nShield Security World

nShield Security World v13.7.3 Key Management Guide

11 April 2025

Table of Contents

1. Introduction

This guide explains how to use the facilities we provide to work with keys. There is often more than one way of performing a particular task. The methods available for working with keys are:

- `generatekey` and related utilities
- The unit front panel (**network-attached HSMs**)

You cannot generate keys from the front panel on the unit. You can generate keys on the client using the methods described in this chapter and view them on the module.

2. Working with keys

2.1. Generating keys

Whenever possible, generate a new key instead of importing an existing key. Because existing keys have been stored in a known format on your hard disk, there is a risk that the existing key has been compromised. Key material can also persist on backup media.

Some applications can generate keys directly.

When you attempt to generate keys for a Security World that complies with FIPS 140 Level 3, you are prompted to insert an Administrator Card or Operator Card.

Use Operator Cards for FIPS authorization. You should only use the Administrator Card Set for setting up new Security Worlds or performing administrative functions.

You may need to specify to the application, the slot you are going to use to insert the card. You need to insert the card only once in a session.

For softcard protected key generation, you must use an Operator Card Set.

Generating a key creates both a key and a certificate request for the following application types:

- `embed` (OpenSSL)
- `kpm`

These requests are generated in PKCS #10 format with base-64 encoding.

2.1.1. Generating keys using the command line

Keys are generated using the command line with the `generatekey` utility. The `--generate` option creates a new key on the host computer that is protected either by the module or by an Operator Card set from the Security World. No key material is stored in an unencrypted form on the host computer.

When you generate a key with `generatekey`, choose a new identifier for the key and use whichever application type is appropriate. The key identifier can only contain digits, lowercase ASCII letters, and hyphens (`-`).

Any uppercase letters you enter in the key identifier are converted to lowercase when the key is generated.

You can use `generatekey` in two ways:

- In interactive mode, by issuing commands without parameters and supplying the required information when prompted by the utility
- In batch mode, by supplying some or all of the required parameters using the command line (`generatekey` prompts interactively for any missing but required parameters).

In interactive mode, you can input `abort` at any prompt to terminate the process.

Batch mode is useful for scripting. In batch mode, if any required parameters are omitted, `generatekey` does not prompt for the missing information but instead will either use available defaults or fail. If you specify one or more parameters incorrectly, an error is displayed and the command fails.

If the Security World was created with audit logging selected then you can request that the usage of a key for cryptographic operations is logged in the audit log. By default only key generation and destruction is logged.

To generate a key, use the command:

```
generatekey --generate [OPTIONS] <APPNAME> [<NAME>=<VALUE> ...]
```

In this command:

- `--generate` option specifies that this instance of `generatekey` is generating a key. Other options can be specified to perform tasks such as importing or retargeting keys. To see a list of options run the command `generatekey --help`.
- the `<APPNAME>` parameter specifies the name of the application for which the key is to be generated. For details of the available application types (`APPNAME`), Key application type (APPNAME).
- The `<NAME>=<VALUE>` syntax is used to specify the properties of the key being generated. For details of the available application types (`APPNAME`), see Key properties (NAME=VALUE).

For details of the available application types (`APPNAME`) and parameters that control other key properties (`NAME=VALUE`), see Key generation options and parameters and parameters.

In interactive mode, `generatekey` prompts you for any required parameters or actions that have not been included in the command. When you give the command:

1. Enter parameters for the command, as requested. If you enter a parameter incorrectly, the request for that information is repeated and you can re-enter the parameter.
2. When all the parameters have been collected, `generatekey` displays the final settings. In a FIPS 140 Level 3 compliant Security World, you are prompted to insert a card for FIPS authorization if no such card is present.
3. If prompted, insert an Administrator Card or an Operator Card from the current Security World.
4. If you want to protect the key with an OCS, you are prompted to insert the relevant cards and input passphrases, as required.

2.1.1.1. Example of key generation with generatekey

To generate a simple RSA key in batch mode, protected by module protection, use the command:

```
generatekey --generate --batch simple type=rsa size=2048 plainname=keya ident=abcd certreq=yes
```

The `generatekey` utility prompts you to insert a quorum of Operator Cards from the `operatorone` OCS. After you have inserted the appropriate number of cards, `generatekey` generates the key.

Although it is not explicitly specified, the created key is recoverable by default if OCS and softcard replacement is enabled for the Security World.

2.1.2. Generating NVRAM-stored keys

NVRAM key storage provides a mechanism for generating keys stored in a module's non-volatile memory and hence within the physical boundary of an nShield module. You can store only a few keys in this way: the number depends on the memory capacity of the module, the size of the key and whether the key has recovery data associated with it.

We recommend that you *do not store keys in NVRAM unless you must do so to satisfy regulatory requirements.* NVRAM key storage was introduced only for users who must store keys within the physical boundary of a module to comply with regulatory requirements. NVRAM-stored keys provide no additional security benefits and their use exposes your ACS to increased risk. Storing keys in nonvolatile memory also reduces load-balancing and recovery capabilities. Because of these factors, we recommend you always use standard Security World keys unless explicitly required to use NVRAM-stored keys.

When you generate an NVRAM-stored key, you must have sufficient nonvolatile memory available in the module or the command fails.

You need backup and recovery procedures, which must be consistent with regulatory requirements, to protect your NVRAM-stored keys. Do *NOT* use Remote Administration to back-up keys to a smart card, as, in transit, the keys would not be physically protected from access by the host system.

An NVRAM-stored key can only be loaded successfully by using the `preload` command-line utility on the generating module. Attempts to load such a key on other modules that have NVRAM fail with `UnknownID` errors.

We provide the `nvram-backup` utility to enable the copying of files, including NVRAM-stored keys, between a module's nonvolatile memory and a smart card.

2.2. Importing keys

Importing a key takes an unprotected key stored on the host and stores it in the Security World in encrypted form.

We recommend generating a new key (or retargeting a key from within the Security World) instead of importing an existing key whenever possible. The import operation does not delete any copies of the key material from the host, and because existing keys have been stored in a known format on your hard disk (and key material can persist on backup media), there is a risk that an existing key has been compromised. It is your responsibility to ensure any unprotected key material is deleted. If a key was compromised before importation, then importing it does not make it secure again.

The following key types can be imported by the tools we provide:

- RSA keys in PEM-encoded PKCS #1 format (from a file). The PEM key that contains the key to import must not require a passphrase.
- EC, ECDH, ECDSA, Ed25519 and X25519 keys in PEM-encoded PKCS #8 format (from a file). The PEM key that contains the key to import must not require a passphrase.
- DES, DES2 and Triple DES keys (entered in hex).

You cannot import keys into a Security World that complies with FIPS

> 140 Level 3. Attempting to import keys into a FIPS 140 Level 3 Security World returns an error.

This request is a PKCS #10 format request in base-64 encoding.

2.2.1. Importing keys from the command line

You can import keys using the `generatekey` utility. To import a key, give the command:

```
generatekey --import [<OPTIONS>] <APPNAME> [<NAME>=<VALUE> ...]
```

This command uses the following options:

Option	Description
`--import`	This option specifies key importation.
`<OPTIONS>`	You can specify particular options when running `generatekey` that control details of key importation.
`<APPNAME>`	This option specifies the name of the application for which the key is to be imported. This must be an application for which `generatekey` can generate keys.
`<NAME>=<VALUE>`	This specifies a list of parameters for the application.

For RSA or elliptic curve keys, you can include `pemreadfile=filename` in the command to specify the file name of the PEM file that contains the key.

Otherwise, you are prompted for this information during import.

In interactive mode, you are prompted for any required parameters or actions that have not been included in the command:

- Enter parameters, as requested. If you enter a parameter incorrectly, the request for that information is repeated and you can re-enter the parameter.
- If you want to protect the key with an OCS, you are prompted to insert the relevant cards and input passphrases, as required.
- If prompted, insert an Administrator Card or an Operator Card from the current Security World.

2.2.1.1. Example of key importation with generatekey

To import an RSA key stored in `/opt/projects/key.pem` (**Linux**) or `C:\projects\key.pem`

(**Windows**) for use with an nShield native application and protect it with the Security World, use the command:

Linux

```
generatekey --generatekey --import simple pemreadfile=/opt/projects/key.pem plainname=importedkey ident=abc
protect=module
```

Windows

```
generatekey --generatekey --import simple pemreadfile=C:\projects\key.pem plainname=importedkey ident=abc
protect=module
```

In this example, `generatekey` requires you to input `RSA` for the key type.

Although not explicitly specified, this key is, by default, recoverable if OCS and softcard replacement is enabled for the Security World.

2.3. Listing supported applications with generatekey

To list supported applications, use the command:

```
generatekey --list-apps
```

2.4. Retargeting keys with generatekey

The `--retarget` option to takes an existing key in the Security World and makes it available for use by another application as if it had been expressly generated for use by that application. Because no key material is exposed during retargeting, this operation is as secure as generating a new key.

When you retarget a key, `generatekey` does not remove the original key from the Security World.

When you retarget a key, you cannot change its protection method. You cannot change the key from module-protected to card-protected, or from card-protected to module-protected.

To retarget a key, use the command:

```
generatekey --retarget [<OPTIONS>] <APPNAME> [from-application=<appname>]
[from-ident=<keyident>]
```

In this command:

Option	Description
`--retarget`	This option specifies key importation.
`<OPTIONS>`	This option specifies any options to include when the command is run. Run the command `generatekey --help` for details about the available options.
`<APPNAME>`	This option specifies the name of the application for which the key is to be generated. This must be an application for which `generatekey` can generate keys.
`from-application=<appname>`	This option specifies the name of the application with which the key is currently associated.
`from-ident=<keyident>`	This option specifies the identifier of the key to be retargeted. You can find this identifier by using the `nfkminfo` command-line utility.

If `generatekey` cannot identify the key type for retargeting, you are prompted to specify the key type. Input the key type and press Enter.

2.5. Viewing keys

You can view existing keys in the Security World using the `nfkminfo` command-line utility or the unit front panel (**network-attached HSMs**).

2.5.1. Viewing information about keys on the unit front panel (network-attached HSMs)

You can view keys that have been created on the client on the same computer as the RFS with SEE machines. You cannot view other keys until they are transferred to the RFS.

To view keys:

1. From the main menu, select **Security World mgmt > Keys > List keys**.
2. Select the application to which the key belongs.
3. Select a key to view its full details.
4. If you wish, select **Verify key ACLs** to verify the key's ACL.

2.5.2. Viewing keys using the command line

The `nfkminfo` command-line utility is used to list keys. To list the keys that have been cre-

ated in the current Security World, use one of the following commands:

```
nfkminfo -k [<APPNAME>[<IDENT>]]
```

```
nfkminfo -l [<APPNAME>[<APPNAME>...]]
```

The `-k`|`--key-list` option lists keys only. The `-l`|`--name-list` option lists keys and their names.

With either option, `<APPNAME>` is an optional application name. If you specify an application name, `nfkminfo` lists only the keys for that application. Commonly, `<APPNAME>` is often one of:

- `custom`
- `embed`
- `pkcs11`
- `kpm`
- `kps`
- `mscapi` (**Windows**)
- `seeconf`
- `seeinteg`
- `simple`

You can also specify your own application names for *APPNAME* as appropriate to your system.

For example, user-defined application names can be created by using the `nfkm` library to generate arbitrary keys.

With the `--name-list` option, `<IDENT>` is the key identifier.

The command `nfkminfo --key-list` returns output of the form:

```
Key summary - 4 keys:
 AppName appname    Ident <ident> AppName <appname>
Ident <ident> AppName <appname>
Ident <ident> AppName <appname>                Ident <ident>
```

To list information about a specific key, specify the `--key-list` option with an application and key identifier:

```
nfkminfo --key-list <appname> <ident>
```

This command returns output of the form:

```
Key AppName <appname> Ident <ident> BlobKA length   752
 BlobPubKA length      316
 BlobRecoveryKA length 868
 name                  "name"
 hash                  hash recovery              Enabled
 protection            CardSet
 other flags           PublicKey +0x0
 cardset               hash_ktBlobKA
 format                6 Token
 other flags           0x0
 hkm                   hash_km hkt                hash_kt hkr       none
BlobRecoveryKA
 format                8 Indirect
 other flags           0x0
 hkm                   none
 hkt                   none
 hkr                   hash_krBlobPubKA
 format                5 Module
 other flags           0x0
 hkm                   hash_km hkt                none
 hkr                   none
No extra entries
```

To list keys and names, specify the `--name-list` option. The command `nfkminfo --name-list` returns output of the form:

```
Key summary - 30 keys
in format key_<appname>_<ident> '<name>')
  key_appname_ident'name '
  key_appname_ident'name '
  key_appname_ident'name '
  key_appname_ident'name '
  key_appname_ident'name '
  key_appname_ident'name '
  key_appname_ident'name '
```

2.6. Verifying Key Generation Certificates with nfkmverify

The `nfkmverify` command-line utility verifies key generation certificates. You can use `nfkmverify` to confirm how a particular Security World and key are protected. It also returns some information about the Security World and key.

The `nfkmverify` utility compares the details in the ACL of the key and those of the card set that currently protects the key.

A key that has been recovered to a different card set shows a discrepancy for every respect that the new card set differs from the old one. For example, a key recovered from a 2-of-1 card set to a 1-of-1 card set has a different card-set hash and a different number of cards, so two discrepancies are reported. The discrepancy is between the card set mentioned in the ACL of the key and the card set by which the key is currently protected (that

is, the card set mentioned in the key blobs).

A key that has been transferred from another Security World shows discrepancies and fails to be verified. Entrust recommends that you verify keys in their original Security World at their time of generation.

If you must replace your Security World or card set, Entrust recommends that you generate new keys whenever possible. If you must transfer a key, perform key verification immediately before transferring the key; it is not always possible to verify a key after transferring it to a new Security World or changing the card set that protects it.

2.6.1. Usage

To verify the key generation certificates from the command line, run the command:

```
nfkmverify [-f|--force] [-v|--verbose] [-U|--unverifiable] [-m|--module=MODULE] [appname ident [appname ident
[...]]]
```

Optionally, the command can also include the following:

Option	Description
`-h|--help`	This option displays help for `nfkmverify`.
`-V|--version`	This option displays the version number for `nfkmverify`.
`-u|--usage`	This option displays a brief usage summary for `nfkmverify`.
`-m|--module=MODULE`	This option performs checks with module `MODULE`.
`-f|--force`	This option forces display of an output report that might be wrong.
`-U|--unverifiable`	This option permits operations to proceed even if the Security World is unverifiable.  If you need the `-U|--unverifiable` option, there may be some serious problems with your Security World.
`-v|--verbose`	This option prints full public keys and generation parameters.
`-C|--certificate`	This option checks the original ACL for the key using the key generation certificate. This is the default.
`-L|--loaded`	These options check the ACL of a loaded key instead of the generation certificate.

Option	Description
`-R|--recov`	This option checks the ACL of the key loaded from the recovery blob.
`--allow-dh-unknown-sg-group`	This option allows an operation to proceed even if a Diffie-Hellman key is using an unrecognized Sophie-Germain group.
`-A|--assigned`	**nShield Solo XC** In a `common-criteria-cmts` Security World `nfkmverify` will identify keys as Assigned or General, see Common Criteria CMTS Mode Assigned Keys (nShield Solo XC) based on the criteria in the *nShield Solo XC Common Criteria Evaluated Configuration Guide*, and print the classification by default. When considering the key's timeout and usage limits `nfkmverify` will consider these limits against the max-keyusage and max-keytimeout values set on a `common-criteria-cmts` Security World. If there is a maximum value set on the Security World, any non-zero value less than or equal to this is considered compatible with the reauthorization conditions for an Assigned Key. If the maximum value is not set on the Security World, no value or any value is considered compatible with the reauthorization conditions for an Assigned Key. This option, in a `common-criteria-cmts` mode Security World, means the `nfkmverify` utility will exit with a non-zero exit code if the key is not an Assigned Key. This supports testing for Assigned Keys programmatically.

2.7. Discarding keys

If you have copies of the Security World data on several computers, erasing the data on one computer does not remove it from any other computer.

To destroy a key permanently you must either erase the OCS that is used to protect it or erase the Security World completely. There are no other ways to destroy a key permanently.

2.8. Restoring keys

We do not supply tools for restoring a key that has been discarded. However if you have kept a backup of the host data for the Security World, you can restore a key from the backup data.

If you have NVRAM-stored keys, you must additionally ensure you have a backup of the key data stored on the relevant modules.

3. Key generation options and parameters

This appendix describes the various options and parameters that you can set when running the `generatekey` utility to control the application type and other properties of a key being generated.

For information about generating keys with the `generatekey` utility, see Generating keys with the command line.

3.1. Key application type (APPNAME)

The `APPNAME` parameter specifies the name of the application for which `generatekey` can generate keys. Specifying an application can restrict your choice of key type. A value for `APPNAME` must follow any `OPTIONS` and must precede any parameters specified for the key:

Parameter	Description
`simple`	Specifying the `simple` application type generates an nShield-native key. No special action is taken after the key is generated.
`custom`	Specifying the `custom` application type generates a key for custom applications that require the key blob to be saved in a separate file. Specifying `custom` also causes the generation of a certificate request and self-signed certificate. However, we recommend that you specify the `simple` (instead of `custom`) application type whenever possible.
`pkcs11`	Specifying the `pkcs11` application type generates keys that are formatted for use with PKCS #11 applications and are given a suitable identifier. The set of possible supported key types is currently limited to: • `DES3` • `DH` • `DSA` • `ECDH` • `ECDSA` • `Ed25519` • `HMACSHA1` • `RSA` • `Rijndael` (`AES`) • `X25519` Some key types are only available if the features that support them have been enabled for the module, if the Security World is not compliant with FIPS 140 Level 3, or if you do not set the `--no-verify` option.

Parameter	Description
embed	Specifying the embed application type generates a PEM-format RSA/DSA key file that points to a key in NFAST_KMDATA so a software application can then use the HSM-protected key. In applications that use Security World software older than v12.60 and would use the legacy OpenSSL CHIL engine with hwcrhk: • The plainname specified in the generatekey command is used as the prefix for all 3 generated files (.key, _req, _selfcert) • .key is appended to all 3 files • The embedsavefile specified in the generatekey command is the destination for all 3 files In applications that use v12.60 or later Security World software : • The plainname specified in the generatekey command is used as the prefix for only the .key file, the prefix for the _req and _selfcert file is embed<hash> • .key is not appended to the _req and _selfcert files • The embedsavefile is the destination only for the .key file, _req and _selfcert are created in the directory from which generatekey was run from
kpm	Specifying the kpm application type generates a key for delivery by an nForce Ultra key server. The generatekey utility automatically creates a special ACL entry that permits a kpm to be delivered to an nForce Ultra's enrolled internal hardware security module.
seeinteg	Specifying the seeinteg application type generates an SEE integrity key. The DSA, RSA, ECDSA and KCDSA algorithms are supported. SEE integrity keys are always protected by an OCS and cannot be imported. You cannot retarget an existing key as an SEE integrity key.
seeconf	Specifying the seeconf application type generates an SEE confidentiality key. Both the Triple DES and AES algorithms are supported for this key type. SEE confidentiality keys are module-protected by default and cannot be imported. You cannot retarget an existing key as an SEE confidentiality key.

3.2. Key properties (NAME=VALUE)

The NAME=VALUE syntax is used to specify the properties of the key being generated.

If a parameter's argument contains spaces, you must enclose the argument within quotation marks (" ").

You can supply an appropriate *VALUE* for the following *NAME* options:

Option	Description
`alias`	The *VALUE* for `alias` specifies an alias to assign to the key.
`assigned` (*nShield Solo XC)	The *VALUE* for `assigned` specifies if the generated key is to be Assigned as defined by *nShield Solo XC Common Criteria Evaluated Configuration Guide*. This is only relevant in common-criteria-cmts mode Security Worlds and the key must be protected with a non-recoverable softcard or token. If set to **yes** the ACL of the generated key will match the definition of an Assigned key in *nShield Solo XC Common Criteria Evaluated Configuration Guide* and will be verified as an Assigned key by `nfkmverify`. The default is **no**.
`blobsavefile`	When using the `custom` application type, the *VALUE* for `blobsavefile` specifies a file name of the form *FILENAME*`_req.`*ext* to which the key blob is saved. Additionally, a text file containing information about the key is saved to a file whose name has the form *ROOT*`_inf.txt`; for asymmetric key types, the public key blob is also saved to a file whose name has the form *ROOT*`_pub.`*EXT*.
`cardset`	The *VALUE* for `cardset` specifies an OCS that is to protect the key (if `protect` is set to `token`). In interactive mode, if you do not specify an OCS, you are prompted to select one at card-loading time. The default is the OCS to which the card currently inserted in the slot belongs (or the first one returned by `nfkminfo`).
`certreq`	Setting `certreq` enables you to generate a certificate request when generating a PKCS #11 key (RSA keys only). The default behavior is to not generate a certificate request. To generate a certificate request you must set the *VALUE* for `certreq` to `yes`, which makes `generatekey` prompt you to fill in the extra fields required to generate a key with a certificate request. The resultant certificate request is saved to the current working directory with a file name of the form *FILENAME* `req.ext` (where *FILENAME* is a name of your choice). An extra file with a name of the form *FILENAME*`.ext` is also generated for use as a pseudo-key-header. This file can be removed after the certificate request has been generated. You can use `certreq` with the `--retarget` option to generate a self-signed certificate for an existing key.
`checks`	For RSA key generation only, this specifies the number of checks to be performed. Normally, you should leave *VALUE* empty to let the module pick an appropriate default.
`curve`	For ECDH and ECDSA key generation only, the *VALUE* for `curve` specifies which curves from the supported range to use. Supported curves are: ANSIB163v1, ANSIB191v1,BrainpoolP160r1, BrainpoolP160t1, BrainpoolP192r1, BrainpoolP192t1, BrainpoolP224r1, BrainpoolP224t1, BrainpoolP256r1, BrainpoolP256t1, BrainpoolP320r1, BrainpoolP320t1, BrainpoolP384r1, BrainpoolP384t1, BrainpoolP512r1, BrainpoolP512t1, NISTP192, NISTP224, NISTP256, NISTP384, NISTP521, NISTB163, NISTB233, NISTB283, NISTB409, NISTB571, NISTK163, NISTK233, NISTK283, NISTK409, NISTK571, SECP160r1 and SECP256k1

Option	Description
`embedconvfile`	The *VALUE* for `embedconvfile` specifies the name of the PEM file that contains the RSA key to be converted.
`embedsavefile`	When using the `embed` application type, the `VALUE` for `embedsavefile` specifies the name for the file where the fake RSA private key is to be saved. The file has the same syntax as an RSA private key file, but actually contains the key identifier rather than the key itself, which remains protected. A certificate request and a self-signed certificate are also written. If the file-name is *ROOT*`.EXT` then the request is saved to *ROOT*`_req.EXT` and the self-signed certificate is saved to *ROOT*`_selfcert.EXT`.
`from-application`	When retargeting a key, the *VALUE* for `from-application` specifies the applica tion name of the key to be retargeted. Only applications for which at least one key exists are acceptable.
`from-ident`	When retargeting a key, the *VALUE* for `from-ident` specifies the identifier of the key to be retargeted (as displayed by the `nfkminfo` command-line utility).
`hexdata`	The *VALUE* for `hexdata` specifies the hex value of DES or Triple DES key to import. The hex digits are echoed to the screen and can appear in process list-ings if this parameter is specified in the command line.
`ident`	The *VALUE* for `ident` specifies a unique identifier for the key in the Security World. For applications of types `simple`, this is the key identifier to use. For other application types, keys are assigned an automatically generated identi-fier and accessed by means of some application-specific name. The following characters are allowed in key IDs: • digits 0-9 • lower-case letters a-z • hyphen (-)
`keystore`	The *VALUE* for `keystore` specifies the file name of the key store to use. This must be an nShield key store.
`keystorepass`	The *VALUE* for `keystorepass` specifies the password to the key store to use.
`logkeyusage`	The *VALUE* for `logkeyusage` specifies if usage of the generated key in crypto-graphic operations is subject to audit logging. If set to **yes** the ACL of the gen erated key will predicate audit-logging entries to be made for cryptographic usages of the key. The default is **no**.
`module`	The *VALUE* for `module` specifies a module to use when generating the key. If there is more than one usable module, you are prompted to supply a value for one of them. The default is the first usable module (one in the current Security World and in the operational state).  You can also specify a module by setting the `--module` option.

Option	Description
paramsreadfile	The *VALUE* for paramsreadfile specifies the name of the group parameters file that contains the discrete log group parameters for Diffie-Hellman keys only. This should be a PEM-formatted PKCS#3 file. If a *VALUE* for paramsreadfile is not specified, the module uses a default file.
pemreadfile	The <*VALUE*> for pemreadfile specifies the name of the PEM file that contains the key to be imported. When importing an RSA key, this is the name of the PEM-encoded PKCS #1 file to read it from. When importing an EC, ECDH, ECDSA, Ed25519 or X25519 key, this is the name of the PEM-encoded PKCS #8 file to read it from. Password-protected PEM files are not supported.
plainname	The *VALUE* for plainname specifies the key name within the Security World. For some applications, the key identifier is derived from the name, but for others the name is just recorded in kmdata (**Linux**) or %NFAST_KMDATA% (**Windows**) and not used otherwise.
protect	The *VALUE* for protect specifies the protection method, which can be module for security-world protection, softcard for softcard protection or token for Operator Card Set protection. The default is token, except for seeconf keys, where the default is module. seeinteg keys are always token-protected. The softcard option is only available when your system has at least one softcard present.
pubexp	For RSA key generation only, the *VALUE* for pubexp specifies (in hexadecimal format) the public exponent to use when generating RSA keys. We recommend leaving this parameter blank unless advised to supply a particular value by Support.
recovery	The *VALUE* for recovery enables recovery for this key and is only available for card-set protected keys in a recovery-enabled world. If set to yes, the key is recoverable. If set to no, key is not recoverable. The default is yes. Non-recoverable module-protected keys are not supported.
seeintegname	If present, the *VALUE* for seeintegname identifies a seeinteg key. The ACL of the newly generated private key is modified to require a certificate from the seeinteg key for its main operational permissions, such Decrypt and Sign (DuplicateHandle, ReduceACL, and GetACL are still permitted without certification.) If you use seeintegname to specify a key that has been recovered with the rocs utility, you must also use the -N option with generatekey.
selfcert	The *VALUE* for selfcert enables you to generate a self-signed certificate when generating a PKCS #11 key (RSA keys only). To generate a self-signed certificate request you must set selfcert to yes, which makes generatekey prompt you to fill in the extra fields required to generate a key with a self-signed certificate. The resultant certificate is saved to the current working directory with a file name of the form *FILENAME*.ext. You can use this parameter with the --retarget option to generated a self-signed certificate for an existing key.

Option	Description
`size`	For key types with variable-sized keys, the *VALUE* for `size` specifies the key size in bits. The range of allowable sizes depends on the key type and whether the `--no-verify` option is used. The default depends on the key type; for infor mation on available key types and sizes, see Cryptographic algorithms. This parameter does not exist for fixed-size keys, nor for ECDH and ECDSA keys which are specified using `curve`.
`strict`	For DSA key generation only, setting the *VALUE* for `strict` to `yes` enables strict verification, which also limits the size to 2048 or 3072 bits. The default is `no`.
`type`	The *VALUE* for `type` specifies the type of key. You must usually specify the key type for generation and import (though some applications only support one key type, in which case you are not asked to choose). Sometimes the type must also be specified for retargeting; for information on available key types and sizes, see Cryptographic algorithms. The `--verify` option limits the avail- able key types.
`x509country`	The *VALUE* for `x509country` specifies a country code, which must be a valid 2-letter code, for the certificate request.
`x509dnscommon`	The *VALUE* for `x509dnscommon` specifies a site domain name, which can be any valid domain name, for the certificate request.
`x509email`	The *VALUE* for `x509email` specifies an email address for the certificate request.
`x509locality`	The *VALUE* for `x509locality` specifies a city or locality for the certificate request.
`x509org`	The *VALUE* for `x509org` specifies an organization for the certificate request.
`x509orgunit`	The *VALUE* for `x509orgunit` specifies an organizational unit for the certificate request.
`x509province`	The *VALUE* for `x509province` specifies a province for the certificate request.
`xsize`	The *VALUE* for `xsize` specifies the private key size in bits when generating Diffie-Hellman keys. The defaults are 256 bits for a key size of 1500 bits or more or 160 bits for other key sizes.

3.3. Available key properties by action/application

The following table shows which actions (generate, import, and retarget) are applicable to the different *NAME* options:

Property	generate	import	retarget
`alias`	X	X	X

Property	generate	import	retarget
blobsavefile	X	X	X
cardset	X	X	
certreq			
checks	X		
curve	X		
embedconvfile		X	
embedsavefile	X	X	X
from-application			X
from-ident			X
hexdata		X	
ident	X	X	
keystore	X	X	X
keystorepass	X	X	X
module	X	X	
nvram	X	X	
paramsreadfile	X		
pemreadfile		X	
plainname	X	X	X
protect	X	X	
pubexp	X		
qsize	X		
recovery	X	X	
seeintegname			
selfcert			
size	X		
strict	X		
type	X		
x509country	X	X	X
x509dnscommon	X	X	X

Property	generate	import	retarget
x509email	X	X	X
x509locality	X	X	X
x509org	X	X	X
x509orgunit	X	X	X
x509province	X	X	X
xsize	X		

The following table shows which applications are applicable to the different *NAME* options:

Property	custom	embed	hwcrhk	pkcs 11	seeconf	seeinteg	seessl	simple	kpm
alias									
blobsavefile	X								
cardset	X	X	X	X				X	X
certreq				X					
checks	X	X	X	X				X	X
curve	X	X	X	X	X	X		X	
embedconvfile		X							
embedsavefile		X		X					
from-application	X	X	X	X				X	X
from-ident	X	X	X	X				X	X
hexdata	X	X	X	X				X	
ident			X					X	X
keystore									
keystorepass									
module	X	X	X	X			X	X	X
nvram	X	X	X	X				X	
paramsreadfile	X	X	X	X	X	X		X	
pemreadfile	X		X					X	X
plainname	X	X		X	X	X	X	X	X
protect	X	X	X	X	X	X	X	X	X

Property	custom	embed	hwcrhk	pkcs 11	seeconf	seeinteg	seessl	simple	kpm
pubexp	X	X	X	X				X	X
qsize	X	X	X	X				X	X
recovery	X	X	X	X	X	X		X	X
seeintegname	X						X	X	
selfcert				X					
size	X	X	X	X	X	X	X	X	X
strict	X	X	X	X				X	
type	X	X	X	X	X	X	X	X	X
x509country		X							X
x509dnscommon		X							X
x509email		X							X
x509locality		X							X
x509org		X							X
x509orgunit		X							X
x509province		X							X
xsize	X	X	X	X				X	

4. Key migration

The current version of Security World software enables you to create a Security World that fully complies with the NIST Recommendations for the Transitioning of Cryptographic Algorithms and Key Sizes (SP800-131Ar1) or alternatively Common Criteria PP 419 221-5 (common-criteria-cmts) depending on the options selected at World creation. This is called World version 3.

We recommend that where compliance with the specifications above is required, you create a new World and create new keys within that World. However, the software also includes a `migrate-world` command-line utility that you can use for migrating existing keys into the new World. This is provided as a convenience for customers who require compliance with the specifications, and who need to continue using existing keys.

In the case of a Common Criteria World the specification prohibits the importing of assigned keys. Only general keys can be imported into a common-criteria-cmts World.

Throughout the following sections, the terms `Source World` refers to the World from which you want to migrate keys, and `Destination World` refers to the World to which you want to migrate keys.

The utility requires the use of two modules. One module is referred to as the source module. The other module is referred to as the destination module.

4.1. Pre-requisites for migrating keys

In order to use the `migrate-world` utility the following will be needed:

- Two HSMs. These can be any of the currently supported HSM types and the two HSMs do not need to be of the same type.
- A quorum of ACS cards for the source World.
- A quorum of ACS cards for the destination World.
- Sufficient blank cards to create new OCS cards for any keys that are OCS protected.
- (**nShield Solo, Solo XC, and Connect**) Remote mode switching must be enabled on both HSMs used for the migration.
  See `enable_remote_mode` in the server_settings section or Top-level menu

4.2. Restrictions on migrating keys

The following restrictions apply to the use of `migrate-world`:

- The source module must be running firmware version 12.50 or later.
- The destination module must be running firmware version 12.50 or later.
- Only recoverable keys can be migrated. If your source keys are non-recoverable, you cannot use the migration utility to migrate keys.
- You can change some, but not all, Security World properties during migration:

Property	Up to 13.3	13.4 and later
Key protection method whether softcard or OCS is used	Fixed	Fixed
Key protection name softcard name or cardset name	Fixed	Editable
Quorum	Editable	Editable

  If the name or quorum is to be changed, you must create the softcard or OCS in the destination world before migration begins.

- Replacement cards should be of the same or newer generation than the cards that they replace.
- The source and destination modules must both have KLF2 warrants in the correct location.

  The migration process directly downloads the warrants from the module for the nShield 5s and nShield 5c HSMs. You do not need to take any action.

  For pre-nShield5 HSMs:

  - If one or both of the modules have a KLF warrant, you must request an upgrade to a KLF2 warrant from nshield.support@entrust.com before you start the migration.
  - For Solo + and Solo XC, the default location is `NFAST_KMDATA/warrants/` (**Linux**) or `NFAST_KMDATA\warrants\` (**Windows**).
  - For Connect + and Connect XC, the default location is `NFAST_KMDATA/hsm-<ESN>/warrants/` (**Linux**) or `NFAST_KMDATA\hsm-<ESN>\warrants\` (**Windows**).
  - After adding or upgrading to a KLF2 warrant, you must reboot the HSM before the warrant file will appear in the warrants directory.
- The operator running the migrate-world utility must have the access rights to create a privileged connection to the hardserver.
- The migration tool must have exclusive use of the modules during migration. Do not use them for any other purpose during migration and if either module is an nShield net-

work-attached HSM, do not enter anything via the front panel during migration.

If the destination World is `fips-140-level-3`, then some keys that were usable in the source World may not be usable in the destination World due to those algorithms or key lengths being restricted. The migration tool might not be able to successfully migrate these keys so they should be removed from the source World before attempting the migration. Any keys of this type that do migrate successfully will be restricted at the point of use.

If the destination World is `fips-140-level-3` or `common-criteria-cmts` the migration tool will automatically remove ExportAsPlain from the ACL of any migrated key during the migration process.

If the destination world does not support audit logging the migration tool will automatically remove LogKeyUsage from the ACL of any migrated key during the migration process.

4.3. About the migration utility

You can run the migration utility in the following modes:

- **Plan mode**: Returns a list of steps for migration and the required card sets and passphrases but does not migrate any keys.
- **Perform mode**: Runs the plan mode prior to presenting the option to proceed and migrate keys according to the plan.

4.3.1. Usage and options

```
migrate-world [OPTIONS] --src-module=<source_module> --dst-module=<dest_module> --source=<source-kmdata-path>
--debug --dst-warrant=<dst-warrant-path> --src-warrant=<src-warrant-path [--plan | --perform] --key-logging
```

Option	Enables you to...
`-k <KEYS>|--keys-at-once=<KEYS>`	Migrate no more than this number of keys per ACS loading. This is useful to prevent ACS time-outs if you have a large number of keys to migrate. (0=unlimited, default=0). It is recommended to limit the number of keys to be migrated at any one time to no more than 100.
`-h|--help`	Obtain information about the options you can use with the utility.

Option	Enables you to...
`-c <CARDSETS>|--cardsets-at-once=<CARDSETS>`	Migrate keys protected by this number of card sets or softcards per ACS loading. This is useful to prevent ACS time-outs if you have a large number of different card sets or softcards to migrate. (0=unlimited, default=0).
`--version`	View the version number of the utility.
`--src-warrant=<src-warrantfile>`	Specify the location of the warrant file of the source module.
`--src-module=<MODULE>`	Specify which module ID to use as the source module.
`--source=<SOURCE>`	Specify the path to the folder that contains the source World data.
`--plan`	View the list of steps that will be carried out.
`--perform`	Migrate keys interactively.
`--dst-warrant=<dst-warrantfile>`	Specify the location of the warrant file of the destination module.
`--dst-module=<ModuleId>`	Specify which module ID to use as the destination module.
`--debug`	Outputs debug messages and stack traces in case of errors. It is recommended to use this only for testing as it will slow down operation and make card timeouts more likely to occur. A large volume of output is produced for each key that is migrated, so it is recommended to migrate a single key at a time when using this option.
`--key-logging`	This option will enable key usage logging on all migrated keys. If the destination World does not support audit logging the keys will still be migrated but LogKeyUsage logging will not be set in the ACL of the migrated keys.
`--src-prots=<list of source protections>`	Specify a comma-separated list of OCS or softcard names in the source security world. The keys will be migrated to the corresponding protections specified with `--dst-prots`.
`--dst-prots=<list of destination protections>`	Specify a comma-separated list of OCS or softcard names in the destination security world. These will be the target protections for the keys that are protected with methods specified with `--src-prots` in the source security world.
`--prots-config=<PATH>`	Specify a configuration file that lists the source and destination protection pairs for migration. The file must contain pairs of tab-separated protection names `src_prot dst_prot`, one pair per line.

Do not terminate path names in the command parameters with a backslash character. If this is not possible then either terminate with a double backslash or insert a blank space between the backslash and the terminating quotation mark.

4.4. Migrating keys

4.4.1. Preparing for migration

Before you begin:

- Install the latest version of the Security World Software from the installation media.
- Ensure that the warrant files for the source and destination modules are stored in their default locations. If the warrant files are not at the default location, the `--src-warrant` and `--dst-warrant` parameters need to be specified in the `migrate-world` command.
  - For Solo +, or Solo XC, the default location is `NFAST_KMDATA/warrants/` (**Linux**) or `NFAST_KMDATA\warrants\` (**Windows**).
  - For Connect +, Connect XC modules, the default location is `NFAST_KMDATA/hsm-<ESN>/warrants/` (**Linux**) or `NFAST_KMDATA\hsm-<ESN>\warrants\` (**Windows**).
  - For nShield 5s and nShield 5c, you do not need to specify warrant locations because they store their warrants within the module.
- Copy the source World data to a location defined by the `--source=<SOURCE>` parameter of the migration tool.
- If the destination World does not exist already, create a new destination World. For instructions, see Create a new Security World.

You must enable all your features on the destination module before migration. Otherwise, the migration will fail.

4.5. Migrating keys process

To ensure the security of your keys, we recommend that the migration process is overseen by ACS-holding personnel and the end-to-end migration process is completed continuously, without any breaks in the process. This will also reduce the possibility of your ACS experiencing a time-out.

If the destination World supports audit logging you can choose whether the migrated keys will have key usage logging enabled or not by use of the `--key-logging` command line switch. If you only wish key usage logging to be enabled on a subset of the keys then you must separate the source keys into two groups and run the migrate-world command separately for each group.

To migrate keys to the destination World:

1. Run the migration utility in the perform mode with the required options. For information about the usage and options you can use, see About the migration utility.
2. Ensure that the data for the destination World is in the standard location for World data, derived from one of the following:
   - Either the environment variable `NFAST_KMLOCAL` or `NFAST_KMDATA`.
   - The default directory:
     - (**Linux**) `/opt/nfast/kmdata/local/`
     - (**Windows**) `C:\ProgramData\Key Management Data\local`, or `C:\Documents and Settings\All Users\Application Data\nCipher\Key Management Data\local`, as appropriate.
3. If the module is not configured to use the destination World, the utility prompts you to program the module and supply the ACS of the destination World.
4. The utility guides you through specific steps, prompting you to supply the required card sets and passphrases.
5. At the end of the migration both the source and destination modules are cleared. If you wish to use the modules then you must reload them with an appropriate Security World.

The utility will attempt to automatically change the module mode when needed. Should the automatic change of mode fail for any reason, then the utility will prompt you to change the module state to either initialization or operational at various points during the pro cedure.

4.6. Verifying the integrity of the migrated keys

To verify the integrity of the migrated keys, run the `nfkmverify` utility with the following options, as appropriate:

- If the keys are module-protected, run the utility with one of the following options:
  - `-L` option, which checks the ACL of a loaded key instead of the generation certificate.
  - `-R` option, which checks the ACL of a key loaded from a recovery blob.
- If the keys are protected by cardsets or softcards, run the `nfkmverify` utility with the `-R` option in combination with the preload utility.

  Example:

```
preload --admin=RE nfkmverify -R -m1 <application> <key-ident>
```

Do not use the `nfkmverify` utility with the default `-C` option. If you use this option, the utility returns errors because the ACL in the cer tificate will reflect the old world.

Note that if the destination World is `fips-140-level-3` then some keys that were usable in the source World may not be usable in the destination World due to those algorithms or key lengths being restricted. The migration tool will successfully migrate the keys but they will be restricted at the point of use.

4.7. Migrating keys using custom protection pairs

Regular security world migration will create new card sets and softcards in the destination world with the same names as the source protections or it will use existing destination pro tections if they share a name and type (card set or softcard) with the source protection.

You can specify custom protection pairs if you want to change the name, the quorum, or the properties of the protection. You can also combine multiple source protections of the same type into one destination protection. You cannot diffuse keys from one source protec tion to multiple destination protections.

The source-destination protection pairs can be selected either as:

- Two comma-separated lists `--src-prots <source protections>` and `--dst-prots <des tination protections>`.
- Tab-separated pairs "source destination", one per line, in a configuration file `--prots -config <file path>`.

The protections can be referred to by their name, 40-character hash, or "c:name" and "s:name" when a source card set and softcard share a name. The source and destination pro tection types must match.

The following example shows the two ways of specifying a set of protection pairs and the different ways each protection can be referred to. The example hashes are shortened for readability.

Protection type	Source protection to be migrated	Target destination protection
card set	`ocs 1`	`ocstarget1`

Protection type	Source protection to be migrated	Target destination protection
softcard	`softcard 1`	`softcardtarget`
card set	`name1` (duplicate name)	`ocstarget1`
softcard	`name1` (duplicate name)	`softcardtarget`
card set	`name2` (duplicate name and type) `hash: XXXXXXX1`	`ocstarget1`
card set	`name2` (duplicate name and type) `hash: XXXXXXX2`	`ocstarget2`

By specifying the lists using the `--src-prots` and `--dst-prots` options:

```
migrate-world [OPTIONS] \
--src-prots "ocs 1,softcard 1,c:name1,s:name1,XXXXXXX1,XXXXXXX2" \
--dst-prots "ocstarget1,softcardtarget,ocstarget1,softcardtarget,ocstarget1,ocstarget2"
```

By using a configuration file specified with the `--prots-config` option:

```
migrate-world [OPTIONS] --prots-config=migration.cfg

--- migration.cfg ---
ocs 1    ocstarget1
softcard 1    softcardtarget
c:name1    ocstarget1
s:name1    softcardtarget
XXXXXXX1    ocstarget1
XXXXXXX2    ocstarget2
--------------------
```

4.8. Troubleshooting

If you encounter any errors that are not listed in the following table, con tact Support.

Error	Explanation	Action
There are no keys requiring migration.	Any migrate-able keys found in the source World already exist in the destination World. The migration utility returns this error if: • The keys have already been migrated • All keys are non-recoverable and therefore cannot be migrated.	None.
Source module must be specified. Destination module must be specified. Source and Destination modules must be different. Module is not usable.	This utility requires you to specify both a source and destination module which must be different modules and both must be usable.	Specify the correct modules.
Source World has indistinguishable cardsets or softcards. Destination World has indistinguishable keys.	There are irregularities in one of the Worlds, but these irregularities do not affect the migration process.	None.
Destination World has indistinguishable cardsets or softcards. Source World has indistinguishable keys. Cannot determine protection of keys.	There are problems with one of the Worlds.	Contact Support.
Source World not recoverable.	The source World is not recoverable and the keys therefore cannot be migrated.	If the source World is not recoverable, you cannot use the migration utility to migrate keys. Contact Support.
Missing security World at PATH. Source world must be specified.	The path for the source World is wrong. There is no World data at the location that was specified when running the migration utility.	Supply the correct path to the source World. If you have supplied the correct path to the directory that contains the source World data, the migration utility has not found a destination World.

Error	Explanation	Action
Source World is the same as the destination World.	An incorrect path was supplied for the source World data when running the utility. The destination World data does not exist in the default location defined by the environment variable `NFAST_ KMLOCAL` or `NFAST_KMDATA`.	Run the utility with the correct path to the source World data. Move the source World data to a different location and then copy the destination World data to the default location. If the default location is defined by an environment variable, configure the variable to point to the location of the destination World, which then becomes the new default location.
Cannot find <NAME> utility, needed by this utility. <NAME> utility is too old, need at least version <VERSION NUMBER>.	The software installation is partially completed. The path (in the environment variable for the operating system) might be pointing to an old version of the software.	Reinstall the software. Ensure that the path points to the latest version of the software.
nFast error: TimeLimitExceeded; in response to SetKM...	The ACS time-out limit has expired.	Restart the key migration process; see Migrating keys.
Destination world does not support audit logging.	You have specified the `--key-logging` option but the destination world does not support audit logging.	None. The keys will be migrated but LogKeyUsage will not be set in the ACL of migrated keys.
Failed to load warrant file <FILE>.	There is a problem reading the warrant file.	Check that your warrant files are in the correct location and have not been edited in any way.

4.9. Migrating KMDATA (Windows)

To move KMDATA from the default location of `C:\ProgramData\nCipher`:

1. Open a command prompt window as an administrator.
2. Use `Xcopy` with the following parameters to copy the default folder to a new location:

```
Xcopy C:\ProgramData\nCipher <Destination> /e /v /o /i
```

3. Enter the new location for the following environment variables:
   a. In the Windows Control Panel, navigate to **Control Panel > System and Security > System > Advanced system settings**.

b. In the **Advanced** tab, select **Environment Variables**.

c. Update the following system variables:

- **NFAST_CERTDIR**: `<path\to\new\folder>\Feature Certificates`
- **NFAST_KMDATA**: `<path\to\new\folder>\Key Management Data`
- **NFAST_LOGDIR**: `<path\to\new\folder>\Log Files`

d. If your Security World client is on or above v12.70.4, add the following environment variable in the same section:

- **NFAST_KNETIDIR**: `<path\to\new\folder>\hardserver.d`

4. In the Services tool, restart the nFast Server process.
5. After the service restarts, run the following command to check the migration was successful:

```
anonkneti -m 127.0.0.1
```

6. After confirming that the migration was successful, delete `C:\ProgramData\nCipher`.

5. Common Criteria CMTS Mode Assigned Keys (nShield Solo XC)

Common Criteria CMTS mode includes the concepts of Assigned Keys and General Keys, as defined in EN 419 221-5.

Assigned Keys provide for more restrictive controls which are enforced with ACLs. An Assigned Key is a secret key with a Key Generation Certificate and with the ACL configuration defined in *nShield Solo XC Common Criteria Evaluated Configuration Guide*, specifically:

- The **Reauthorization conditions** and **Key Usage** attributes cannot be changed.
- The **Authorisation Data** attribute can only be changed by presentation of the current **Authorisation Data**, it cannot be changed or reset by an Administrator.
- The key cannot be exported by wrapping with another key.
- The key must be generated. It cannot be imported.

These properties of an Assigned Key enable the sole control that's required for a secret key used to create a digital signature.

A General Key is one that does not meet the criteria for an Assigned Key.

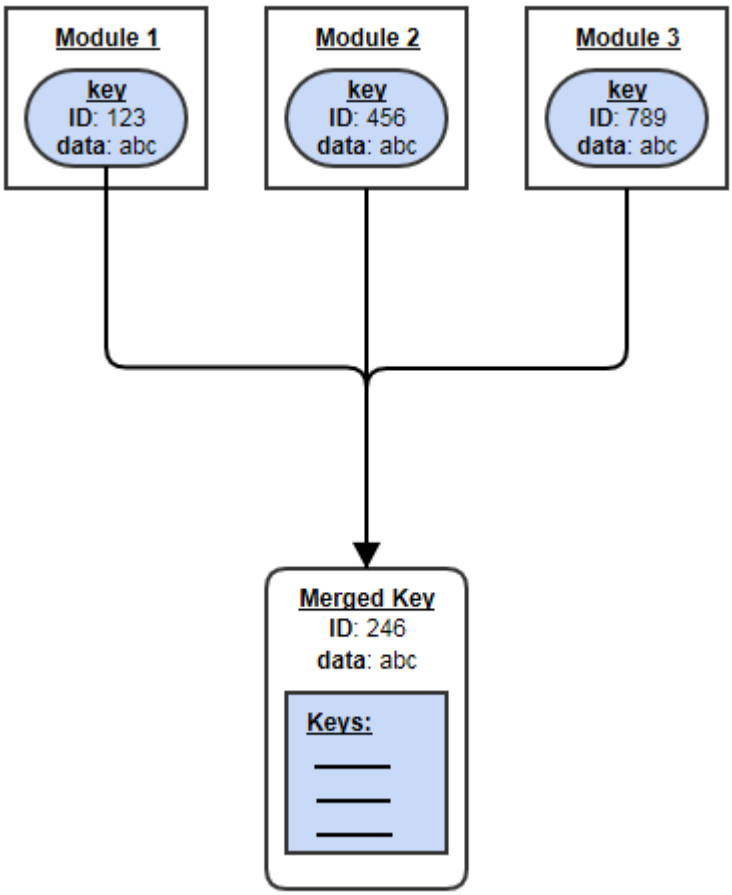
For both Assigned and General Keys in a Common Criteria CMTS Security World it is not possible to export or import as plain text. This is enforced by the HSM.

The ACL configuration defining an Assigned Key is described in the *nShield Solo XC Common Criteria Evaluated Configuration Guide*. Determination of the Assigned status of a key uses the `nfkmverify` utility and the Key Generation certificate recorded in the key when it was created.

The `generatekey` and `mkaclx` utilities have been enhanced to offer support for generating Assigned Keys, see Key generation options and parameters for `generatekey` and the online help for `mkaclx`.

6. Merged Keys Concept

A merged key is a level of abstraction higher than normal keys. It holds an internal list of nor mal key IDs, each associated with its corresponding module. When a command to the hard-server specifies a MergedKey ID instead of a normal (single) key ID, the hardserver chooses an HSM and corresponding single key ID from the list in the Merged Key. See diagram below. Which module is chosen may depend on multiple factors, including load sharing set-tings in the hardserver config.



Benefits of MergedKeys:

- A client need hold only a single M_KeyID reference instead of one for each HSM.
- That ID remains usable even while the key's actual IDs on HSMs can fluctuate.
- The hardserver can use heuristics to choose the most appropriate HSM (for example, the least heavily loaded one).
- If some HSMs become unavailable, the hardserver uses the remaining ones automati-cally.
    - A MergedKey can be updated, removing its entry for a defunct HSM and corre-

sponding single-key ID.

- New HSMs can be added: if a new HSM is made operational and added to the relevant security world, then
  - the key can be loaded onto that HSM, thus creating a new single-key ID for that key on that HSM, and then
  - the new (Key ID, HSM) pair can be added to the existing Merged Key.

7. Cryptographic algorithms

7.1. Introduction

This topic details the implemented restrictions imposed in various firmware modes. It covers different module features, not just algorithms and mechanisms.

For the most part, a blank table cell means "no restriction"; there are a few exceptions to this, for example, flag settings for particular modes. The information is low-level and may need interpreting to answer high-level questions. This topic does not cover higher level APIs like PKCS#11 or JCE.

The document was last updated in June 2024, for v13.5.1/v13.6.3.

Security World mode designation	new-world "mode" parameter	Description
Unrestricted		The unrestricted Security World mode protects keys with FIPS approved cryptography, but it is not designed to be fully compliant with all the requirements and restrictions of a particular certification standard. This mode can be used by customers who want their keys securely managed within the FIPS level 3 boundary, but don't need full compliance with the certification approved modes of operation. For Solo XC, Solo+ and Edge, the unrestricted mode is compliant with FIPS 140-2 Level 2.
FIPS 140 Level 3	`fips-140-level-3`	This is the FIPS 140 level 3 approved mode of operation. Customers needing FIPS 140 Level 3 compliance can use this mode on an HSM with a FIPS validated fw version.
Common Criteria CMTS	`common-criteria-cmts`	The Common Criteria approved mode of operation for Protection Profile EN 419 221-5 Cryptographic Module for Trust Services. Customers needing Common Criteria (CC) compliance can use this mode on an HSM with a CC validated fw version.

7.2. FIPS information

In a FIPS 140 Level 3 Security World, the nShield HSM only supports FIPS-approved algorithms and key sizes.

- If you have a FIPS 140 Level 3 Security World and have any protocols that use algorithms not approved by FIPS, you have the following options:
  - If you need to use these non-approved algorithms, you can migrate to a
    - **(nShield Connect, Edge, and Solo HSMs)** FIPS 140 Level 2 Security World.
    - **(nShield 5c and 5s HSMs)** Non-FIPS Security World but continue to use hardware and firmware validated for FIPS 140 Level 3.
  - If you have strict FIPS 140 Level 3 requirements, you must replace your protocols to use approved algorithms.
- If you have a FIPS 140 Level 3 Security World and have existing long-term keys for unapproved algorithms, you have the following options:
  - Migrate to a
    - **(nShield Connect, Edge, and Solo HSMs)** FIPS 140 Level 2 Security World.
    - **(nShield 5c and 5s HSMs)** Non-FIPS Security World but continue to use hardware and firmware validated for FIPS 140 Level 3.
  - Replace the keys with approved keys before upgrading to the current firmware. Keys for unapproved algorithms are incompatible with this Security World.

To obtain more details on the specific algorithms that are FIPS approved for use in the HSM, refer to the nShield Security Policy for the particular FIPS CMVP certified nShield product that you are using.

For the FIPS CMVP certificates for nShield products, see https://csrc.nist.gov/projects/cryptographic-module-validation-program/validated-modules/search. The FIPS CMVP certificate links to the Security Policy.

7.3. Compatibility of Security World versions with FIPS

To comply with the latest FIPS cryptographic transitions, Security World v3 was introduced in firmware version 12.50. If an nShield HSM is upgraded to use firmware version 12.50 or later, any v2 Security Worlds using the HSM that were compliant with FIPS 140 Level 3 will no longer be compliant.

You can create a v3 Security World that is compliant with FIPS 140 Level 3 from a host server if you meet the following criteria:

- The host server is running Security World host-side software version 12.50 or later.
- The HSM is running firmware version 12.50 or later.

Your solution is only FIPS 140 compliant if you are running the exact firmware version that has been FIPS 140 certified.

7.4. Configuration

In the following table, "Unrestricted", "FIPS 140 Level 3", and "Common Criteria CMTS" refer to the Security World mode designation. The cells in these columns detail any restrictions for the corresponding feature in each of the Security World modes. A blank cell means that the feature has no restrictions.

FIPS 140 Level 3: In v3 Security Worlds, in FIPS 140 Level 3 mode, some smaller key sizes are disabled.

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
InitModeFlags	UseFIPSApprovedInternalMechanisms		UseFIPSApprovedInternalMechanisms AuditLogging
NSOPermsModeFlags	AlwaysUseStrongPrimes	FIPSLevel3Enforcedv2 AlwaysUseStrongPrimes StrictSP80056Ar3	CommonCriteriaCMTSRestrictions AlwaysUseStrongPrimes
Public NSOPerms	ReadFile FormatToken GenerateLogToken LoadLogicalToken WriteShare ChangeSharePIN GetRTC	LoadLogicalToken WriteShare ChangeSharePIN GetRTC	ReadFile FormatToken GenerateLogToken LoadLogicalToken WriteShare ChangeSharePIN GetRTC

7.5. Functionality

In the following table, "Unrestricted", "FIPS 140 Level 3", and "Common Criteria CMTS" refer to the Security World mode designation. The cells in these columns detail any restrictions for the corresponding feature in each of the Security World modes. A blank cell means that the feature has no restrictions.

FIPS 140 Level 3: In v3 Security Worlds, in FIPS 140 Level 3 mode, some smaller key sizes are disabled.

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
Cmd_Import		No private key import Public key import requires FIPS auth	No private key import
ExportAsPlain		Forbidden for private keys	

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
Key generation		Requires FIPS auth	
Key generation		Pairwise check always on	
Impath			Forbidden
Minimum impath groups	DHPrime3072	DHPrimeMODP3072	n/a
Default module attributes	ModuleAttribTag_Challenge ModuleAttribTag_ESN ModuleAttribTag_KML ModuleAttribTag_KLF2 ModuleAttribTag_KNSO ModuleAttribTag_KMList ModuleAttribTag_KLF3 (nShield 5 & later)		
SignModuleState with KLF		Forbidden	
Audit logging			Mandatory
AlwaysUseStrongPrimes		Mandatory	

7.6. Asymmetric Algorithms and Mechanisms

In the following table, "Unrestricted", "FIPS 140 Level 3", and "Common Criteria CMTS" refer to the Security World mode designation. The cells in these columns detail any restrictions for the corresponding feature in each of the Security World modes. A blank cell means that the feature has no restrictions.

FIPS 140 Level 3: In v3 Security Worlds, in FIPS 140 Level 3 mode, some smaller key sizes are disabled.

7.6.1. Diffie-Hellman Key Agreement

Algorithm	FIPS approved in a v1 or v2 Security World	FIPS approved in a v3 Security World	Key type	Supported by generatekey
Diffie-Helman	Y	Y	DH or DHEx	Y
ElGamal	Y	Y	DH	Y

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
DHPrivate key generation (KeyType_DHPrivate)		Forbidden	

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
DHPrivate default size	1024/160	2048/224	1024/160
DHPrivate key agreement (Mech_DHKeyExchange)		Forbidden (including DLIES)	
DHExPrivate key generation (KeyType_DHExPrivate)			
DHExPrivate domain parameters		Restricted as per SP800-56Ar3	
DHExPrivate key generation minimum size		2048/224 minimum if |p|=3072, |q|>=256.	
DHExPrivate default size	2048/256		
DHExPrivate key agreement minimum size		2048	
DHExPrivate key agreement (Mech_DHExKeyExchange)		Forbidden with Cmd_Decrypt (Permitted with KDF)	
ElGamal encryption/decryption (Mech_ElGamal)		Forbidden	
IEEE DLIES with ANSI X9.63 KDF and 3DES CBC encryption (Mech_DLIESe3DEShSHA1)		Forbidden	
IEEE DLIES with ANSI X9.63 KDF and AES CBC encryption (Mech_DLIESeAEShSHA1)		Forbidden	
IEEE DLIES with ANSI X9.63 KDF and AES CBC encryption (Mech_DLIESeAEShSHA1DHEx)			

When a DHEx key is loaded into the module, the domain parameters are validated. If the domain parameters do not match those found in SP800-56Ar3, the validation time is signifi cantly longer. Entrust recommends that you always use SP800-56Ar3 domain parameters.

7.6.2. DSA Signature

Algorithm	FIPS approved in a v1 or v2 Security World	FIPS approved in a v3 Security World	Key type	Supported by generatekey
DSA	Y	Y [1]	DSA	Y

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
DSA key generation (KeyType_DSA)		Forbidden in ECp521-mAES FIPS worlds[1]	
DSA key generation sizes		FIPS 186-4 sizes only; 2048 minimum	
DSA signature key sizes		FIPS 186-4 sizes only; 2048/224 minimum	
DSA signature hashes		RIPEMD160 & SHA-1 forbidden	
Legacy DSA domain generation (KeyType_DSAComm)		Forbidden	
Legacy DSA domain generation (KeyType_DSACommVariableSeed)			
FIPS 186-4 DSA domain generation (KeyType_DSACommFIPS186_3)			
DSA SHA-1 signature (Mech_DSA)		Forbidden	
DSA SHA-2 signature (Mech_DSAhSHA224, Mech_DSAhSHA256, Mech_DSAhSHA384, Mech_DSAhSHA512)			
DSA RIPMED160 signature (Mech_DSAhRIPMED160)		Forbidden	

[1] DSA is no longer approved for digital signature generation in FIPS 186-5, and therefore not permitted in a FIPS 140 Level 3 Security World created with the `ECp521mAES` cipher suite.

7.6.3. RSA Signature/Encryption

Algorithm	FIPS approved in a v1 or v2 Security World	FIPS approved in a v3 Security World	Key type	Supported by generatekey
RSA	Y	Y	RSA	Y

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
RSA key generation (KeyType_RSAPrivate)	Strong primes always on[1]		
RSA key generation public modulus size		2048 minimum; multiple of 2	
RSA key generation rules (<1024)	FIPS 186-4 B.3.6	Forbidden	FIPS 186-4 B.3.6
RSA key generation rules (>=1024)	FIPS 186-4 B.3.6		
RSA key generation/import public exponent		16-256 bits	
RSA signature key sizes		2048 minimum	
RSA signature hashes		RIPEMD160 & SHA-1 forbidden	
RSA raw encryption/decryption (any RSA mech with bignum plaintext)		Forbidden with Mech_RSApPKCS1 (pPKCS11), permitted otherwise	
RSA PKCS#1 encryption/decryption (Mech_RSApPKCS1, Mech_RSApPKCS1pPKCS11 with bytes plaintext)		Forbidden	
RSA raw sign/verify (any RSA mech with bignum plaintext)		Forbidden with Mech_RSApPKCS1 (pPKCS11), permitted otherwise	

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
RSA PKCS#1 any-hash signature (Mech_RSApPKCS1, Mech_RSApPKCS1pPKCS11 with bytes/hash plaintext)		Forbidden	
RSA PKCS#1 SHA-1 signature (Mech_RSApPKCS1, Mech_RSAhSHA1pPKCS1 with bytes/hash plaintext)		Forbidden	
RSA PKCS#1 SHA-2 signature (Mech_RSAhSHA224pPKCS1, Mech_RSAhSHA256PKCS1, Mech_RSAhSHA384pPKCS1, Mech_RSAhSHA512pPKCS1 with bytes/hash plaintext)			
RSA PKCS#1 SHA-3 signature (Mech_RSAhSHA3b224pPKCS1, Mech_RSAhSHA3b256PKCS1, Mech_RSAhSHA3b384pPKCS1, Mech_RSAhSHA3b512pPKCS1 with bytes/hash plaintext)			
RSA PSS SHA-1 signature (Mech_RSAhSHA1pPSS with bytes/hash plaintext)		Forbidden	
RSA PSS SHA-2 signature (Mech_RSAhSHA224pPSS, Mech_RSAhSHA256pPSS, Mech_RSAhSHA384pPSS, Mech_RSAhSHA512pPSS with bytes/hash plaintext)			

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
RSA PSS SHA-3 signature (Mech_R-SAhSHA3b224pPSS, Mech_R-SAhSHA3b256pPSS, Mech_R-SAhSHA3b384pPSS, Mech_R-SAhSHA3b512pPSS with bytes/hash plaintext)			
RSA PSS RIPEMD160 signature (Mech_R-SAhRIPMED160pPSS with bytes/hash plaintext)		Forbidden	
RSA SHA-1 OAEP encryption (Mech_RSApOAEP with bytes plaintext)			
RSA SHA-2 OAEP encryption (Mech_R-SApOAEPhSHA224, Mech_R-SApOAEPhSHA256, Mech_R-SApOAEPhSHA384, Mech_R-SApOAEPhSHA512 with bytes plaintext)			
RSA SHA-3 OAEP encryption (Mech_R-SApOAEPhSHA3b224, Mech_R-SApOAEPhSHA3b256, Mech_R-SApOAEPhSHA3b384, Mech_R-SApOAEPhSHA3b512 with bytes plaintext)			

[1] FIPS Security Worlds always have "always use strong primes" enabled. This setting is optional for non-FIPS Security Worlds. The "strong primes" algorithm is the only FIPS-com-

pliant RSA keygen algorithm currently offered.

7.6.4. Elliptic Curve Key Agreement

Algorithm	FIPS approved in a v1 or v2 Security World	FIPS approved in a v3 Security World	Key type	Supported by generatekey
ECDH	Y	Y	ECDH or EC	Y
ECIES	N	N	ECDH or EC	N

`KeyType_ECPrivate` allows a single key to be used for key establishment and signature generation, depending on the permissions in its ACL. If you require FIPS 140 compliance, then additional care must be taken to comply with the rules about using a single key for multiple purposes, such as section 5.2, *General Key Management Guidance: Key Usage* of SP800-57pt1r5. The HSM can help enforce these rules, for example, by placing the sign permission in a permission group with `UseLim_Global` (use limit) set to a maximum use count of 1.

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
ECC enablement	EllipticCurve feature (enabled by default from firmware V13.5 onwards)		
ECC domain parameters		224 minimum; SECP256k1 forbidden; non-named curves forbidden	
ECDH key agreement (Mech_ECDHKeyExchange)		Forbidden with Cmd_Decrypt (Permitted with Cmd_DeriveKey)	
ECDHC key agreement (Mech_ECDHCKeyExchange)		Forbidden with Cmd_Decrypt (Permitted with Cmd_DeriveKey)	
ECDH key generation (KeyType_ECDHPrivate, KeyType_ECPrivate)			
ECDHLax key generation (KeyType_ECDHLaxPrivate)		Forbidden	

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
ECDHLax key agreement (Mech_ECDHLaxKeyExchange)		Forbidden	

7.6.5. Elliptic Curve Signature

Algorithm	FIPS approved in a v1 or v2 Security World	FIPS approved in a v3 Security World	Key type	Supported by generatekey
ECDSA	Y [1]	Y [1]	ECDSA or EC	Y

[1] FIPS 140 approval is only for use with ECDSA keys, not with EC keys.

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
ECC enablement	EllipticCurve feature enabled by default from V13.5 onwards		
ECC domain parameters		224 minimum; SECP256k1 forbidden; non-named curves forbidden	
ECDSA key generation (KeyType_ECDSAPrivate, KeyType_ECPrivate)			
ECDSA signature RNG		Never uses unvalidated RNG	
ECDSA signature hash		RIPEMD160 & SHA-1 forbidden	
ECDSA verify hash		RIPEMD160 forbidden	
ECDSA SHA-1 sign (Mech_ECDSA)		Forbidden	
ECDSA SHA-1 verify (Mech_ECDSA)			
ECDSA RIPMED160 sign/verify (Mech_ECDSAhRIPEMD160)		Forbidden	

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
ECDSA SHA-2 sign/verify (Mech_ECDSAhSHA224, Mech_ECDSAhSHA256, Mech_ECDSAhSHA384, Mech_ECDSAhSHA512)			
ECDSA SHA-3 sign/verify (Mech_ECD-SAhSHA3b224, Mech_ECDSAhSHA3b256, Mech_ECDSAhSHA3b384, Mech_ECDSAhSHA3b512)			
ECDSA sign/verify GBCS mode (Mech_ECD-SAhSHA256kGBCS)	Forbidden		

7.6.6. X25519/Curve25519 Signature/Encryption

Algorithm	FIPS approved in a v1 or v2 Security World	FIPS approved in a v3 Security World	Key type	Supported by generatekey
X25519	N	N	X25519	Y
Ed25519	N	N	Ed25519	Y

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
Ed25519 key generation (KeyType_Ed25519Pri-vate)		Firmware versions 13.6.x and older: Forbidden Firmware versions 13.7.x and newer: No restriction	
Pure Ed25519 sign/verify (Mech_Ed25519)		Firmware versions 13.6.x and older: Forbidden Firmware versions 13.7.x and newer: No restriction	
Prehashed Ed25519 sign/verify (Mech_Ed25519ph)		Firmware versions 13.6.x and older: Forbidden Firmware versions 13.7.x and newer: No restriction	

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
Prehashed Ed25519 sign/verify with context (Mech_Ed25519phctx)		Firmware versions 13.6.x and older: Forbidden Firmware versions 13.7.x and newer: No restriction	
X25519 key generation (KeyType_X25519Private)		Forbidden	
X25519 key agreement (Mech_X25519KeyExchange)		Forbidden	

7.6.7. Ed448 Signature

Algorithm	FIPS approved in a v1 or v2 Security World	FIPS approved in a v3 Security World	Key type	Supported by generatekey
Ed448	Y	Y	Ed448	N

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
Ed448 key generation (KeyType_Ed448Private)		Firmware versions 13.6.x and older: Forbidden Firmware versions 13.7.x and newer: No restriction	
Pure Ed448 sign/verify (Mech_Ed448)		Firmware versions 13.6.x and older: Forbidden Firmware versions 13.7.x and newer: No restriction	
Pure Ed448 sign/verify with context (Mech_Ed448ctx)		Firmware versions 13.6.x and older: Forbidden Firmware versions 13.7.x and newer: No restriction	
Prehashed Ed448 sign/verify (Mech_Ed448ph)		Firmware versions 13.6.x and older: Forbidden Firmware versions 13.7.x and newer: No restriction	

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
Prehashed Ed448 sign/verify with context (Mech_Ed448phctx)		Firmware versions 13.6.x and older: Forbidden Firmware versions 13.7.x and newer: No restriction	

7.6.8. KCDSA Signature

Algorithm	FIPS approved in a v1 or v2 Security World	FIPS approved in a v3 Security World	Key type	Supported by generatekey
KCDSA	N	N	KCDSA	N

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
KCDSA enablement	KISAAlgorithms feature required		
KCDSA key generation (KeyType_KCDSAPrivate)		Forbidden	
KCDSA signature (Mech_KCDSAHASH160, Mech_KCDSASHA1, Mech_KCDSASHA224, Mech_KCDSASHA256, Mech_KCDSARIPMED160)		Forbidden	
KCDSA domain generation (KeyType_KCDSACommon)		Forbidden	

7.7. Symmetric Mechanisms and Algorithms

In the following table, "Unrestricted", "FIPS 140 Level 3", and "Common Criteria CMTS" refer to the Security World mode designation. The cells in these columns detail any restrictions for the corresponding feature in each of the Security World modes. A blank cell means that the feature has no restrictions.

FIPS 140 Level 3: In v3 Security Worlds, in FIPS 140 Level 3 mode, some smaller key sizes are disabled.

7.7.1. ARIA

Algorithm	FIPS approved in a v1 or v2 Security World	FIPS approved in a v3 Security World	Key type	Supported by generatekey
ARIA	N	N	Aria	N

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
ARIA key generation (KeyType_ARIA)		Forbidden	
ARIA CBC no padding (Mech_ARIAmCBCp-NONE)		Forbidden	
ARIA ECB no padding (Mech_ARIAmECBp-NONE)		Forbidden	

7.7.2. Camellia

Algorithm	FIPS approved in a v1 or v2 Security World	FIPS approved in a v3 Security World	Key type	Supported by generatekey
Camellia	N	N	Camellia	N

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
Camellia key generation (KeyType_Camellia)		Forbidden	
Camellia CBC no padding (Mech_CamelliamCBCp-NONE)		Forbidden	
Camellia ECB no padding (Mech_CamelliamECBp-NONE)		Forbidden	

7.7.3. CAST256

Algorithm	FIPS approved in a v1 or v2 Security World	FIPS approved in a v3 Security World	Key type	Supported by generatekey
CAST 256	N	N	CAST256	N

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
CAST256 key generation (KeyType_CAST256)		Forbidden	
CAST256 CBC PKCS#5 padding (Mech_CAST256mCB-Ci128pPKCS5)		Forbidden	
CAST256 ECB PKCS#5 padding (Mech_CAST256mECBpP-KCS5)		Forbidden	
CAST256 CBC no padding (Mech_CAST256mCBCp-NONE)		Forbidden	
CAST256 ECB no padding (Mech_CAST256mECBp-NONE)		Forbidden	
CAST256 CBC-MAC PKCS#5 padding (Mech_CAST256mCBC-MACi0pPKCS5)		Forbidden	

7.7.4. DES

Algorithm	FIPS approved in a v1 or v2 Security World	FIPS approved in a v3 Security World	Key type	Supported by generatekey
DES	N	N	DES	N
DES2	N	N	DES	Y
Triple DES	Y	N [1]	Triple DES	Y

[1] Not FIPS approved for encryption operations, but available for decryption operations.

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
Single-DES key generation (KeyType_DES)		Forbidden	

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
Single-DES CBC PKCS#5 padding (Mech_DESmCBCi64pPKCS5)		Forbidden	
Single-DES CBC no padding (Mech_DESmCBCpNONE)		Forbidden	
Single-DES ECC PKCS#5 padding (Mech_DESmEBCpPKCS5)		Forbidden	
Single-DES ECB no padding (Mech_DESmECBpNONE)		Forbidden	
Single-DES CBC-MAC PKCS#5 padding (Mech_DESmCBCMACi0pPKCS5)		Forbidden	
Single-DES CBC-MAC no padding (Mech_DESmCBCMACpNONE)		Forbidden	
2-key triple-DES key generation (KeyType_DES2)		Forbidden	
2-key triple-DES PKCS#5 padding (Mech_DES2mCBCi64pPKCS5)		Forbidden	
2-key triple-DES CBC no padding (Mech_DES2mCBCpNONE)		Forbidden	
2-key triple-DES ECC PKCS#5 padding (Mech_DES2mEBCpPKCS5)		Forbidden	

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
2-key triple-DESS ECB no padding (Mech_DES2mECBp-NONE)		Forbidden	
2-key triple-DES CBC-MAC PKCS#5 padding (Mech_DES2mCBC-MACi0pPKCS5)		Forbidden	
2-key triple-DES CBC-MAC no padding (Mech_DES2mCBCMACp-NONE)		Forbidden	
3-key triple-DES key generation (KeyType_DES3)		Forbidden	
3-key triple-DES PKCS#5 padding (Mech_DES3mCBCi64pP-KCS5)		Decrypt only	
3-key triple-DES CBC no padding (Mech_DES3mCBCp-NONE)		Decrypt only	
3-key triple-DES ECC PKCS#5 padding (Mech_DES3mEBCpP-KCS5)		Decrypt only	
3-key triple-DESS ECB no padding (Mech_DES3mECBp-NONE)		Decrypt only	
3-key triple-DES CBC-MAC PKCS#5 padding (Mech_DES3mCBC-MACi0pPKCS5)		Forbidden	
3-key triple-DES CBC-MAC no padding (Mech_DES3mCBCMACp-NONE)		Forbidden	

7.7.5. AES (aka Rijndael)

Algorithm	FIPS approved in a v1 or v2 Security World	FIPS approved in a v3 Security World	Key type	Supported by generatekey
AES	Y	Y	AES or Rijndael	Y

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
AES key generation (KeyType_Rijndael)			
AES CBC PKCS#5 padding (Mech_RijndaelmCB-Ci128pPKCS5)			
AES ECB PKCS#5 padding (Mech_RijndaelmECBpP-KCS5)			
AES CBC no padding (Mech_RijndaelmCBCp-NONE)			
AES ECB no padding (Mech_RijndaelmECBp-NONE)			
AES GCM (Mech_RijndaelmGCM) with module-generated IV			
AES GCM (Mech_RijndaelmGCM) with user-supplied IV		Forbidden	
AES GCM (Mech_AESmGCM)			
AES KWP (Mech_AESKeyWrap-Padded)			
AES CMAC with PKCS#5 padding (Mech_RijndaelmCMAC)			

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
AES CBC-MAC with PKCS#5 padding (Mech_RijndaelmCBC-MACi0pPKCS5)		Forbidden	
AES CBC-MAC with no padding (RijndaelmCBCMACi0p-NONE)		Forbidden	

7.7.6. RC4

Algorithm	FIPS approved in a v1 or v2 Security World	FIPS approved in a v3 Security World	Key type	Supported by generatekey
Arcfour	N	N	Arcfour	N

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
RC4 key generation (KeyType_ArcFour)		Forbidden	
RC4 encrypt/decrypt (Mech_ArcFourpNONE)		Forbidden	

7.7.7. SEED

Algorithm	FIPS approved in a v1 or v2 Security World	FIPS approved in a v3 Security World	Key type	Supported by generatekey
SEED	N	N	SEED	N

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
SEED key generation (KeyType_SEED)		Forbidden	
SEED CBC PKCS#5 padding (Mech_SEEDmCBCi128pPKCS5)			

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
SEED ECBPKCS#5 padding (Mech_SEEDmECBpP-KCS5)			
SEED CBC no padding (Mech_SEEDmCBCp-NONE)			
SEED ECB no padding (Mech_SEEDmECBp-NONE)			
SEED CBC-MAC PKCS#5 padding (Mech_SEEDmCBC-MACi0pPKCS5)			

7.7.8. HMAC

Algorithm	FIPS approved in a v1 or v2 Security World	FIPS approved in a v3 Security World	Key type	Supported by generatekey
MD5 HMAC	N	N	HMACMD5	N
RIPEMD160 HMAC	N	N	HMACRIPEMD160	N
SHA-1 HMAC	Y	Y	HMACSHA1	Y
SHA-224 HMAC	Y	Y	HMACSHA224	N
SHA-256 HMAC	Y	Y	HMACSHA256	Y
SHA-384 HMAC	Y	Y	HMACSHA384	Y
SHA-512 HMAC	Y	Y	HMACSHA512	Y

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
HMAC SHA-1/2/3 key generation (KeyType_HMACSHA1, KeyType_HMACSHA224, KeyType_HMACSHA256, KeyType_HMACSHA384, KeyType_HMACSHA512, KeyType_HMAC-SHA3b224, KeyType_HMAC-SHA3b256, KeyType_HMAC-SHA3b384, KeyType_HMAC-SHA3b512)		Minimum 14 bytes (112 bits)	
HMAC SHA-1/2/3 sign/verify (Mech_HMACSHA1, Mech_HMACSHA224, Mech_HMACSHA256, Mech_HMACSHA384, Mech_HMACSHA512, Mech_HMACSHA3b224, Mech_HMACSHA3b256, Mech_HMACSHA3b384, Mech_HMACSHA3b512)			
HMAC MD5 key generation (KeyType_HMACMD5)		Forbidden	
HMACMD5 sign/verify (Mech_HMACMD5)		Forbidden	
HMAC RIPEMD160 key generation		Forbidden	
HMACRIPEMD160 sign/verify (Mech_HMACRIPEMD160)		Forbidden	

7.8. DeriveKey Mechanisms

In the following table, "Unrestricted", "FIPS 140 Level 3", and "Common Criteria CMTS" refer to the Security World mode designation. The cells in these columns detail any restrictions for the corresponding feature in each of the Security World modes. A blank cell means that

the feature has no restrictions.

FIPS 140 Level 3: In v3 Security Worlds, in FIPS 140 Level 3 mode, some smaller key sizes are disabled.

7.8.1. Key Wrapping (see also IES variants)

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
EncryptMarshalled (DeriveMech_EncryptMarshalled, DeriveMech_DecryptMarshalled)		AESKeyWrapPadded & RSApPKC-S1OAEPhSHA512 only	
AESKW non-default ICV		Forbidden (wrap & unwrap)	
Raw encryption (DeriveMech_RawEncrypt, DeriveMech_Decrypt) permitted mechanisms		AESKeyWrapPadded, RijndaelmGCM, AESmGCM, OAEP with NIST hashes	
Padded raw encryption (DeriveMech_RawEncryptZeroPad, DeriveMech_RawDecryptZeroPad)		Forbidden	
PKCS#8 wrap (DeriveMech_PKCS8Encrypt, DeriveMech_PKCS8Decrypt, DeriveMech_PKCS8DecryptEx) permitted mechanisms		AESKeyWrapPadded, RijndaelmGCM, AESmGCM, OAEP with NIST hashes	
AES Key Wrap (DeriveMech_AESKeyWrap, DeriveMech_AEKeyUnwrap) (see also Mech_AESKeyWrapPadded)			

7.8.2. Key Derivation

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
MAC on a key (DeriveMech_RawSign)		KeyType_Random output only	
NIST SP800-56Cr1 KDF (DeriveMech_ConcatenationKDF) with SHA1 or SHA-2			
NIST SP800-56Cr1 KDF (DeriveMech_ConcatenationKDF) with RIPEMD160 hash		Forbidden	
ANSI X9.63 KDF (DeriveMech_ConcatenationKDF)		Forbidden	
Either ConcatenationKDF with RSA key agreement (DeriveMech_ConcatenationKDF)		Forbidden	
Either ConcatenationKDF with ECDHC key agreement (DeriveMech_ConcatenationKDF)			
Either ConcatenationKDF with ECDH key agreement (DeriveMech_ConcatenationKDF) with h=1			
Either ConcatenationKDF with ECDH (DeriveMech_ConcatenationKDF) with h>1		Forbidden	
SP800-108 KDF with AES-CMAC (DeriveMech_NISTKDFmCTRpRijndaelCMACr32)			
SP800-108 KDF with AES-CMAC or HMAC SHA-256, HMAC SHA-384 or HMAC-384 (DeriveMech_NISTKDFmCTRr8)			

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
DES split/join XOR (DeriveMech_DESsplitXOR, DeriveMech_DESjoinXOR, DeriveMech_DESjoinXORsetParity, DeriveMech_DES2splitXOR, DeriveMech_DES2joinXOR, DeriveMech_DES2joinXORsetParity, DeriveMech_DES3splitXOR, DeriveMech_DES3joinXOR, DeriveMech_DES3joinXORsetParity)		Forbidden	
Random split/join XOR (DeriveMech_RandsplitXOR, DeriveMech_RandjoinXOR)			
AES split/join XOR (DeriveMech_AESsplitXOR, DeriveMech_AESjoinXOR)			
Key concatenation (DeriveMech_ConcatenateBytes)			
Public from private (DeriveMech_PublicFromPrivate)			

7.8.3. Key Agreement

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
ECCMQV with ANSI X9.63 KDF (DeriveMech_ECCMQV)		Forbidden	

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
ECCMQV with SP800-56Ar3 KDF (DeriveMech_ECCMQVd-NISTCKDF)			
ECDH key agreement (DeriveMech_ECDHKA)		Forbidden	
DH key agreement (DeriveMech_DHKA)		Forbidden	
X25519 key agreement (DeriveMech_X25519KA)		Forbidden	

7.8.4. IES Variants

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
ECIES (DeriveMech_ECIESKey-Wrap, DeriveMech_ECIESKeyUnwrap) with ECDH/ECDHC and ANSI X9.63 KDF		Forbidden	
X25519 ECIES (DeriveMech_ECIESKey-Wrap, DeriveMech_ECIESKeyUnwrap)		Forbidden	
RSA key wrap of symmetric key (DeriveMech_RSAKey-Wrap, DeriveMech_RSAKeyUnwrap) with OAEP and AES-KWP			
RSA key wrap of asymmetric key (DeriveMech_RSAKey-Wrap, DeriveMech_RSAKeyUnwrap) with OAEP, AES-KWP and PKCS#8			

7.8.5. Rainbow

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
ARQC verification (DeriveMech_Compos-iteARQCVerify)		Forbidden	
Watchword sign/verify (DeriveMech_Composite-WatchWordVerify, DeriveMech_Composite-WatchWordSign)		Forbidden	

7.8.6. HyperLedger

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
HyperLedger client key derivation (DeriveMech_HyperledgerClient)		Forbidden	

7.8.7. MILENAGE

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
MILENAGEOP key generation		Forbidden	
MILENAGESubscriber key generation		Forbidden	
MILENAGERC key generation		Forbidden	
MILENAGEOPC key derivation		Forbidden	
MILENAGEAV key derivation (f1...f5)		Forbidden	
MILENAGEResync (f1s/f5s)		Forbidden	
MILENAGEGenAUTS (for testing)		Forbidden	

7.8.8. TUAK

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
TUAKSubscriber key generation		Forbidden	
TUAKTOP key generation		Forbidden	
TUAKf1 key derivation		Forbidden	
TUAKf1s key derivation		Forbidden	
TUAKf2345 key derivation		Forbidden	
TUAKf5s key derivation		Forbidden	

7.8.9. Hashing

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
SHA-1 (Mech_SHA1Hash)			
SHA-2 (Mech_SHA224Hash, Mech_SHA256Hash, Mech_SHA384Hash, Mech_SHA512Hash)			
SHA-3 (Mech_SHA3b224Hash, Mech_SHA3b256Hash, Mech_SHA3b384Hash, Mech_SHA3b512Hash)			
HAS160 (Mech_HAS160Hash)		Forbidden	
RIPEMD160 (Mech_RIPEMDS160Hash)		Forbidden	
Tiger (Mech_TigerHash)		Forbidden	

7.9. Internal Security Mechanisms

In the following table, "Unrestricted", "FIPS 140 Level 3", and "Common Criteria CMTS" refer to the Security World mode designation. The cells in these columns detail any restrictions

for the corresponding feature in each of the Security World modes. A blank cell means that the feature has no restrictions.

FIPS 140 Level 3: In v3 Security Worlds, in FIPS 140 Level 3 mode, some smaller key sizes are disabled.

Feature	Unrestricted	FIPS 140 Level 3	Common Criteria CMTS
3DES internal security mechanisms (Mech_3DESwSHA1, Mech_3DESwCRC32)	Forbidden		
V2 Blobcrypt (AES, RSA & DH ISMs)	Forbidden		
V3 Blobcrypt (AES & RSA ISMs)	Mandatory		
Share key KDF	Proprietary KDF	NISTKDFmCTRpRijndaelCMACr32	