EC Diffie-Hellman (CVL Cert. #27; key establishment methodology provides 112, 128, 192 or 256 bits of encryption strength; non-compliant less than 112 bits of encryption strength)

The following curves are supported: P-224, P-256, P-384, P-521, K-233, K-283, K-409, K-571, B- 233, B-283, B-409 and B-571.

**AES**

AES (Cert. #2122; key establishment methodology provides between 128 and 256 bits of encryption strength)

Key sizes: 128, 192, 256 bits.

CBC mode.

**Triple-DES**

Triple-DES (Cert. #1349, key wrapping; key establishment methodology provides 112 bits of encryption strength; non-compliant less than 112 bits of encryption strength).

CBC mode.

Key sizes: 2-Key (112 bits, key unwrap only), 3-Key (168 bits).

After 2^28 encryption operations, the Triple-DES key must not be used for any further encryption operations

**EC-MQV**

ECMQV (key agreement; key establishment methodology provides 112, 128, 192 or 256 bits of encryption strength; non-compliant less than 112 bits of encryption strength)

The following curves are supported: P-224, P-256, P-384, P-521, K-233, K-283, K-409, K-571, B- 233, B-283, B-409 and B-571.

### 10.1.5 Other

**Deterministic Random Bit Generator**

Certificate #232

SP 800-90A compliant using AES-CTR with AES-256 SP800-133 CKG, vendor affirmed

**Non-deterministic entropy source**

Allowed non-deterministic entropy source, used to seed approved random bit generator.

## 10.2 Non-FIPS Approved Algorithms

This section describes the non-approved or non-compliant algorithms supported by the Cryptographic Module. If used, the module will not be operating in the approved mode of operation.

Keys that have been established with the help of a key establishment algorithm of a non- compliant strength are also non-compliant.

### 10.2.1 Symmetric

- Aria
- Arc Four (compatible with RC4)
- Camellia
- CAST 6 (RFC2612)
- DES
- 2-Key Triple-DES encryption, MAC generation (112 bit key size)
- SEED (Korean Data Encryption Standard) - requires Feature Enable activation
- SP800-38F, AES KW (non-compliant)
- AES CMAC (non-compliant)
- AES GCM with externally supplied IV
- SP800-108 KDF with AES CMAC in Counter Mode KDF (non-compliant)

### 10.2.2 Asymmetric

- El Gamal (encryption using Diffie-Hellman keys)
- KCDSA (Korean Certificate-based Digital Signature Algorithm) - requires Feature Enable activation
- RSA encryption, decryption, sign and key encapsulation (modulus 1024 supported)
- 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 or B-163
- ECDH with curves P-192, K-163 or B-163
- ECMQV with curves P-192, K-163 or B-163
- DH with key size p < 2048 bits or q < 224 bits
- DLIES

### 10.2.3 Hashing and Message Authentication

- HAS-160 - requires Feature Enable activation
- MD5 - requires Feature Enable activation
- RIPEMD 160
- Tiger
- HMAC (MD5, RIPEMD160, Tiger) or key size less than 112 bits

### 10.2.4 Other

- SSL/TLS master key derivation
- PKCS#8

# Contact Us

| | |
|---|---|
| Web site: | https://www.ncipher.com |
| Help Centre: | https://help.ncipher.com |
| Email Support: | support@ncipher.com |
| Contact Support Numbers: | https://www.ncipher.com/services/support/contact-support |

www.ncipher.com

## About nCipher Security

nCipher Security, an Entrust Datacard company, is a leader in the general-purpose hardware security module (HSM) market, empowering world-leading organizations by delivering trust, integrity and control to their business critical information and applications. Today's fast-moving digital environment enhances customer satisfaction, gives competitive advantage and improves operational efficiency – it also multiplies the security risks. Our cryptographic solutions secure emerging technologies such as cloud, IoT, blockchain, and digital payments and help meet new compliance mandates. We do this using our same proven technology that global organizations depend on today to protect against threats to their sensitive data, network communications and enterprise infrastructure. We deliver trust for your business critical applications, ensure the integrity of your data and put you in complete control – today, tomorrow, always. www.ncipher.com

Search: nCipherSecurity

TRUST. INTEGRITY. CONTROL.