# The nForce modules security policy

## nForce 150 PCI, nForce 300 PCI

**Date:** **14 February 2003**

**Version:** **1.2.27**

© Copyright 2003 nCipher Corporation Limited, Cambridge, United Kingdom.

Reproduction is authorised provided the document is copied in its entirety without modification and including this copyright notice.

nCipher™, nForce™, nShield™, nCore™, KeySafe™, CipherTools™, CodeSafe™, SEE™ and the SEE logo are trademarks of nCipher Corporation Limited.

nFast® and the nCipher logo are registered trademarks of nCipher Corporation Limited.
All other trademarks are the property of the respective trademark holders.

nCipher Corporation Limited makes no warranty of any kind with regard to this information, including, but not limited to, the implied warranties of merchantability and fitness to a particular purpose. nCipher Corporation Limited shall not be liable for errors contained herein or for incidental or consequential damages concerned with the furnishing, performance or use of this material.

**Patents**

UK Patent GB9714757.3. Corresponding patents/applications in USA, Canada, South Africa, Japan and International Patent Application PCT/GB98/00142.

## Contents


Chapter 1: Purpose 1

Roles 3

Services available to each role 4

Keys 15

Rules 20

Physical security 25

Strength of functions 26

Algorithms 27

# Chapter 1: Purpose

The nForce secure e-Commerce accelerators are multi-tasking hardware modules that are optimized for performing modular arithmetic on very large integers. The nForce 150 PCI, nForce 300 PCI also offer a complete set of key management protocols.

| Unit ID | Model Number | Format | RTC NVRAM | SEE | Potting | EMC Class | FIPS 140-2 overall level |
|---|---|---|---|---|---|---|---|
| nForce PCI 00 | nC3022-00 | embedded | No | Disabled | Yes | A | 2 |
| nForce PCI 150 | nC3022-150 | embedded | No | Disabled | Yes | A | 2 |

The nForce modules: nForce 150 PCI, nForce 300 PCI are FIPS 140-2 level 2 embedded devices. The units are identical in operation and only vary in the processing speed.

The module runs firmware provided by nCipher. There is the facility for the user to upgrade this firmware. In order to determine that the module is running the correct version of firmware they should use the NewEnquiry service which reports the version of firmware currently loaded. This should be 2.0.0 and 2.0.2.

The module can be initialized to comply with the roles and services section at either level 2, role based authorization, or level 3, identity based authorization. See “To comply with FIPS 140-2 level 3 roles and services and key management” on page 21. The initialization parameters are reported by the NewEnquiry and SignModuleState services. A user can determine which mode the module is operating in using the KeySafe GUI or the command line utilities supplied with the module, or their own code - these operate outside the security boundary.

The module offers some general services that do not involve stored keys and are therefore excluded from FIPS 140-2. While the module is executing these services it is operating in a non-FIPS 140-2 mode. When it is executing any of the FIPS 140-2 services using one of the FIPS approved algorithms, the module is operation in a FIPS 140-2 approved mode, See “Cryptographic services” on page 5.

The nForce modules connects to the host computer via a PCI bus. The nForce modules must be accessed by a custom written application. Full documentation for the nCipher API can be downloaded from the nCipher web site: http://www.ncipher.com.

The nForce modules stores keys on the hard disk of the host computer in encrypted form called key blobs.

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

- Windows NT 4.0 for Intel
- Windows 2000
- Solaris
- HP-UX
- AIX
- Linux x86
- FreeBSD x86

Windows NT was used for the validation.

# Roles

The nForce modules defines the following roles:

## User

All users initially connect to the nForce modules in the User role. In this role the user can load previously created tokens and use these to load keys from key blobs. Once they have loaded a key they can then use it to perform cryptographic operations as defined by the Access Control List (ACL) stored with the key.

Each key blob contains an ACL that determines what services can be performed on that key. This ACL can require a certificate from a Security Officer authorizing the action. Some actions including writing tokens always require a certificate.

## nCipher Security Officer

The nCipher Security Officer (NSO) is responsible for overall security of the module.

The nCipher Security Officer is identified by a key pair, referred to as $K_{NSO}$. The hash of the public half of this key is stored when the unit is initialized. Any operation involving a module key or writing a token requires a certificate signed by $K_{NSO}$.

The nCipher Security Officer is responsible for creating the authentication tokens (smart cards) for each user and ensuring that these tokens are physically handed to the correct person.

A user assumes the role of NSO by loading the private half of $K_{NSO}$ and presenting the KeyID for this key to authorize a command.

## Junior Security Officer

Where the nCipher Security Officer want to delegate responsibility for authorizing an action they can create a key pair and give this to their delegate who becomes a Junior Security Officer (JSO). An ACL can then refer to this key, and the JSO is then empowered to sign the certificate authorizing the action. The JSO's keys should be stored on a key blob protected by a token that is not used for any other purpose.

In order to assume the role of JSO the user loads the JSO key and presents the KeyID of this key, and if required the certificate singed by $K_{NSO}$ that delegates authority to the key, to authorize a command.

A JSO can delegate portions of their authority to a new user in the same way. The new user will be a JSO if they have authority they can delegate, otherwise they will be a user.

# Services available to each role

For more information on each of these services refer to the nCipher Developer's Guide and nCipher Developer's Reference.

## Non Cryptographic services

### Status information

The status of the module is also indicated by the LED on the rear panel.

### Self Test

The module self tests be initiated by clearing the module: either by pressing the Clear button on the rear panel, or by submitting the ClearUnit command.

### Module state

The following services change the module state.

- Fail
  A test service that causes the module to fail.
- Clear unit
  Clears the module - equivalent to pressing the clear button.

*Note* *The nCipher server will only issue these commands if the user is connected with Administrator privileges.*

### Firmware Upgrade

It is possible to upgrade the firmware in the nForce modules. The module will only accept new firmware that has been signed by nCipher, using the Firmware Integrity Key and encrypted by the Firmware Confidentiality Keys. The monitor also checks that the VSN of the firmware is as high or higher than the VSN of the firmware currently installed.

*Note* *The module will only be operating in a FIPS approved mode if you install firmware that has been validated by NIST/CSE. Users who require FIPS validation should only upgrade firmware after NIST/CSE issue a new certificate.*

The module must be in the Pre-Monitor state. The user uses the LoadROM utility which issues the correct commands to load the new firmware.

The following commands/services are only ever issued by the LoadROM utility:

- Maintenence
- ProgrammingBegin
- ProgrammingBeginChunk
- ProgrammingLoadBlock
- ProgrammingEndChunk
- ProgrammingEnd

## Cryptographic services

The following services provide user authentication or cryptographic functionality. The functions available depend on whether the user is in the unauthorized role, an authorized user including junior security officer (JSO), or the nCipher Security Officer (NSO):

For each operation it lists the supported algorithms:

| Command/ Service | Role | | | Description<br>*Key/CSP access*<br>Key types |
|---|---|---|---|---|
| | Unuath | User / JSO | NSO | |
| Bignum Operation | yes | Yes | Yes | Performs simple mathematical operations.<br>*No access to keys or CSPs* |
| Change Share PIN | - | pass phrase | pass phrase | Updates the pass phrase used to encrypt a token share. The pass phrase supplied by the user is not used directly, it is first hashed and then combined with the module key. To achieve this the command decrypts the existing share using the old share key derived from old pass phrase, module key and smart card identity. It then derives a new share key based on new pass phrase, module key and smart card identity, erases old share from smart card and writes a new share encrypted under the new share key.<br>***Sets** the pass phrase for a share, **uses** module key, **uses** share key, **uses** module key, **creates** share key, **uses** new share key, **exports** encrypted share, **erases** old share* |
| Channel Open | - | handle, ACL | handle, ACL | Opens a communication channel which can be used for bulk encryption or decryption.<br>***Uses** a key object*<br>AES, DES, Triple DES, Arc Four, Blowfish, CAST 5, CAST 6, Serpent, SEED, Twofish |

| Command/ Service | Role | | | Description<br>*Key/CSP access*<br>Key types |
|---|---|---|---|---|
| | Unuath | User / JSO | NSO | |
| Channel Update | - | handle | handle | Performs encryption/decryption on a previously opened channel. The operation and key are specified in ChannelOpen<br>***Uses** a key object*<br>AES, DES, Triple DES, Arc Four, Blowfish, CAST 5, CAST 6, Serpent, SEED, Twofish |
| Decrypt | - | handle, ACL | handle, ACL | Decrypts a cipher text with a stored key returning the plain text.<br>***Uses** a key object*<br>AES, DES, Triple DES, Arc Four, Blowfish, CAST 5, CAST 6, Serpent, SEED, Twofish, Diffie-Helman, RSA, El Gamal, KCDSA |
| Derive Key | - | 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.<br>***Uses** a key object, **create** a new key object*<br>AES, DES, Triple DES, PKCS #8, SSL key derivation, XOR CAST 5, CAST 6, Blowfish, Arc Four, Serpent, SEED, Twofish |
| Destroy | - | handle | handle | Removes an object, if an object has multiple handles as a result of RedeemTicket service, this removes the current handle.<br>***Erases** a Impath, logical token, or any key object.* |
| Duplicate | - | handle, ACL | handle, ACL | Creates a second instance of a key object with the same ACL and returns a handle to the new instance.<br>***Creates** a new key object* |
| Encrypt | - | handle, ACL | handle, ACL | Encrypts a plain text with a stored key returning the cipher text.<br>***Uses** a key object*<br>AES, DES, Triple DES, Arc Four, Blowfish, CAST 5, CAST 6, Serpent, SEED, Twofish, RSA, El Gamal, KCDSA |
| EraseFile | level 2 only | cert | yes | Removes a file, but not a logical token, from a smart card or software token.<br>*No access to keys or CSPs* |
| Erase Share | level 2 only | cert | yes | Removes a share from a smart card or software token.<br>***Erases** a share* |
| Existing Client | yes | yes | yes | Starts a new connection as an existing client.<br>*No access to keys or CSPs* |

| Command/ Service | Role | | | Description<br>*Key/CSP access*<br>Key types |
|---|---|---|---|---|
| | Unuath | User / JSO | NSO | |
| Export | - | handle, ACL | handle, ACL | Exports a key in plain text. If the unit has been initialized to comply with FIPS 140-2 level 3 roles and services and key management, this service is only available for public keys.<br>***Exports*** *a [public] key object.*<br>Any key type |
| Feature Enable | - | cert | cert | Enables a service.<br>This requires a certificate signed by the nCipher Master Feature Enable key.<br>***Uses*** *the public half of the nCipher Master Feature Enable Key* |
| Firmware Authenticate | yes | yes | yes | Performs a zero knowledge challenge response protocol based on HMAC that enables a user 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.<br>*No access to keys or CSPs* |
| FormatToken | level 2 only | cert | yes | Formats a smart card or software token ready for use.<br>*May* ***use*** *a module key to create challenge response value* |
| Generate Key | level 2 only | cert | yes | Generates a symmetric key of a given type with a specified ACL and returns a handle. Optionally returns a certificate containing the ACL.<br>***Creates*** *a new symmetric key object.* ***Sets*** *the ACL and Application data for that object. Optionally* ***uses*** *module signing key and* ***exports*** *the key generation certificate*<br>AES, DES, Triple DES, Arc Four, Blowfish, CAST 5, CAST 6, Serpent, SEED, Twofish |
| Generate Key Pair | level 2 only | cert | 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. Returns two key handles. Optionally returns certificates containing the ACL.<br>***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.*<br>RSA, El Gamal, KCDSA, DSA |
| Generate KLF | - | FE | FE | Generates a new long term key.<br>***Erases*** *the module long term key,* ***creates*** *new module long term key* |

| Command/ Service | Role | | | Description<br>*Key/CSP access*<br>Key types |
|---|---|---|---|---|
| | Unuath | User / JSO | NSO | |
| Generate Logical Token | level 2 only | cert | yes | Creates a new logical token, which can then be written as shares to smart cards or software tokens<br>***Uses** module key, **Creates** a logical token* |
| Get ACL | - | handle, ACL | handle, ACL | Returns the ACL for a given handle.<br>***Exports** the ACL for a key object* |
| Get Application Data | - | handle, ACL | handle, ACL | Returns the application information stored with a key.<br>***Exports** the application data of a key object* |
| Get Challenge | yes | yes | yes | Returns a random nonce that can be used in certificates<br>*No access to keys or CSPs* |
| Get Key Info | - | handle | handle | Superseded by GetKeyInfoEx retained for compatability.<br>***Exports** the SHA-1 hash of a key object* |
| Get Key Info Extended | - | handle | handle | Returns the hash of a key for use in ACLs<br>***Exports** the SHA-1 hash of a key object* |
| Get Logical Token Info | - | handle | handle | Superseded by GetLogicalTokenInfoEx retained for compatability.<br>***Exports** the SHA-1 hash of a logical token* |
| Get Logical Token Info Extended | - | handle | handle | Returns the token hash and number of shares for a logical token<br>***Exports** the SHA-1 hash of a logical token* |
| 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.<br>***Exports** the public half of the module's signing key* |
| 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.<br>***Exports** the public half of the module's long term key* |
| Get Share ACL | yes | yes | yes | Returns the access control list for a share<br>***Exports** the ACL for a token share on a smart card* |
| GetSlot Info | yes | yes | yes | Returns status of the physical token in a slot. Enables a user to determine if the correct token is present before issuing a ReadShare command. If the token is formatted with a challenge response key, performs the protocol.<br>*May **use** a module key* |

| Command/ Service | Role | | | Description<br>*Key/CSP access*<br>Key types |
|---|---|---|---|---|
| | Unuath | User / JSO | NSO | |
| Get Slot List | yes | yes | yes | Returns the list of slots available from this module.<br>*No access to keys or CSPs* |
| GetTicket | - | 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 halndle to the object,<br>***Uses** a key object, logical token, Impath, SEEWorld* |
| Hash | yes | yes | yes | Hashes a value, see "Strength of functions" on page 25 for a list of supported algorithms.<br>*No access to keys or CSPs*<br>HSA-160, MD2, MD5, RIPEMD 160, SHA-1, SHA-256, SHA-384, SHA-512 |
| Impath Get Info | - | handle | handle | Returns status information about an impath<br>***Uses** an Impath, exports status information* |
| Impath Key Exchange Begin | yes | yes | yes | Creates a new inter-module path and returns the key exchange parameters to send to the peer module. This service is feature enabled.<br>***Creates** a set of Impath keys* |
| Impath Key Exchange Finish | - | handle | handle | Completes an impath key exchange. Require the key exchange parameters from the remote module.<br>***Creates** a set of Impath keys* |
| Impath Receive | - | handle | handle | Decrypts data with the Impath decryption key.<br>***Uses** an Impath key* |
| Impath Send | - | handle | handle | Encrypts data with the impath encryption key.<br>***Uses** an Impath key* |
| Import | level 2 only | cert | yes | Loads a key and ACL from the host and returns a handle. Imports a key in plain text. If the unit has been initialized to comply with FIPS 140-2 level 3 roles and services and key management, this service is only available for public keys.<br>***Creates** a new key object, sets the key value, ACL and App data.*<br>Any key type |
| Load Blob | - | handle | handle | Loads a key that has been stored in a key blob. The user must first have loaded the token or key used to encrypt the blob.<br>***Uses** module key, logical token, or archiving key, **creates** a new key object* |

| Command/ Service | Role | | | Description<br>*Key/CSP access*<br>Key types |
|---|---|---|---|---|
| | Unuath | User / JSO | NSO | |
| Load Logical Token | yes | yes | yes | Allocates space for a new logical token - the individual shares can then be assembled using ReadShare or ReceiveShare. Once assembeld the token can be used in LoadBlob or MakeBlob commands<br>***Uses*** *module key* |
| Make Blob | - | handle, ACL | handle, ACL | Creates a key blob containing the key and returns it. The key object to be exported may be any algorithm.<br>***Uses*** *module key, logical token or archiving key,* ***exports*** *encrypted key object* |
| Mod Exp | yes | yes | yes | Performs a modular expoentiation on values supplied with the command.<br>*No access to keys or CSPs* |
| Mod Exp CRT | yes | yes | yes | Performs a modular expoentiation on values, supplied with the command using Chinese Remainder Theorem.<br>*No access to keys or CSPs* |
| Module Info | yes | yes | yes | Returns low level status information about the module. This service is designed for use in nCipher's test routines.<br>*No access to keys or CSPs* |
| NewClient | yes | yes | yes | Returns a client id.<br>*No access to keys or CSPs* |
| New Enquiry | yes | yes | yes | Returns status information<br>*No access to keys or CSPs* |
| No Operation | yes | yes | yes | Does nothing, can be used to determine that the module is responding to commands.<br>*No access to keys or CSPs* |
| Pause For Notifications | yes | yes | yes | This service enables the module to send status information to the nCipher server. The command is automatically submitted by the server.<br>*No access to keys or CSPs* |

| Command/ Service | Role | | | Description<br>*Key/CSP access*<br>Key types |
|---|---|---|---|---|
| | Unuath | User / JSO | NSO | |
| Random Number | yes | yes | yes | Generates a random number for use in a application using the on-board random number generator.<br>*Note 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.<br>*No access to keys or CSPs* |
| Random Prime | yes | yes | yes | Generates a random prime. This uses the same mechanism as is used for DSA, RSA and Diffie-Helman key generation.<br>*No access to keys or CSPs* |
| Read File | - | cert | yes | Reads a file, but not a logical token, from a smart card or software token.<br>*No access to keys or CSPs* |
| Read Share | yes | yes | yes | Reads a share from a physical token. Once sufficient shares have been loaded recreates token- may require several ReadShare or ReceiveShare commands.<br>***Uses** pass phrase, module key, **creates** share key, **uses** share key, **creates** a logical token* |
| Receive Share | - | handle, encrypted share | handle, encrypted share | Takes a share encrypted with SendShare and a pass phrase and uses them to recreate the logical token. - may require several ReadShare or ReceiveShare commands<br>***Uses** an Impath key, **uses** pass phrase, module key, **creates** share key, **uses** share key, **creates** a logical token* |
| Redeem Ticket | ticket | ticket | ticket | Gets a handle in the current name space for the object referred to by a ticket created by GetTicket.<br>***Uses** a key object, logical token, Impath, or SEEWorld* |
| Remove KM | - | cert | yes | Removes a loaded module key.<br>***Erases** a module key* |
| Set ACL | - | handle, ACL | handle, ACL | Sets the ACL for an existing key. The existing ACL for the key must allow the operation.<br>***Sets** the Access Control List for a key object* |
| Set Application Data | - | handle, ACL | handle, ACL | Stores information with a key.<br>***Sets** the application data stored with a key object* |

| Command/ Service | Role | | | Description<br>*Key/CSP access*<br>Key types |
|---|---|---|---|---|
| | Unuath | User / JSO | NSO | |
| Set KM | - | cert | yes | Loads a Triple DES key object as a module key.<br>***Uses** a key object, sets a module key* |
| Set KNSO | init | init | - | Superseded by SetNSOPerm. Although this command can still be used it cannot authorize recently added services added.<br>***Sets** the nCipher Security officer's key hash* |
| Set NSO Perm | init | init | - | Loads a key hash as the nCipher Security Officer's Key and sets the security policy to be followed by the module. This can only be performed while the unit is in the initialisation state.<br>***Sets** the nCipher Security officer's key hash* |
| Sign | - | handle, ACL | handle, ACL | Returns the digital signature or MAC of plain text using a stored key. See "Strength of functions" on page 26 for a list of supported algorithms.<br>***Uses** a key object*<br>RSA, DSA, DES MAC, Triple DES MAC, HMAC |
| Sign Module State | - | handle, ACL | handle, ACL | Signs a certificate describing the modules security policy, as set by SetNSOPerm<br>***Uses** the module signing key* |
| Send Share | - | handle, ACL | handle, ACL | Reads a logical token share and encrypts it under an impath key for transfer to another module where it can be loaded using ReceiveShare<br>***Uses** an Impath key, **exports** encrypted share.* |
| Statistics Enumerate Tree | yes | yes | yes | Lists the statistics available.<br>*No access to keys or CSPs* |
| Statistic Get Value | yes | yes | yes | Returns a particular statistic.<br>*No access to keys or CSPs* |
| Verify | - | handle, ACL | handle, ACL | Verifies a digital signature using a stored key. See "Strength of functions" on page 26 for a list of supported algorithms.<br>***Uses** a key object*<br>RSA, DSA, DES MAC, Triple DES MAC, HMAC |

| Command/ Service | Role | | | Description<br>*Key/CSP access*<br>Key types |
|---|---|---|---|---|
| | Unuath | User / JSO | NSO | |
| Write File | - | cert | yes | Writes a file, but not a logical token, to a smart card or software token.<br>*No access to keys or CSPs* |
| Write Share | - | cert handle | handle | Writes a new share to a smart card or software token. The number of shares that can be created is specified when the token is created. All shares must be written before the token is destroyed.<br>***Sets** pass phrase, **uses** module key, **creates** share, **uses** pass phrase and module key, **creates** share key, **uses** module key, **uses** share key, **exports** encrypted share* |

| Code | Description |
|---|---|
| - | The user can not perform this service in this role. |
| yes | The user can perform this service in this role without further authorization. |
| handle | The user can perform this service if they possess a valid handle for the resource: key, channel, impath, token, buffers.<br>The handle is an arbitrary number generated when the object is created.<br>The handle for an object is specific to the user that created the object.<br>The ticket services enable a user to pass an ID for an object they have created to another user |
| ACL | The user can only perform this service with a key if the ACL for the key permits this service. The ACL may require that the user 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.* |
| pass phrase | A user can only load a share, or change the share PIN, if they possess the pass phrase used to encrypt the share. The module key with which the Pass Phrase was combined must also be present. |
| cert | A user can only perform this service if they are in possession of a certificate from the nCipher Security Officer. This certificate will reference a key.<br>*The module verifies the signature on the certificate before permitting the operation.* |
| FE | This service is not available on all modules. It must be enabled using the FeatureEnable service before it can be used. |

| Code | Description |
|---|---|
| level 2 only | These services are available to the unauthorized user only when the module is initialized in it FIPS 140-2 level 2 mode.<br>The module can be initialized to comply with FIPS 140-2 level 3 roles and services and key management by setting the fIPS_level3_compliance flag.<br>If this flag is set:<br>the Generate Key, Generate Key Pair and Import commands require authorization with a certificate signed by the nCipher Security Officer.<br>the Import command fails if you attempt to import a key of a type that can be used to Sign or Decrypt messages.<br>the Generate Key, Generate Key Pair, Import and Derive Key operations will not allow you to create an ACL for a secret key that allows the key to be exported in plain text. |
| encrypted share | The ReceiveShare command requires a logical token share encrypted using an Impath key. created by the SendShare command. |
| ticket | The RedeemTicket command requires the ticket generated by GetTicket |
| init | These services are used to initialise the module. They are only available when the module is in the initialisation mode. To put the module into initialisation mode you must have physical access to the module and put the mode switch into the initialisation setting. In order to restore the module to operational mode you must put the mode switch back to the Operational setting. |

| 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. |
| Set | Changes a CSP to a given value |
| Use | Performs an operation with an existing CSP - without revealing or changing the CSP |

# Keys

For each type of key used by the nForce modules, the following section describes the access that a user has to the keys.

The nForce modules refers to keys by their handle, an arbitrary number, or by its SHA-1 hash.

## Security Officer's key

The nCipher Security officer's key must be set as part of the initialisation process. This is a public/private key pair that the nCipher Security Officer uses to sign certificates to authorize key management and other secure operations.

The SHA-1 hash of the public half of this key pair is stored in the module EEPROM.

The public half of this key is included as plain text in certificates.

The module treats anyone in possession of the private half of this key as the Security Officer.

If you use the standard tools supplied by nCipher to initialise the module, then this key is a DSA key stored as a key blob protected by a logical token on the Administrator Card Set. If a customer writes their own tools to initialise the module, they can choose between RSA, DSA or KDSA, and are responsible for ensuring the private half of this key is stored securely.

## Junior Security Officer's key

Because the nCipher Security Officer's key has several properties, it is good practice to delegate authority to one or more Junior Security Officers, each with authority for defined operations.

To create a Junior Security Officer (JSO) the NSO creates a certificate signing key for use as their JSO key. This key must be protected by a logical token in the same manner as any other application key.

Then to delegate authority to the JSO the nCipher Security Officer creates a certificate containing an Access Control List specifying the authority to be delegated and the hash of the JSO key to which the powers are to be delegated.

The JSO can then authorize the actions listed in the ACL - as if they were the NSO - by presenting the JSO key and the certificate.

If the JSO key is created with the Sign permission in its ACL the JSO may delegate parts of their authority to another key, the holder of the delegate key will need to present the certificate singed by the NSO and the certificate singed by the JSO. If the JSO key only has UseAsCertificate permissions, then they cannot delegate authority.

If you use the standard tools supplied by nCipher to initialise the module, then this key is a DSA key stored as a key blob protected by a logical token on the Administrator Card Set. If a customer writes their own tools to initialise the module, they can choose between RSA, DSA or KDSA, and are responsible for ensuring the private half of this key is stored securely.

## Long term signing key

The nForce modules stores a 160 bits in the EEPROM provided by the Dallas 4320. This data is combined with a discrete log group stored in the module firmware to produce a DSA key.

This key can be reset to a new random value by the GenerateKLF service. It can be used to sign a module state certificate using the SignModuleState service and the public value retrieved by the non-cryptographic service Get Long Term Key.

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.

## Module signing key

When the nForce modules is initialized it automatically generate a DSA key pair that it uses to sign certificates. 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 under some other key. This key is only ever used to verify that a certificate was generated by a specified module.

## Module keys

Module keys are triple DES used to protect tokens. The nForce modules generates the first module key $K_{M0}$ when it is initialized. This module key is guaranteed never to have been known outside this module.

The Security Officer can load further module keys. These can be generated by the module or may be loaded from an external source. Setting a key as a module key stores the key in EEPROM.

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.

## Logical tokens

A logical token is a Triple DES key used to protect key blobs.

When you create a logical token, you must specify parameters, including the total number of shares, and the number or shares required to recreate the token, the quorum. The total number can be any integer between 1 and 64 inclusive. The quorum can be any integer from 1 to the total number.

A logical token is always generated randomly, using the on-board random number generator.

While loaded in the module logical tokens are stored in the object store.

Tokens keys are never exported from the module, except on physical tokens or software tokens. When a module key is exported the logical token - the Triple DES key plus the token parameters - is first encrypted with a module key. Then the encrypted token is split into shares using the Shamir Secret Sharing algorithm, even if the total number of shares is one. Each share is then encrypted using a share key and written to a physical token - a smart card - or a software token. Logical tokens can be shared between one or more physical token. The properties for a token define how many shares are required to recreate the logical token. Shares can only be generated when a token is created. The firmware prevents a specific share from being written more than once.

## Share Key

A share key is used to protect a logical token share when they are written to a smart card or software token. A share may be protected using a pass phrase. The pass phrase is combined with a module key, the share number and the smart card id to form a unique Triple DES key used to encrypt the share. This Triple DES key is not stored on the module. It is recalculated every time share is loaded. The share data includes a MAC, if the pass phrase is incorrect the MAC will not verify correctly and the share is rejected.

## Impath keys

An impath is a secure channel between two modules.

To set up an impath two modules perform a validated key-exchange, currently using Diffie Helman.

The key exchange parameters are signed by the modules signing key Once the modules have validated the signatures the module derives four symmetric keys using a protocol based on SHA-1. Currently symmetric keys are Triple DES.

The four keys are used for encryption, decryption, MAC creation, MAC validation. The protocol ensures that the key Module 1 uses for encryption is used for decryption by module 2.

All impath keys are stored as objects in the object store in DRAM.

## Key objects

Keys used for encryption, decryption, signature verification and digital signatures are stored in the module as objects in the object store in DRAM. All key objects are identified by a random identifier that is specific to the user 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 a user 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 there 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 a logical token. Key objects may also be exported as key blobs under an archiving key.

Key objects can only be exported in plain text if their ACL permits this operation. If the module has been initialized to comply with FIPS 140-2 level 3 roles and services and key management the ACL for a private or secret key cannot include the export as plain service.

A user may pass a key to another user 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 user 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 user is still bound by the original ACL.

## Session keys

Keys used for a single session are generated as required by the module. They are stored along with their ACL as objects in the object store. These may be of any supported algorithm.

## Archiving keys

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

- Triple DES keys
- A combination of Triple DES and RSA keys.
  In this case a random Triple DES key is created which is used to encrypt working key and then this key is wrapped by the RSA key.

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

When you generate or import the archiving, 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.

## Certificate signing keys

The ACL associated with a key object can call for a certificate to be presented to authorize the action. The required key can either be the nCipher Security Officer's key or any other key. Keys are specified in the ACL by an identifying key SHA-1 hash. The key type is also specified in the ACL although DSA is standard, any signing algorithm may be used, all nCipher tools use DSA.

Certain services can require certificates signed by the nCipher Security Officer.

## Firmware Integrity Key

All firmware is signed using a DSA key pair. A module only loads 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 DRAM as part of the start up procedure.

## Firmware Confidentiality Key

All firmware is encrypted using triple DES 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 DRAM as part of the start up procedure.

## nCipher Master Feature Enable Key

For commercial reasons not all nCipher modules offer all services. Certain services must be enabled separately. In order to enable a service the user presents a certificate signed by the nCipher Master Feature Enable Key. This causes the module to set the appropriate bit in the Dallas EEPROM.

The nCipher 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 DRAM as part of the start up procedure.

# Rules

## Identification and authentication

All communication with the nForce modules is performed via a server program running on the host machine, using standard inter process communication, using sockets in UNIX operating system, named pipes under Windows NT.

In order to use the module the user must first log on to the host computer and start an nCipher enabled application. The application connects to the nCipher server, this connection is given a client ID, a 120-bit arbitrary number.

Before performing any service the user must present the correct authorization. Where several stages are required to assemble the authorization, all the steps must be performed on the same connection.

## Access Control

Keys are stored on the host computer's hard disk in an encrypted format, known as a key blob. In order to load a key the user must first load the token used to encrypt this blob.

Tokens can be divided into shares. Each share can be stored on a smart card or software token on the computer's hard disk. These shares are further protected by encryption with a pass phrase and a module key. Therefore a user can only load a key if they possess the physical smart cards containing sufficient shares in the token, the required pass phrases and the module key are loaded in the module.

Module keys are stored in EEPROM in the module. They can only be loaded or removed by the nCipher Security Officer, who is identified by a public key pair created when the module is initialized. It is not possible to change the Security Officer's key without re initializing the module, which clears all the module keys, thus preventing access to all other keys.

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. If a per-authorization limit is reached the user must reauthorize the key by reloading the token.

All objects are referred to by handle. Handles are cross-referenced to client IDs. 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 clientIDs.

## Access Control List

All key objects have an Access Control List (ACL). The user 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 objected 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. If a per authorization limit is exceeded then the logical token protecting the key must be reloaded before the key can be reused.

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

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

## 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.

## Error conditions

If the nForce modules 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 user can resubmit the command at a later time. The server program can record a log of all such failures.

If the nForce modules 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. The unit must be reset.

## Security Boundary

The security boundary is the PCB

The following items are excluded from FIPS 140-2 validation as they are not security relevant.

- PCI connector
- jumper block
- clear button
- status LED
- heatsinks
- serial connector
- mini-DIN connector

The SEE functionality is excluded from this validation as nCipher does not licence SEE on nForce modules because these modules lack a real time clock or non-volatile RAM required to make full use of SEE.

## To comply with FIPS 140-2 level 3 roles and services and key management

The nCipher enabled application must perform the following services, for more information refer to the nCipher Security Officer's Guide and Technical Reference Manual.

## To initialise a module

1. Fit the initialisation link and restart the module
2. Use the Initialise command to enter the Initialisation state.
3. Generate a key pair to use a Security Officer's key.
4. Generate a logical token to use to protect the Security Officer's key.
5. Write one or more shares of this token onto software tokens.
6. Export the private half of the Security Officer's key as a key blob under this token.
7. Export the public half of the Security Officer's key as plain text.
8. Use the Set Security Officer service to set the Security Officer's key and the operational policy of the module. In order to comply with FIPS 140-2 level 3 roles and services and key management you must set at least the following flags:
   - NSOPerms_ops_ReadFile
   - NSOPerms_ops_WriteFile
   - NSOPerms_ops_EraseShare
   - NSOPerms_ops_EraseFile
   - NSOPerms_ops_FormatToken
   - NSOPerms_ops_GenerateLogToken
   - NSOPerms_ops_SetKM
   - NSOPerms_ops_RemoveKM
   - NSOPerms_ops_StrictFIPS140
9. Keep the tokens and key blobs safe.
10. You can create extra module keys in order to distinguish groups of users.
11. You may want to create working keys and user authorization at this stage.
12. Remove the initialisation link and restart the module.

*Note* *nCipher supply a graphical user interface KeySafe and a command line tool new-world that automate these steps.*

- If you use KeySafe, you must set the **StrictFIPS 140** flag.
- If you use new-world, you must select the **-F** flag.

## To return a module to factory state

This clears the Security Officer's key, the module signing key and any loaded module keys.

1. Fit the initialisation link and restart the module
2. Use the Initialise command to enter the Initialisation state.
3. Load a random value to use as the hash of the security officer's key.
4. Set Security Officer service to set the Security Officer's key and the operational policy of the module.
5. Remove the initialisation link and restart the module.

After this operation the module must be initialized correctly before it can be used in a FIPS approved mode.

*Note* *nCipher supply a graphical user interface KeySafe and a command line tool new-world that automate these steps.*

### To create a new user

1. Create a logical token.
2. Write one or more shares of this token onto software tokens.
3. For each key the user will require, export the key as a key blob under this token.
4. Give the user any pass phrases used and the key blob.

*Note* *nCipher supply a graphical user interface KeySafe and a command line tool new-world that automate these steps.*

### To authorize the user to create keys

1. Create a new key, with an ACL that only permits UseAsSigningKey. This action may need to be authenticated.
2. Export this key as a key blob under the users token.
3. Create a certificate signed by the nCipher Security Officer's key that:
   - includes the hash of this key as the certifier
   - authorizes the action GenerateKey or GenerateKeyPair depending on the type of key required.
   - if the user needs to store the keys, enables the action MakeBlob, limited to their token.
4. Give the user the key blob and certificate.

*Note* *nCipher supply a graphical user interface KeySafe and a command line tool new-world that automate these steps.*

### To authorize a user to act as a Junior Security Officer

1. Generate a logical token to use to protect the Junior Security Officer's key.
2. Write one or more shares of this token onto software tokens
3. Create a new key pair,
   - Give the private half an ACL that permits Sign and UseAsSigningKey.
   - Give the public half an ACL that permits ExportAsPlainText
4. Export the private half of the Junior Security Officer's key as a key blob under this token.
5. Export the public half of the Junior Security Officer's key as plain text.
   - Create a certificate signed by the nCipher Security officer's key includes the hash of this key as the certifier
   - authorizes the actions GenerateKey, GenerateKeyPair
   - authorizes the actions GenerateLogicalToken, WriteShare and MakeBlob, these may be limited to a particular module key.
6. Give the Junior Security Officer the software token, any pass phrases used, the key blob and certificate.

*Note* *nCipher supply a graphical user interface KeySafe and a command line tool new-world that automate these steps.*

### To authenticate a user to use a stored key

1. Use the LoadLogicalToken service to create the space for a logical token.
2. Use the ReadShare service to read each share from the software token.
3. Use the LoadBlob service to load the key from the key blob.
4. The user can now perform the services specified in the ACL for this key.

*Note* *To assume Security Officer role load the Security Officer's key using this procedure. The Security Officer's key can then be used in certificates authorising further operations.*

*Note* *nCipher supply a graphical user interface KeySafe and a command line tool new-world that automate these steps.*

### To authenticate a user to create a new key

1. If you have not already loaded your user token, load it as above.
2. Use the LoadBlob service to load the authorization key from the key blob.
3. Use the KeyId returned to build a signing key certificate.
4. Present this certificate with the certificate supplied by the security officer with the GenerateKey, GenerateKeyPair or MakeBlob command.

*Note* *nCipher supply a graphical user interface KeySafe and a command line tool new-world that automate these steps.*

# Physical security

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

The module has a clear button. Pressing this button put 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, module keys, module signing key and Security Officer's key can be cleared by returning the module to the factory state, as described above.

## Checking the module

To ensure physical security, make the following checks regularly:

- Examine the epoxy resin security coating for obvious signs of damage.
- The smart card reader is directly plugged into the module and the cable has not been tampered with.

# Strength of functions

## Attacking Object IDs

Connections are identified by a ClientID, a random 32 bit number.

Objects are identified by am KeyID - this should be renamed ObjectID as it can now be used for more than just keys but retains its old name for code compatibility reasons. Again this is a random 32 bit number.

In order to randomly gain access to a key loaded by another user you would need to guess two random 32 bit numbers. The module can process about $2^{16}$ commands per minute - therefore the chance of succeeding within a minute is $2^{16} / 2^{64} = 2^{-48}$.

## Attacking Tokens

If a user chooses to use a logical token with only one share, no pass phrase and leaves the smart card containing the share in the slot than another user could load the logical token. The module does not have any idea as to which user inserted the smart card. this can be prevented by:

- not leaving the smart card in the reader
  if the smart card is not in the reader, they can only access the logical token by correctly guessing the ClientID and ObjectID for the token.
- requiring a pass phrase
  hen loading a share requiring a pass phrase the user must supply the SHA-1 hash of the pass phrase. The hash is combined with a module key, share number and smart card id to recreate the key used to encrypt the share. If the attacker has no knowledge of the pass phrase they would need to make $2^{80}$ attempts to load the share. The module enforces a five seconds delay between failed attempts to load a share.
- requiring more than one share
  if a logical token requires shares from more than one smart card the attacker would have to repeat the attack for each share required.

Logical tokens are 168 bit Triple DES keys. Shares are encrypted under a combination of a module key, share number and card ID. If you could construct a logical token share of the correct form without knowledge of the module key and the exact mechanism used to derive the share key the chance that it would randomly decrypt into a valid token are $2^{-168}$.

# Algorithms

## FIPS approved algorithms:

| | |
|---|---|
| AES | Certificate #15 |
| DES | Certificate #24 |
| DES MAC | Certificate #24 vendor affirmed. Verified implementation functions: genblk_signbegin,genblk_verifybegin, genblk_macdata, genblk_signend,genblk_verifyend. |
| Triple DES | Certificate #34<br><br>Double and triple length keys<br>Approved for Federal Government Use - Modes are CBC and ECB |
| Triple DES MAC | Certificate #34 vendor affirmed<br><br>Verified implementation functions: genblk_signbegin,genblk_verifybegin, genblk_macdata, genblk_signend,genblk_verifyend |
| DSA/SHA-1 | Certificate #11 |
| RSA | Vendor Affirmed to PKCS #1 |
| HMAC SHA-1 | HMAC implementation vendor affirmed, SHA-1 covered by DSA certificate #11 |

## Non-FIPS approved algorithms

### Symmetric

- Arc Four (compatible with RC4)
- Blowfish
- CAST 5 (RFC2144)
- CAST 6 (RFC2612)
- Serpent
- SEED (Korean Data Encryption Standard) - requires Feature Enable activation
- Twofish

### Asymmetric

- Diffie-Helman
- El Gamal
- KCDSA - requires Feature Enable activation

### Hash

- HSA-160 - requires Feature Enable activation
- MD2
- MD5
- RIPEMD 160
- SHA-256
- SHA-384
- SHA-512

### MAC

- HMAC (MD2, MD5, SHA-256, SHA-384, SHA-512, RIPEMD160)

### Other

- SSL master key derivation
- PKCS #8 key wrapping.