nShield Security Policy

MiniHSM in FIPS 140-2 level 3 mode
# Chapter 1: Purpose

- **Introduction**
- **Module Ports and Interfaces**
- **Excluded Components**
- **Roles**
  - Unauthorised
  - User
  - nCipher Security Officer
  - Junior Security Officer
- **Services available to each role**
- **Keys**
  - Security Officer's key
  - Junior Security Officer's key
  - Long term signing key
  - Module signing key
  - Module keys
  - Logical tokens
  - Share Key
  - Impath keys
  - Key objects
  - Session keys
  - Archiving keys
  - Certificate signing keys
  - Firmware Integrity Key
  - Firmware Confidentiality Key
  - nCipher Master Feature Enable Key
- **Rules**
  - Identification and authentication
  - Procedures to initialise a module to comply with FIPS 140-2 Level 3
  - Procedures to initialise a module to comply with FIPS 140-2 Level 2
  - Operating a level 2 module in FIPS mode
  - To return a module to factory state
  - To create a new operator
  - To authorize the operator to create keys
  - To authorize an operator to act as a Junior Security Officer
  - To authenticate an operator to use a stored key
  - To authenticate an operator to create a new key
- **Physical security**
  - Checking the module
Strength of functions 42
   Attacking Object IDs 42
   Attacking Tokens 42
   Key Blobs 43
   Impaths 43
   KDP key provisioning 43
   Derived Keys 43
Self Tests 45
   Firmware Load Test 45
Supported Algorithms 47
   FIPS approved algorithms: 47
   Non-FIPS approved algorithms 48

nCipher addresses 50
Chapter 1: Purpose

Introduction

MiniHSM tamper resistant Hardware Security Modules are multi-tasking hardware modules that are optimized for performing modular arithmetic on very large integers. The modules also offer a complete set of key management protocols.

The MiniHSM provides the security for the nCipher's TSMC products.

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

<table>
<thead>
<tr>
<th>Unit ID</th>
<th>Model Number</th>
<th>RTC</th>
<th>NVRAM</th>
<th>SEE</th>
<th>Potting</th>
<th>EMC</th>
<th>Crypto Accelerator</th>
<th>Overall level</th>
</tr>
</thead>
<tbody>
<tr>
<td>nCipher MiniHSM</td>
<td>nC4031Z-10</td>
<td>Yes</td>
<td>Optional</td>
<td>Yes</td>
<td>B</td>
<td>none</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

The module has been validated at build standard A, B and build standard N.

Build standard "N" meets the latest EU regulations regarding ROHS.

nCipher also supply modules to third party OEM vendors for use in a range of security products.

The module runs firmware provided by nCipher. There is the facility for the administrator 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.

The validated firmware versions are 2.22.17 and 2.22.34.

The module can be initialised to comply with the requirements for Roles and Services at either level 2 or level 3.

- When initialized in level 2 mode the firmware version is 2.22.17-2 or 22.22.34-2 (level 2 mode) and the level 2 certificate applies.

- When initialized in level 3 mode the firmware version is 2.22.6-3, 2.22.34-3, 2.22.43-3 (level 3 mode) and the level 3 certificate applies.

When initialized in level 3 mode the firmware version is 2.22.17-3 or 22.22.34-3 (level 3 mode) and the level 3 certificate applies.
The initialization parameters are reported by the NewEnquiry and SignModuleState services. An operator 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 modules must be accessed by a custom written application. Full documentation for the nCore API can be downloaded from the nCipher web site: http://www.nCipher.com.

MinHSMs have on-board non-volatile memory. There are services that enable memory to be allocated as files. Files have Access Control Lists that determine what operations can be performed on their contents. nShield modules have on-board Real-time clock.

MiniHSMs include a technology called, the Secure Execution Environment (SEE). This enables operators to load a SEE machine. A SEE machine is operator written code that implements a specific Software Interrupt interface. This enables operators to implement non-cryptographic code in a protected memory space on the module that is outside the logical security boundary.

SEE code is executed in a protected environment. Whenever the SEE machine is running the nCore kernel is locked. Whenever the nCore kernel is active the SEE machine is locked. The SEE machine is excluded from the requirements of FIPS PUB 140-2. While the SEE machine is active the module is running in a non-FIPS mode.

The SEE machine has no direct access to objects stored on the module. In order to use cryptographic functions it must submit a job to the nCore kernel using the nCore API. The testing shows that the interface between the nCore kernel and the SEE machine is secure and that a malicious SEE machine cannot gain access to objects protected by the nCore kernel.

Before a operator can send commands to the SEE machine they must create an instance called a SEE World. A SEE World is treated as a separate operator by the module and must present the correct authentication before they can submit commands to the nCore kernel.

MiniHSM are supplied with the SEE functions disabled. In order to use these functions the customer must purchase a feature-enable certificate enabling the functions for a specific module. The SEE feature is export controlled and therefore is not available in some territories.

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

- Windows
- Solaris
- HP-UX
- AIX
- Linux x86
• FreeBSD x86

Windows XP Professional SP2 and a Debian Linux distribution with kernel version 2.2.26 were used to test the module for this validation.
Module Ports and Interfaces

The following table lists the logical interfaces to the module and how they map to physical interfaces.

<table>
<thead>
<tr>
<th>Logical Interface</th>
<th>Physical Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data In</td>
<td>serial bus, Serial Interface,</td>
</tr>
<tr>
<td>Data Out</td>
<td>serial bus, Serial Interface,</td>
</tr>
<tr>
<td>Control In</td>
<td>serial bus, Temperature SensorMode pins</td>
</tr>
<tr>
<td>Status Out</td>
<td>serial bus, LED pin</td>
</tr>
<tr>
<td>Power</td>
<td>power pins</td>
</tr>
</tbody>
</table>
**Excluded Components**

The following components are excluded from FIPS 140-2 validation:

- 4 pin serial interface (RTS, CTS, data, data_ground) used for commands
- 4 pin serial interface (RTS, CTS, data, data_ground) for connection to smart card reader.
- Mode pins
- Clear pin
- Reset pin
- Status LED pin
- SEE machine

The mode pins must be connected to an external three position switch, referred to elsewhere as the mode switch.

The clear pin must be connected to an external push to make button - referred to as the clear button. This enables an operator to clear the module in a controlled manner which ensures communication on the host serial port is not disrupted.

The reset pin instantly clears the module - this will disrupt communication on the serial port, so should only be used if all components that communicate with the module are also being reset.

The LED pin is designed to be connected to the anode of an external LED, the cathode of which is connected to earth. Statuses are indicated by a change in voltage on this pin - which will cause an external LED to flash.

The smart card serial interface must be attached to an external 9-pin D-type connector to allow connection of an nCipher smart card reader.
Roles

The module defines the following roles:

Unauthorised

All connections are initially unauthorized. If the module is initialized in level 3 mode, an unauthorized operator is restricted to status commands, and commands required to complete authorization protocol.

User

An operator enters the user role by providing the required authority to carry out a service. The exact accreditation required to perform each service is listed in the table of services.

In order to perform an operation on a stored key, the operator must first load the key blob. If the key blob is protected by a logical token, the operator must first load the logical token by loading shares from smart cards.

If the module is initialized in level 3 mode, the user role requires a certificate from the security officer to import or generate a new key. This certificate is linked to a token protected key.

Once an operator in the user role has loaded a key they can then use this key 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 operator and ensuring that these tokens are physically handed to the correct person.

An operator 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 operator loads the JSO key and presents the KeyID of this key, and if required the certificate signed by $K_{NSO}$ that delegates authority to the key, to authorize a command.

A JSO can delegate portions of their authority to a new operator in the same way. The new operator will be a JSO if they have authority they can delegate, otherwise they will assume the user role.
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.
The following services provide authentication or cryptographic functionality. The functions available depend on whether the operator is in the unauthenticated role, the user or junior security officer (JSO) roles, or the nCipher Security Officer (NSO) role. For each operation it lists the supported algorithms. Algorithms in square brackets are not under the operator's control. Algorithms used in optional portions of a service are listed in italics.

Note: Algorithms marked with an asterisk are not approved by NIST. If the module is initialised in its level 3 mode, these algorithms are disabled. If module is initialized in level 2 mode, the algorithms are available. However, if you choose use them, the module is not operating in FIPS approved mode.

### Key Access

<table>
<thead>
<tr>
<th>Command / Service</th>
<th>Role</th>
<th>Unauth</th>
<th>JSO / User</th>
<th>NSO</th>
<th>Description</th>
<th>Key/CSP access</th>
<th>Key types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bignum Operation</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Performs simple mathematical operations.</td>
<td>No access to keys or CSPs</td>
<td>[SHA-1 and AES or Triple DES]</td>
</tr>
<tr>
<td>Change Share PIN</td>
<td>No</td>
<td>pass phrase</td>
<td>pass phrase</td>
<td></td>
<td>Updates the pass phrase used to encrypt a token share. The pass phrase supplied by the operator 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.</td>
<td>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</td>
<td></td>
</tr>
<tr>
<td>Command / Service</td>
<td>Role</td>
<td>Unauth</td>
<td>JSO / User</td>
<td>NSO</td>
<td>Description</td>
<td>Key/CSP access</td>
<td>Key types</td>
</tr>
<tr>
<td>------------------</td>
<td>------</td>
<td>--------</td>
<td>------------</td>
<td>-----</td>
<td>-------------</td>
<td>----------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Channel Open</td>
<td>No</td>
<td>handle</td>
<td>ACL</td>
<td>handle, ACL</td>
<td>Opens a communication channel which can be used for bulk encryption or decryption.</td>
<td>Uses a key object</td>
<td>AES, DES*, Triple DES, Arc Four*, Aria*, Camellia*, SEED*</td>
</tr>
<tr>
<td>Channel Update</td>
<td>No</td>
<td>handle</td>
<td>handle</td>
<td>handle</td>
<td>Performs encryption/decryption on a previously opened channel. The operation and key are specified in ChannelOpen.</td>
<td>Uses a key object</td>
<td>AES, DES*, Triple DES, Arc Four*, Aria*, Camellia*, SEED*</td>
</tr>
<tr>
<td>CheckUserACL</td>
<td>No</td>
<td>handle</td>
<td>handle</td>
<td>handle</td>
<td>Determines whether the ACL associated with a key object allows a specific operator defined action.</td>
<td>Uses a key object</td>
<td></td>
</tr>
<tr>
<td>Clear Unit</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Zeroises all loaded keys, tokens and shares. Clear Unit does not erase long term keys, such as module keys.</td>
<td>Zeroises objects.</td>
<td>All</td>
</tr>
<tr>
<td>Create Buffer</td>
<td>No</td>
<td>cert [handle]</td>
<td>cert [handle]</td>
<td></td>
<td>Allocates an area of memory to load data. If the data is encrypted, this service specifies the encryption key and IV used. This service is feature enabled. The decrypt operation is performed by LoadBuffer.</td>
<td>Uses a key object</td>
<td>AES, DES*, Triple DES, Arc Four*, Aria*, Camellia*, SEED*</td>
</tr>
<tr>
<td>Create SEE World</td>
<td>No</td>
<td>handle cert</td>
<td>handle cert</td>
<td></td>
<td>Creates a SEE World, passing in the initialization data stored in a buffer. This command checks the DSA signatures on the buffer using the public key provided. It also specifies whether debugging is allowed or not. Enabling debugging requires a certificate from the nCipher Security Officer.</td>
<td>No access to keys or CSPs</td>
<td></td>
</tr>
<tr>
<td>Command / Service</td>
<td>Role</td>
<td>Description</td>
<td>Key/CSP access</td>
<td>Key types</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
<td>-------------</td>
<td>----------------</td>
<td>-----------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decrypt</td>
<td>No</td>
<td>Decrypts a cipher text with a stored key returning the plain text.</td>
<td>Uses a key object</td>
<td>AES, DES*, Triple DES, Arc Four*, Aria*, Camellia*, SEED*, Diffie-Hellman, ECDH, RSA*, ElGamal*, KCDSA*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Derive Key</td>
<td>No</td>
<td>The nCipher DeriveKey service provides functions that the FIPS 140-2 standard describes as key wrapping and split knowledge - it does not provide key derivation in the sense understood by FIPS 140-2. 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 wrap keys according to the nCipher KDP so that a key server can distribute the wrapped key to micro-HSM devices.</td>
<td>Uses a key object, create a new key object.</td>
<td>AES, RSA, EC-DH, EC_MQV, Triple DES, PKCS #8*, TLS key derivation, XOR, DLIES (D/H plus Triple DES or D/H plus AES), Aria*, Arc Four*, Camellia*, DES*, SEED*, KCDSA*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destroy</td>
<td>No</td>
<td>Removes an object, if an object has multiple handles as a result of RedeemTicket service, this removes the current handle.</td>
<td>Erases a Impath, SEEWorld, logical token, or any key object.</td>
<td>All</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Duplicate</td>
<td>No</td>
<td>Creates a second instance of a key object with the same ACL and returns a handle to the new instance.</td>
<td>Creates a new key object.</td>
<td>All</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encrypt</td>
<td>No</td>
<td>Encrypts a plain text with a stored key returning the cipher text.</td>
<td>Uses a key object</td>
<td>AES, DES*, Triple DES, RSA*, ElGamal*, Arc Four*, Aria*, Camellia*, SEED*, KCDSA*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command / Service</td>
<td>Role</td>
<td>Description</td>
<td>Key/CSP access</td>
<td>Key types</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
<td>-------------</td>
<td>----------------</td>
<td>----------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Erase File</td>
<td>level 2 only</td>
<td>Removes a file, but not a logical token, from a smart card or software token.</td>
<td>No access to keys or CSPs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Erase Share</td>
<td>level 2 only</td>
<td>Removes a share from a smart card or software token.</td>
<td>Erases a share</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Existing Client</td>
<td>yes</td>
<td>Starts a new connection as an existing client.</td>
<td>No access to keys or CSPs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Export</td>
<td>No handle, ACL</td>
<td>If the unit is operating in FIPS mode this operation is only available for public keys - see Operating a level 2 module in FIPS mode on page 38. 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.</td>
<td>Exports a [public] key object.</td>
<td>Level 2 mode - Any key type Level 3 mode - RSA, DSA, ECDSA, Diffie-Hellman, El-Gamal and ECDH public keys</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feature Enable</td>
<td>No cert</td>
<td>Enables a service. This requires a certificate signed by the nCipher Master Feature Enable key.</td>
<td>Uses the public half of the nCipher Master Feature Enable Key</td>
<td>[DSA]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Firmware Authenticate</td>
<td>yes</td>
<td>Performs a zero knowledge challenge response protocol based on HMAC that enables an operator to ensure that the firmware in the module matches the firmware supplied by nCipher. The protocol generates a random value to use as the HMAC key.</td>
<td>No access to keys or CSPs</td>
<td>HMAC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Foreign Token Command (Bypass)</td>
<td>No handle</td>
<td>Sends an ISO-7816 command to a smart card over the channel opened by ForeignTokenOpen.</td>
<td>Uses a bypass channel.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Foreign Token Open (Bypass)</td>
<td>No FE, cert</td>
<td>Opens a channel to foreign smart card that accepts ISO-7816 commands. This service cannot be used if the smart card has been formatted using FormatToken. The channel is closed when the card is removed from the reader, or if the handle is destroyed. This service is feature enabled.</td>
<td>Creates a bypass channel.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command / Service</td>
<td>Role</td>
<td>Description</td>
<td>Key/CSP access</td>
<td>Key types</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------</td>
<td>--------------------------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FormatToken</td>
<td>level 2 only</td>
<td>Formats a smart card or software token ready for use.</td>
<td>May use a module key to create challenge response value</td>
<td>[AES, Triple DES]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generate Key</td>
<td>level 2 only</td>
<td>Generates a symmetric key of a given type with a specified ACL and returns a handle. Optionally returns a certificate containing the ACL.</td>
<td>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.</td>
<td>AES, Triple DES, Arc Four*, Aria*, Camellia*, DES*, SEED* [DSA]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generate Key Pair</td>
<td>level 2 only</td>
<td>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.</td>
<td>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.</td>
<td>Diffie-Helman, DSA, ECDSA, EC-DH, EC-MQV, RSA, ElGamal*, KCDISA* [DSA]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generate KLF</td>
<td>No</td>
<td>Generates a new long term key.</td>
<td>Erases the module long term key, creates new module long term key.</td>
<td>[DSA]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generate Logical Token</td>
<td>level 2 only</td>
<td>Creates a new logical token, which can then be written as shares to smart cards or software tokens</td>
<td>Uses module key. Creates a logical token.</td>
<td>[AES or Triple DES]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get ACL</td>
<td>No</td>
<td>Returns the ACL for a given handle.</td>
<td>Exports the ACL for a key object.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command / Service</td>
<td>Role</td>
<td>Description</td>
<td>Key/CSP access</td>
<td>Key types</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------------------------------------------</td>
<td>-----------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get Application Data</td>
<td>Unauth: No, User: handle, ACL</td>
<td>Returns the application information stored with a key.</td>
<td>Exports the application data of a key object.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get Challenge</td>
<td>User: yes, JSO / User: yes</td>
<td>Returns a random nonce that can be used in certificates</td>
<td>No access to keys or CSPs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get Key Info</td>
<td>NSO / User: handle</td>
<td>Superseded by GetKeyInfoEx retained for compatibility.</td>
<td>Exports the SHA-1 hash of a key object</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get Key Info Extended</td>
<td>NSO / User: handle</td>
<td>Returns the hash of a key for use in ACLs</td>
<td>Exports the SHA-1 hash of a key object</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get Logical Token Info</td>
<td>NSO / User: handle</td>
<td>Superseded by GetLogicalTokenInfoEx retained for compatibility.</td>
<td>Exports the SHA-1 hash of a logical token</td>
<td>[SHA-1]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get Logical Token Info Extended</td>
<td>NSO / User: handle</td>
<td>Returns the token hash and number of shares for a logical token</td>
<td>Exports the SHA-1 hash of a logical token</td>
<td>[SHA-1]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get Module Keys</td>
<td>User: yes, JSO / User: yes</td>
<td>Returns a hashes of the Security Officer's key and all loaded module keys .</td>
<td>Exports the SHA-1 hash of KNSO and module keys.</td>
<td>[SHA-1]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get Module Long Term Key</td>
<td>User: yes, JSO / User: yes</td>
<td>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.</td>
<td>Exports the public half of the module’s long term key.</td>
<td>[DSA]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get Module Signing Key</td>
<td>User: yes, JSO / User: yes</td>
<td>Returns the public half of the module’s signing key. This can be used to verify certificates signed with this key.</td>
<td>Exports the public half of the module’s signing key.</td>
<td>[DSA]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get RTC</td>
<td>User: yes</td>
<td>Reports the time according to the on-board real-time clock</td>
<td>No access to keys or CSPs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get Share ACL</td>
<td>User: yes</td>
<td>Returns the access control list for a share</td>
<td>Exports the ACL for a token share on a smart card.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command / Service</td>
<td>Role Unauth</td>
<td>JSO / User</td>
<td>NSO</td>
<td>Description</td>
<td>Key/CSP access</td>
<td>Key types</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
<td>------------</td>
<td>-----</td>
<td>-------------</td>
<td>----------------</td>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>GetSlot Info</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>Reports status of the physical token in a slot. Enables an operator to determine if the correct token is present before issuing a ReadShare command. If the token was formatted with a challenge response value, uses the module key to authenticate the smart card.</td>
<td>Uses a module key if token is formatted with a challenge response value.</td>
<td>[AES, Triple DES]</td>
<td></td>
</tr>
<tr>
<td>Get Slot List</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>Reports the list of slots available from this module.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GetTicket</td>
<td>No</td>
<td>handle</td>
<td>handle</td>
<td>Gets a ticket - an invariant identifier - for a key. This can be passed to another client or to a SEE World which can redeem it using RedeemTicket to obtain a new handle to the object.</td>
<td>Uses a key object, logical token, Impath, SEEWorld.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get World Signers</td>
<td>No</td>
<td>handle</td>
<td>handle</td>
<td>Returns a list of the keys used to sign a SEEWorld identified by the key hash and the signing mechanism used. This command can only be called from inside the SEE world.</td>
<td>Uses a SEEWorld object.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hash</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>Hashes a value.</td>
<td>No access to keys or CSPs</td>
<td>HSA-160*, MD5*, RIPEMD 160*, SHA-1, SHA-256, SHA-384, SHA-512</td>
<td></td>
</tr>
<tr>
<td>Impath Get Info</td>
<td>No</td>
<td>handle</td>
<td>handle</td>
<td>Reports status information about an impath</td>
<td>Uses an Impath, exports status information.</td>
<td>[DSA and Diffie Hellman] AES, Triple-DES</td>
<td></td>
</tr>
<tr>
<td>Impath Key Exchange Begin</td>
<td>FE</td>
<td>FE</td>
<td>FE</td>
<td>Creates a new inter-module path and returns the key exchange parameters to send to the peer module.</td>
<td>Creates a set of Impath keys</td>
<td>[DSA and Diffie Hellman] AES, Triple-DES</td>
<td></td>
</tr>
<tr>
<td>Impath Key Exchange Finish</td>
<td>No</td>
<td>handle</td>
<td>handle</td>
<td>Completes an impath key exchange. Require the key exchange parameters from the remote module.</td>
<td>Creates a set of Impath keys.</td>
<td>[DSA and Diffie Hellman, AES, Triple-DES]</td>
<td></td>
</tr>
<tr>
<td>Impath Receive</td>
<td>No</td>
<td>handle</td>
<td>handle</td>
<td>Decrypts data with the Impath decryption key.</td>
<td>Uses an Impath key.</td>
<td>[AES or Triple DES]</td>
<td></td>
</tr>
<tr>
<td>Command / Service</td>
<td>Role</td>
<td>Key types</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
<td>-----------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Unauth</td>
<td>JSO / User</td>
<td>NSO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Impath Send</td>
<td>No</td>
<td>handle</td>
<td>handle</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Import</td>
<td>level 2 only</td>
<td>cert</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initialise</td>
<td>init</td>
<td>init</td>
<td>init</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load Blob</td>
<td>No</td>
<td>handle</td>
<td>handle</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Description**

- **Impath Send**: Encrypts data with the impath encryption key.
- **Import**: Loads a key and ACL from the host and returns a handle. 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 38. 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.
- **Initialise**: Initializes the module, returning it to the factory state. This clears all NVRAM files, all loaded keys and all module keys and the module signing key. It also generates a new KM0 and module signing key. 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 nCipher module. This key is not used to encrypt any other data.
- **Load Blob**: Loads a key that has been stored in a key blob. The operator must first have loaded the token or key used to encrypt the blob.

**Key/CSP access**

- Uses an Impath key.
- Creates a new key object, sets the key value, ACL and App data.
- Erases keys, Creates KM0 and KML.
- Uses module key, logical token, or archiving key, creates a new key object.

**Key types**

- [AES or Triple DES]
- Level 2 mode - Any key type
- Level 3 mode - RSA, DSA, Diffie-Hellman, ECDSA or ECDH public keys
- [DSA]
<table>
<thead>
<tr>
<th>Command / Service</th>
<th>Role</th>
<th>Description</th>
<th>Key/CSP access</th>
<th>Key types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Buffer</td>
<td>Unaut, JSO / User</td>
<td>Loads signed data into a buffer. 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. Requires the handle of a buffer created by <code>CreateBuffer</code>. On a MiniHSM this data can be read from flash rather than over the serial interface.</td>
<td>No access to keys or CSPs</td>
<td></td>
</tr>
<tr>
<td>Load Logical Token</td>
<td>monito</td>
<td>Allocates space for a new logical token - the individual shares can then be assembled using <code>ReadShare</code> or <code>ReceiveShare</code>. Once assembled the token can be used in <code>LoadBlob</code> or <code>MakeBlob</code> commands.</td>
<td>Uses module key</td>
<td>[AES or Triple DES]</td>
</tr>
<tr>
<td>Load User Flash</td>
<td>monito</td>
<td>Stores signed data in Flash memory. This data can be retrieved by the Load Buffer service to remove the requirement to repeatedly load the data over the serial interface.</td>
<td>No access to keys or CSPs</td>
<td></td>
</tr>
<tr>
<td>Make Blob</td>
<td>No</td>
<td>Creates a key blob containing the key and returns it. The key object to be exported may be any algorithm.</td>
<td>Uses module key, logical token or archiving key, <em>exports</em> encrypted key object.</td>
<td>AES, Triple DES or RSA and AES</td>
</tr>
<tr>
<td>Mod Exp</td>
<td>yes</td>
<td>Performs a modular exponentiation on values supplied with the command.</td>
<td>No access to keys or CSPs</td>
<td></td>
</tr>
<tr>
<td>Mod Exp CRT</td>
<td>yes</td>
<td>Performs a modular exponentiation on values, supplied with the command using Chinese Remainder Theorem.</td>
<td>No access to keys or CSPs</td>
<td></td>
</tr>
<tr>
<td>Module Info</td>
<td>yes</td>
<td>Reports low level status information about the module. This service is designed for use in nCipher’s test routines.</td>
<td>No access to keys or CSPs</td>
<td></td>
</tr>
<tr>
<td>Command / Service</td>
<td>Role</td>
<td>Description</td>
<td>Key/CSP access</td>
<td>Key types</td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
<td>-------------</td>
<td>----------------</td>
<td>-----------</td>
</tr>
<tr>
<td>NewClient</td>
<td>yes</td>
<td>Returns a client id.</td>
<td>No access to keys or CSPs</td>
<td></td>
</tr>
<tr>
<td>New Enquiry</td>
<td>yes</td>
<td>Reports status information.</td>
<td>No access to keys or CSPs</td>
<td></td>
</tr>
<tr>
<td>No Operation</td>
<td>yes</td>
<td>Does nothing, can be used to determine that the module is responding to commands.</td>
<td>No access to keys or CSPs</td>
<td></td>
</tr>
<tr>
<td>NVMem Allocate</td>
<td>No cert</td>
<td>Allocates an area of non-volatile memory as a file and sets the ACLs for this file. This command can now be used to write files protected by an ACL to a smart card</td>
<td>No access to keys or CSPs</td>
<td></td>
</tr>
<tr>
<td>NVMem Free</td>
<td>No cert</td>
<td>Frees a file stored in non-volatile memory. This command can now be used to write files protected by an ACL to a smart card</td>
<td>No access to keys or CSPs</td>
<td></td>
</tr>
<tr>
<td>NVMem List</td>
<td>yes</td>
<td>Reports a list of files stored in the non-volatile memory. This command can now be used to list files protected by an ACL on a smart card</td>
<td>No access to keys or CSPs</td>
<td></td>
</tr>
<tr>
<td>NVMem Operation</td>
<td>No cert, ACL</td>
<td>Performs an operation on a file stored in non-volatile memory. Operations include: read, write, increment, decrement, etc. This command can now be used to write files protected by an ACL to a smart card</td>
<td>No access to keys or CSPs</td>
<td></td>
</tr>
<tr>
<td>Random Number</td>
<td>yes</td>
<td>Generates a random number for use in an application using the on-board random number generator. There are separate services for generating keys. 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.</td>
<td>No access to keys or CSPs</td>
<td></td>
</tr>
<tr>
<td>Command / Service</td>
<td>Role</td>
<td>Description</td>
<td>Key/CSP access</td>
<td>Key types</td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
<td>-------------</td>
<td>----------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Random Prime</td>
<td>yes</td>
<td>Generates a random prime. This uses the same mechanism as is used for RSA and Diffie-Hellman key generation. The primality checking conforms to ANSI X9.31.</td>
<td>No access to keys or CSPs</td>
<td></td>
</tr>
<tr>
<td>Read File</td>
<td>level 2</td>
<td>Reads a file, but not a logical token, from a smart card or software token. This command can only read files without ACLs.</td>
<td>Reads a file, but not a logical token, from a smart card or software token. This command can only read files without ACLs.</td>
<td></td>
</tr>
<tr>
<td>Read Share</td>
<td>yes</td>
<td>Reads a share from a physical token. Once sufficient shares have been loaded recreates token- may require several ReadShare or ReceiveShare commands.</td>
<td>Uses pass phrase, module key, creates share key, uses share key, creates a logical token.</td>
<td>[SHA-1, AES or Triple DES]</td>
</tr>
<tr>
<td>Receive Share</td>
<td>No</td>
<td>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</td>
<td>Uses an Impath key, uses pass phrase, module key, creates share key, uses share key, creates a logical token</td>
<td>[AES, Triple DES]</td>
</tr>
<tr>
<td>Redeem Ticket</td>
<td>No</td>
<td>Gets a handle in the current name space for the object referred to by a ticket created by GetTicket.</td>
<td>Uses a key object, logical token, Impath, or SEEWorld</td>
<td></td>
</tr>
<tr>
<td>Remove KM</td>
<td>No</td>
<td>Removes a loaded module key.</td>
<td>Erases a module key</td>
<td></td>
</tr>
<tr>
<td>Command / Service</td>
<td>Role</td>
<td>Description</td>
<td>Key/CSP access</td>
<td>Key types</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>SEE Job</td>
<td>No</td>
<td>Sends a command to a SEE World.</td>
<td>No access to keys or CSPs</td>
<td></td>
</tr>
<tr>
<td>Set ACL</td>
<td>No</td>
<td>Sets the ACL for an existing key. The existing ACL for the key must allow the operation.</td>
<td>Sets the Access Control List for a key object</td>
<td></td>
</tr>
<tr>
<td>Set Application Data</td>
<td>No</td>
<td>Stores information with a key.</td>
<td>Sets the application data stored with a key object</td>
<td></td>
</tr>
<tr>
<td>Set KM</td>
<td>No</td>
<td>Loads a key object as a module key.</td>
<td>Uses a key object, sets a module key</td>
<td>AES, Triple DES</td>
</tr>
<tr>
<td>Set NSO Perm</td>
<td>init</td>
<td>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.</td>
<td>Sets the nCipher Security officer’s key hash.</td>
<td>[SHA-1 hash of DSA key]</td>
</tr>
<tr>
<td>Set RTC</td>
<td>No</td>
<td>Sets the real-time clock.</td>
<td>No access to keys or CSPs</td>
<td></td>
</tr>
<tr>
<td>Set SEE Machine</td>
<td>No</td>
<td>Loads the contents of a buffer (created by CreateBuffer / LoadBuffer) as the SEE machine for this module. This command checks the signatures on the buffer.</td>
<td>Uses a public key provided in buffer</td>
<td>DSA, Triple DES MAC, HMAC</td>
</tr>
<tr>
<td>Sign</td>
<td>No</td>
<td>Returns the digital signature or MAC of plain text using a stored key.</td>
<td>Uses a key object</td>
<td>RSA, DSA, ECDSA, Triple DES MAC, HMAC, KCDSA*</td>
</tr>
<tr>
<td>Sign Module State</td>
<td>No</td>
<td>Signs a certificate describing the modules security policy, as set by SetNSOPerm</td>
<td>Uses the module signing key</td>
<td></td>
</tr>
<tr>
<td>Send Share</td>
<td>No</td>
<td>Reads a logical token share and encrypts it under an impath key for transfer to another module where it can be loaded using ReceiveShare</td>
<td>Uses an Impath key, exports encrypted share.</td>
<td>[AES, Triple DES]</td>
</tr>
<tr>
<td>Command / Service</td>
<td>Role</td>
<td>Description</td>
<td>Key/CSP access</td>
<td>Key types</td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
<td>-------------</td>
<td>----------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Statistics / Enumerate Tree</td>
<td>Unauth yes</td>
<td>yes yes</td>
<td>No access to keys or CSPs</td>
<td></td>
</tr>
<tr>
<td>Statistic Get Value</td>
<td>yes yes yes</td>
<td>Yes Reports a particular statistic. No access to keys or CSPs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trace SEE World</td>
<td>No cert yes</td>
<td>Yes Reports debugging output from a SEE World. No access to keys or CSPs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Update Firmware Service (Calls Programming Begin Programming Begin Chunk Programming Load Block Programming End Chunk Programming End)</td>
<td>monitor monitor monitor</td>
<td>These commands are used in the update firmware service. The individual commands are required to load the candidate firmware image in sections small enough to be transported by the interface. nCipher supply the LoadROM utility for the administrator to use for this service. This utility issues the correct command sequence to load the new firmware. The module will only be operating in a FIPS approved mode if you install firmware that has been validated by NIST / CSE. Administrators who require FIPS validation should only upgrade firmware after NIST / CSE issue a new certificate. The monitor also checks that the Version Sequence Number (VSN) of the firmware is as high or higher than the VSN of the firmware currently installed. Uses Firmware Integrity Key and Firmware Confidentiality Keys.</td>
<td>[DSA, Triple DES]</td>
<td></td>
</tr>
<tr>
<td>Verify</td>
<td>No handle, ACL handle, ACL</td>
<td>Verifies a digital signature using a stored key. Uses a key object.</td>
<td>RSA, DSA, ECDSA, Triple DES MAC, HMAC, KCDSA*</td>
<td></td>
</tr>
<tr>
<td>Write File</td>
<td>level 2 cert yes</td>
<td>Writes a file, but not a logical token, to a smart card or software token. Note these files do not have an ACL, use the NVMEM commands to create files with an ACL.</td>
<td>No access to keys or CSPs</td>
<td></td>
</tr>
</tbody>
</table>
## Services available to each role

<table>
<thead>
<tr>
<th>Command / Service</th>
<th>Role</th>
<th>Description</th>
<th>Key/CSP access</th>
<th>Key types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write Share</td>
<td>Unauth</td>
<td>No cert handle</td>
<td>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</td>
<td>[AES, Triple DES, SHA-1]</td>
</tr>
</tbody>
</table>

### Code Description

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>The operator can not perform this service in this role.</td>
</tr>
<tr>
<td>yes</td>
<td>The operator can perform this service in this role without further authorization.</td>
</tr>
</tbody>
</table>
| handle | The operator can perform this service if they possess a valid handle for the resource: key, channel, impath, token, buffers, **SEEWorld**.  
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 or **SEEWorld**. |
| ACL  | 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.  
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 | An operator 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   | An operator can only perform this service if they are in possession of a certificate from the nCipher Security Officer. This certificate will reference a key. 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

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
</table>
| level 2 only | These services are available to the unauthenticated operator only when the module is initialized in it FIPS 140-2 level 2 mode. 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. If this flag is set:  
  • the `Generate Key`, `Generate Key Pair` and `Import` commands require authorization with a certificate signed by the nCipher Security Officer.  
  • the `Import` command fails if you attempt to import a key of a type that can be used to `Sign` or `Decrypt` messages.  
  • the `GenerateKey`, `GenerateKeyPair`, `Import` and `DeriveKey` 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. |
| monitor | These services are used to reprogram the module. They are only available when the module is in the monitor mode. To put the module into monitor mode you must have physical access to the module and put the mode switch into the monitor setting. In order to restore the module to operational mode you reinitialize the module and then return it to operational state. |
**Keys**

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

The nShield and nCipher modules refer 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 FRAM.

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 signed by the NSO and the certificate signed 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 nShield and nCipher modules store a 160 bit random number in the FRAM. 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 GetLongTermKey.

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

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 nShield and nCipher 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 FRAM 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 AES or Triple DES used to protect tokens. The nShield and nCipher 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. $K_{M0}$ is an AES key. 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 FRAM.

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 an AES or Triple DES key used to protect key blobs. Logical tokens are associated with module keys. The key type depends on the key type of the module key.
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 Threshold 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.

Logical tokens are not used for key establishment.

**Share Key**

A share key is used to protect a logical token share when they are written to a smart card or software token that is used for authentication. The share key is created by creating a message comprised of an nCipher secret prefix, Module key, Share number, smart card unique id and an optional 20 bytes supplied by the operator (expected to be the SHA-1 hash of a pass phrase entered into the application), and using this as the input to the approved pRNG function to form a unique key used to encrypt the share - this is either an AES or Triple DES key depending on the key type of the logical token which is itself determined by the key type of the module key. This key is not stored on the module. It is recalculated every time share is loaded. The share data includes a MAC, if the MAC does not verify correctly the share is rejected.

The share key is not used directly to protect CSPs. The logical token needs to be reassembled from the shares using Shamir Threshold Sharing Scheme and then be decrypted using the module key. Only then can the logical token be used to decrypt application keys.

**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-Hellman.
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, and 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 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 a logical token. Key objects may also be exported as key blobs under an archiving key. The key blob can be stored on the host disk or in the module NVRAM.

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. An operator may pass a key to another operator - or to a SEE World - 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.

**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.
- An integrated encryption scheme using Diffie-Hellman or RSA and AES or Triple DES.

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 operator presents a certificate signed by the nCipher Master Feature Enable Key. This causes the module to set the appropriate bit in the FRAM.

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

Communication with the nShield and nCipher 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 operator 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 32-bit arbitrary number.

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 12. An operator cannot access any service that accesses CSPs without first presenting a smart card, or software token.

The nShield and nCipher modules performs identity based authentication. Each individual operator is given a smart card that holds their authentication data - the logical token share - in an encrypted form. All operations require the operator to first load the logical token from their smart card.

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 operator 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 an operator 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 FRAM 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 operator must reauthorize the key by reloading the token.
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 ClientIDs.

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. 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 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 list Operator Defined actions. These actions do not permit any operations within the module, but can be tested with the CheckUserAction service. This enables SEE programs to make use of the ACL system for their own purposes. For example payShield uses this feature to determine the role of a Triple-DES key within EMV.

An ACL can also specify a host service identifier. In which case the ACL is only met if the nCipher server 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.

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 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 change in voltage on the LED pin causing the LED connected to this pin to flash 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 physical security boundary is the plastic jig that contains the potting on both sides of the PCB.

All cryptographic components are covered by potting.

There is also a logical security boundary between the nCore kernel and the SEE.

Some items are excluded from FIPS 140-2 validation as they are not security relevant see Excluded Components on page 9

Status information

The module has an LED pin that indicates the overall state of the module. This pin must be connected to an external LED.

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. Where the module is fitted inside an NetHSM this information can be displayed on the LCD on the NetHSM's front panel.

Procedures to initialise a module to comply with FIPS 140-2 Level 3

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

1 Put the mode switch into the initialisation position and restart the module
2 Use either the graphical user interface KeySafe or the command line tool new-world. Using either tool you must specify the number of cards in the Administrator Card Set and the encryption algorithm to use, Triple-DES or AES. To ensure that the module is in Level 3 mode you must

- Using Keysafe select the option "**Strict FIPS 140 Mode**" = Yes.
- Using new-world specify the -F flag in the command line

3 The tool prompts you to insert cards and to enter a pass phrase for each card.

4 When you have created all the cards, reset the mode switch into the operational position and restart the module.

If a module is initialised in level 3 mode

- Keysafe displays **Strict FIPS 140-2 Mode** = Yes in the information panel for that module.
- The command line tool Enquiry include **StrictFIPS** in the list of flags for the module

### Procedures to initialise a module to comply with FIPS 140-2 Level 2

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

1 Put the mode switch into the initialisation position and restart the module

2 Use either the graphical user interface KeySafe or the command line tool new-world. Using either tool you must specify the number of cards in the Administrator Card Set and the encryption algorithm to use, Triple-DES or AES. To ensure that the module is in Level 2 mode you must

- Using Keysafe select the option "**Strict FIPS 140 Mode**" = **No**
- Using new-world do NOT specify the -F flag in the command line

3 The tool prompts you to insert cards and to enter a pass phrase for each card.

4 When you have created all the cards, reset the mode switch into the operational position and restart the module.

If a module is initialised in level 2 mode

- Keysafe displays **Strict FIPS 140-2 Mode** = **No** in the information panel for that module.
• The command line tool Enquiry does NOT include StrictFIPS in the list of flags for the module

**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.

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

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

**To create a new operator**

1. Create a logical token.
2. Write one or more shares of this token onto software tokens.
3 For each key the operator will require, export the key as a key blob under this token.

4 Give the operator any pass phrases used and the key blob.

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

To authorize the operator to create keys

1 Create a new key, with an ACL that only permits UseAsSigningKey.

2 Export this key as a key blob under the operator's token.

3 Create a certificate signed by the nCipher Security Officer's key that:

4 includes the hash of this key as the certifier

5 authorizes the action GenerateKey or GenerateKeyPair depending on the type of key required.

6 if the operator needs to store the keys, enables the action MakeBlob, limited to their token.

7 Give the operator the key blob and certificate.

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

To authorize an operator 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,

4 Give the private half an ACL that permits Sign and UseAsSigningKey.

5 Give the public half an ACL that permits ExportAsPlainText

6 Export the private half of the Junior Security Officer's key as a key blob under this token.

7 Export the public half of the Junior Security Officer’s key as plain text.

8 Create a certificate signed by the nCipher Security Officer's key includes the hash of this key as the certifier

9 authorizes the actions GenerateKey, GenerateKeyPair
10 authorizes the actions \texttt{GenerateLogicalToken}, \texttt{WriteShare} and \texttt{MakeBlob}, these may be limited to a particular module key.

11 Give the Junior Security Officer the software token, any pass phrases used, the key blob and certificate.

nCipher supply a graphical user interface \texttt{KeySafe} and a command line tool \texttt{new-world} that automate these steps.

\textbf{To authenticate an operator to use a stored key}

1 Use the \texttt{LoadLogicalToken} service to create the space for a logical token.

2 Use the \texttt{ReadShare} service to read each share from the software token.

3 Use the \texttt{LoadBlob} service to load the key from the key blob.

4 The operator can now perform the services specified in the ACL for this key.

5 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.

nCipher supply a graphical user interface \texttt{KeySafe} and a command line tool \texttt{new-world} that automate these steps.

\textbf{To authenticate an operator to create a new key}

1 If you have not already loaded your operator token, load it as above.

2 Use the \texttt{LoadBlob} service to load the authorization key from the key blob.

3 Use the \texttt{KeyId} returned to build a signing key certificate.

4 Present this certificate with the certificate supplied by the Security Officer with the \texttt{GenerateKey}, \texttt{GenerateKeyPair} or \texttt{MakeBlob} command.

nCipher supply a graphical user interface \texttt{KeySafe} and a command line tool \texttt{new-world} that automate these steps.
Physical security

All security critical components of the module 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 port provided by the appliance in which the module is integrated 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 a **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 operator 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 $\frac{2^{16}}{2^{64}} = 2^{-48}$ which is significantly less that the required chance of 1 in 1,000,000 ($\approx 2^{-20}$)

**Attacking Tokens**

If an operator 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 operator could load the logical token. The module does not have any idea as to which operator 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**
  
  when loading a share requiring a pass phrase the operator 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 either 168-bit Triple DES keys or 256-bit AES 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 $2^{-168}$ or $2^{-256}$.
Key Blobs

Key blobs are used to protect keys outside the module. There are two formats of blob - indirect and direct. Direct blobs are used for token and module key protected blobs. Indirect blobs are used by the nCipher Security World for key-recovery and for pass-phrase recovery.

Direct blobs use a Integrated encryption scheme, which takes a 168 or 256 bit key and a nonce and uses SHA-1 to derive a triple-DES or AES encryption key, used for encryption and a HMAC key, used for integrity. Indirect key, take the public half of a 1024-bit RSA key and a nonce as the input, and derive the keys from this. Both forms provide at least 80-bits of security.

Impaths

Impaths protect the transfer of encrypted shares between modules.

When negotiating an Impath, the module verifies a 1024-bit DSA signatures to verify the identity of the other module. It then uses 1024-bit Diffie-Hellman key exchange to negotiate a 168-bit triple-DES encryption keys used to protect the channel. This provides a minimum of 80-bits of security for the encrypted channel.

Note  The shares sent over the channel are still encrypted using their share key, decryption only takes place on the receiving module.

KDP key provisioning

The KDP protocol used to transfer keys from a module to a micro HSM uses 1024-bit DSA signatures to identify the end point and a 2048-bit Diffie-Hellman key exchange to negotiate the Triple-DES or AES keys used to encrypt the keys in transit providing a minimum of 100-bits of security for the encrypted channel.

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.

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Use</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Splitting</td>
<td>Splits a symmetric key into separate components for split knowledge key export</td>
<td>Components are raw byte blocks.</td>
</tr>
<tr>
<td>PKCS8 wrapping</td>
<td>Encrypts a key using a pass phrase.</td>
<td>Not available in FIPS 140-2 level 3 mode</td>
</tr>
<tr>
<td>PKCS8 unwrapping</td>
<td>Decrypts a wrapped key using a pass phrase.</td>
<td>Not available in FIPS 140-2 level 3 mode</td>
</tr>
<tr>
<td>SSL3 master key derivation</td>
<td>Setting up a SSL session</td>
<td>Not available in FIPS 140-2 level 3 mode</td>
</tr>
<tr>
<td>TLS master key derivation</td>
<td>Setting up a TLS session</td>
<td>May use any supported encryption mechanism that accepts a byteblock. The operator must ensure that they chose a wrapping key that has an equivalent strength to the key being transported.</td>
</tr>
<tr>
<td>Key Wrapping</td>
<td>Encrypts one key object with another to allow the wrapped key to be exported.</td>
<td></td>
</tr>
</tbody>
</table>

If the module is initialized in its level 3 mode all non-approved key wrapping and key establishment mechanisms are disabled.
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.

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.
- A statistical check on the random number generator
- Known answer and pair-wise consistency checks on all algorithms and pRNG
- Verification of a MAC on EEPROM contents to ensure it is correctly initialized.

This sequence takes a few seconds after which the module enters the Pre-Maintenance, Pre-Initialisation, Uninitialised or Operational state; depending on the position of the mode switch and the validity of the the EEPROM contents.

While it is powered on, the module continuously monitors the temperature recorded by its internal temperature sensor. If the temperature is outside the operational range it enters the error state.

The module also continuously monitors the hardware RNG and the approved SHA-1 based pRNG. 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.

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.

Firmware Load Test

When an administrator loads new firmware, the module reads the candidate image into working memory. It then performs the following tests on the image before it replaces the current application:

- The image contains a valid signature which the module can verify using the Firmware Integrity Key
- The image is encrypted with the Firmware Confidentiality Key stored in the module.
• The Version Security Number for the image is at least as high as the stored value.

Only if all three tests pass is the new firmware written to permanent storage.

Updating the firmware clears the security officer's key and all stored module keys. The module will not re-enter operational mode until the administrator has correctly re-initialized it.
Supported Algorithms

FIPS approved algorithms:

- AES
  Certificate #258
- Triple DES
  Certificate #339
  Double and triple length keys
  Approved for Federal Government Use - Modes are CBC and ECB
- Triple DES MAC
  Triple-DES Certificate #339 vendor affirmed
- DSA
  Certificate #136
- ECDSA
  Certificate #2
- RSA
  Certificate #68
- SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512
  Certificate #333
- HMAC SHA-1, HMAC SHA-224, HMAC SHA-256, HMAC SHA-384 and HMAC SHA-512
  Certificate #68
- Random Number Generator
  (FIPS 186-2 Change Notice 1 SHA-1 and FIPS 186-2 RNG General Purpose RNG)
  Certificate #91
Non-FIPS approved algorithms

Note  Algorithms marked with an asterisk are not approved by NIST. If the module is initialised in its level 3 mode, these algorithms are disabled. If module is initialized in level 2 mode, the algorithms are available. However, if you choose to use them, the module is not operating in FIPS approved mode.

Symmetric

- Arc Four (compatible with RC4)*
- Blowfish*
- CAST 5 (RFC2144)*
- CAST 6 (RFC2612)*
- DES*

Note  Non-compliant due to CAVP DES transition policy.

- Serpent*
- SEED (Korean Data Encryption Standard) - requires Feature Enable activation*
- Twofish*

Asymmetric

- Diffie-Hellman (key agreement, key establishment methodology provides between 80 and 256 bits of encryption strength)
- El Gamal* (encryption using Diffie-Helman keys)
- Elliptic Curve Diffie-Hellman (key agreement, key establishment methodology provides 192-bits of encryption strength);
- RSA (key wrapping, key establishment methodology provides between 80 and 256 bits of encryption strength)
- KCDSA - requires Feature Enable activation*
- EC-MQV (key establishment methodology provides between 80 and 256 of encryption strength)
Hash

- HAS-160 - requires Feature Enable activation
- MD2
- MD5
- RIPEMD 160

MAC

- HMAC (MD2, MD5, RIPEMD160)
- DES MAC

Other

- SSL/TLS master key derivation
- PKCS #8 key wrapping

Note  
*TLS key derivation is approved for use by FIPS 140-2 validated modules - though there is as yet no validation test. MD5 may be used within TLS key derivation.*
nCipher Corporation Ltd.  
Cambridge, UK  
Jupiter House  
Station Road  
Cambridge  
CB1 2JD  
UK  
Tel: +44 (0) 1223 723600  
Fax: +44 (0) 1223 723601  
E-mail: sales@ncipher.com

nCipher Inc.  
Boston Metro Region, USA  
92 Montvale Avenue, Suite 4500  
Stoneham, MA 02180  
USA  
Tel: 800-NCIPHER  
800-6247437  
+1 (781) 994 4000  
Fax: +1 (781) 994 4001  
E-mail: sales@us.ncipher.com

Internet addresses  
Web Site: http://www.ncipher.com/  
Support: http://www.ncipher.com/support/  
Online Documentation: http://active.ncipher.com/documentation/

Note  nCipher also maintain international sales offices. Please contact the UK, or the US, head office for details of your nearest nCipher representative.