nCIPHER

www.ncipher.com

Non-proprietary Security Policy FIPS 140-2 level 3

nShield SOLO XC F3 &
nShield SOLO XC F3 for nShield Connect XC

Copyright

Date	April 2019
Doc. No	TesUSA-DDQ-000054-EN

nCipher Security Limited
Registered Office: 350 Longwater Avenue,
Green Park, Reading, R62 66F, United Kingdom
Registered in England No. 00868273

Contents

Tables

Figure

1. Introduction

1.1 Purpose

The nShield Hardware Security Modules provide support for the widest range of cryptographic algorithms, application programming interfaces (APIs) and host operating systems, enabling the devices to be used with virtually any business application—from identity management, web services and database encryption to tokenization, PKI services and strong authentication.

The nShield Hardware Security Modules are defined as a multi-chip embedded cryptographic modules as defined by FIPS 140-2. Both modules, enumerated below, possess the following attributes:

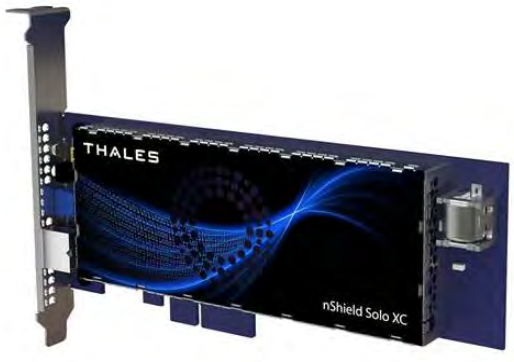
- Real Time Clock
- Potting
- Cryptographic acceleration
- EMC classification B
- Secure Execution Environment (optional)

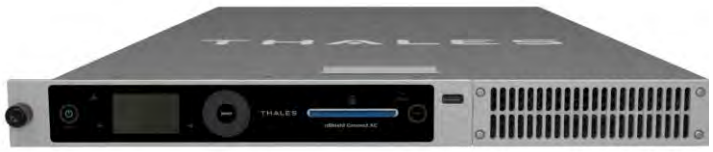
Unit ID	Hardware number	Overall FIPS level
nShield Solo XC F3	NC4035E-000	3
nShield Solo XC F3 for nShield Connect XC	NC4335N-000[1]	3

Table 1 - Product configuration

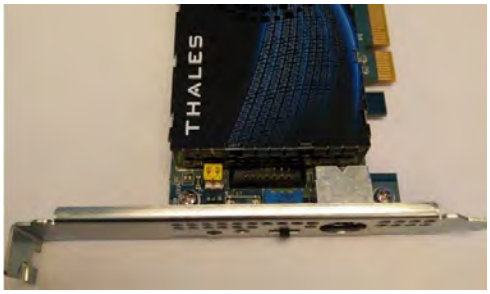
	nShield Solo XC F3 & nShield Solo XC F3 for nShield Connect XC
Firmware versions	3.3.21 3.4.1 3.4.2
Cryptographic library	nShield X Algorithm library

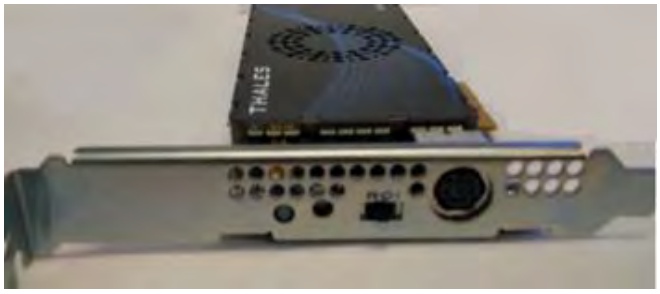
Table 2 - Product references





[1] This module is embedded in the nShield Connect XC appliance with model number NH2075-x (where x is B, M or H)





NOTE: The nShield Solo XC F3 is embedded in the Connect XC

Figure 1 – nCipher nShield overview

All modules are supplied at build standard "A".

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

The modules run firmware provided by nCipher. There is the facility for the Crypto Officer 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 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 cryptographic 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.

The modules 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 an on-board Real-time clock.

The module can be connected to a computer running one of the following operating systems (Windows, Solaris, HP-UX, AIX, Linux x86 / x64). Windows and Linux was used to test the module for this validation.

1.2 Overall FIPS level 3

The FIPS 140-2 security levels for the module in overall FIPS level 3 configuration are as follows:

Security Requirement	Security level
Cryptographic Module Specification	3
Cryptographic Module Ports and Interfaces	3
Roles, Service and Authentication	3
Finite State Model	3
Physical Security	3
Operational Environment	NA
Cryptographic Key Management	3
EMI/EMC	3
Self-Tests	3
Design Assurance	3
Mitigation of Other Attacks	NA
Overall FIPS level	3

Table 3 - Security Level of Security Requirements

1.3 Cryptographic boundary

The physical cryptographic boundary is the potted area on the PCB board, which includes the epoxy resin itself, the metal jig surrounding the potted area, and the heatsink.

1.4 Ports and Interfaces

The module has the following physical ports:

- PCIe2 bus (data input/output, control input, status output and power)
- Status LED (status output)
- PS-2 Serial connector (data input/output)
- Mode switch (control input)
- 16-way header (data input/output)
- DIP switches (control input)
- Clear button (control input)

1.5 Operational environment

The module's operating environment is non-modifiable. The FIPS 140-2 Operational Environment requirements are not applicable because the module contains a non-modifiable operational environment

2. Cryptographic functionality

2.1 Critical Security Parameters

Table 4 below enumerates the Critical Security Parameters utilized in the nShield Solo XC, and Table 5 breaks out the asymmetric public portions of those CSPs in order to document the specific input, output, storage, and zeroization attributes and services associated with those CSP objects.

CSP	Type	Description	Generation	Input	Output	Storage	Zeroization
Application keys (KA)	See description	Keys associated with a user to perform cryptographic operations, that can be used with one of the following validated algorithms: - AES keys - #3664, #3697, #3711 - Triple-DES keys - #2046, #2073 - RSA keys - #1897, #1917, #1903 - DSA keys - #1034, #1039 - ECDSA keys - #771, #790, #776 - Key Agreement - #669, #696, #682 - KBKDF - #73, #75 - ECMQV - #1111	DRBG	Load Blob - encrypted	Make Blob - encrypted	Ephemeral, stored in volatile RAM	Destroy, Initialize Unit, Clear Unit
Archiving keys (KR)	AES-256 CBC with HMAC SHA2 integrity mechanism	Key used to protect an archive copy of a key.	DRBG	Load Blob - encrypted	Make Blob - encrypted	Ephemeral, stored in volatile RAM	Initialize Unit
DRBG internal state	N/A	The module uses the Hash_DRBG from SP800-90A with SHA-256 as the underlying approved hash function. This DRBG is seeded from the on-board entropy source whenever the module is initialized and is reseeded according to SP800-90A with a further 512 bits of entropy after every 2048 bytes of output.	NDRNG	No	No	Ephemeral, stored in volatile RAM	Clear Unit
EncFSKey	AES-256 CBC	An AES Key used to protect the Encrypting File System. Constructed at startup time using a vendor affirmed PBKDF (case 1). For the PBKDF process, the password has 288 bits of entropy, the salt has 800 bits of entropy, and the process utilizes 10, 000 iterations. The password and salt consist of entropy generated in the module from the approved DRBG at manufacture time.	See description	No	No	Ephemeral, stored in volatile RAM	Clear Unit

CSP	Type	Description	Generation	Input	Output	Storage	Zeroization
Firmware Confidentiality Key (KFC)	Triple-DES 3 keys, CBC mode	Protect the source code during transport. The source code is deciphered and stored only after the firmware integrity check completes successfully.	At nCipher Security	Firmware Update - encrypted	No	Non-volatile memory – flash	N/A
Impath keys	AES-128 CBC, integrity HMAC with SHA 256	Used for secure channel between two modules. It consists in a set of four keys for cryptographic operations: encryption, decryption, MAC creation and MAC validation.	DH 3072 bit key-exchange between two modules	No	No	Ephemeral, stored in volatile RAM	Initialize Unit, Clear Unit
Key blob	N/A	Used for secure external storage of keys. A Key blob is encrypted by a Logical Token (LT), Module Key (KM), and optionally an Archiving key (KR).	N/A	Load Blob - encrypted	Make Blob - encrypted	Ephemeral, stored in volatile RAM	Clear Unit
KJSO	3072 DSA	nShield Junior Security Officer key used with its associated certificate to perform the operations allowed by the NSO.	DRBG	Load Blob - encrypted	Make Blob - encrypted	Ephemeral, stored in volatile RAM	Initialize Unit, Clear Unit
Module key (KM)	AES-256 CBC	Key used to protect logical tokens and associated module Key blobs.	DRBG	Load Blob - encrypted	Make Blob - encrypted	Non-volatile memory – flash (encrypted)	Initialize Unit
KML	3072 DSA	Module signing key. When the nShield module is initialized it automatically generates a 3072-bit DSA key pair that it uses to sign certificates using DSA with SHA-256. This key is only ever used to verify that a certificate was generated by a specific module.	DRBG	N/A	N/A	Non-volatile memory – flash (encrypted)	Initialize Unit
KNSO	3072 DSA	nShield Security Officer key used for NSO authorisation and Security World integrity. Used to sign Delegation Certificates and to directly authorize commands during recovery operations.	DRBG	Load Blob - encrypted	Make Blob - encrypted	Ephemeral, stored in volatile RAM	Initialize Unit
Logical Token (LT)	AES-256 CBC	A logical token is a key used to protect Key blobs. For storage, the logical token is encrypted with Module key (KM) and then split into shares that are stored on smartcard or a softcard encrypted with a share key and optional passphrase.	DRBG	Input from Shares	Output to Shares	Ephemeral, stored in volatile RAM	Clear Unit
Passphrase	N/A	A passphrase (or PIN) is optional and protects a share. It is created by the user at share creation time.	Share creation	Write share	N/A	Ephemeral, stored in volatile RAM	Clear Unit
Remote Administration Session keys	AES-256 CBC, integrity with CMAC	Used for a single session and generated as required by the module between remote smartcards and the module.	ECDH P521 key agreement	N/A	N/A	Ephemeral, stored in volatile RAM	Initialize Unit, Clear Unit, New Session

CSP	Type	Description	Generation	Input	Output	Storage	Zeroization
Share	N/A	A Logical Token (LT) is encrypted with Module key (KM) and then split with the Shamir Threshold Sharing Scheme to create the share. This share is then stored on a smartcard or softcard optionally encrypted by a share key.	Established via the Shamir Secret Sharing protocol	Read Share - encrypted	Write Share - encrypted	Ephemeral, stored in volatile RAM	Write Share, Initialize Unit, Clear Unit, Destroy, Remove a Softcard, Remove a Smartcard
Share Keys	AES-128 CBC, integrity HMAC using SHA-1	Protects a share when written to a smartcard or softcard. Shares are protected by a passphrase (optional).	DRBG	N/A	N/A	Ephemeral, stored in volatile RAM	Remove a Softcard

Table 4 - Critical Security Parameters

Public key	Type	Description	Generation	Input	Output	Storage
Firmware Integrity key (NFIK)	ECDSA P521 with SHA256	Public key used to ensure the integrity of the firmware during boot. The module validates the signature before new firmware is written to flash	At nCipher Security	Firmware Update	No	In Firmware
KJWAR	ECDSA P521	nCipher root warranting public key for Remote Administrator Cards and Remote Operator Cards	At nCipher Security	Firmware Update	None	Encrypted in the module
Application keys - asymmetric public key	See description	Public key associated to the Application keys: - RSA keys - #1897, #1917, #1903 - DSA keys - #1034, #1039 - ECDSA keys - #771, #790, #776 - Key Agreement - #669, #696, #682 - ECMQV - #1111	At creation of the application key	Load Blob - encrypted	Key export	Ephemeral, stored in volatile RAM
KJSO public key	3072 DSA	Public key associated to the KJSO	At creation of the KJSO	Load Blob - encrypted	Key export	Ephemeral, stored in volatile RAM
KNSO public key	3072 DSA	Public key associated to the KNSO	At creation of the KNSO	Load Blob - encrypted	Key export	Ephemeral, stored in volatile RAM
KML public key	3072 DSA	Public key associated to the KML	At creation of the KML, and upon request	No	Key export	Ephemeral, stored in volatile RAM

Public key	Type	Description	Generation	Input	Output	Storage
Impath public key	3072 DH	Public key associated to the Impath keys for creation of the secure channel between two modules	Internally using approved DH key-exchange	Impath Secure Channel (Cmd_ImpathReceive)	Impath Secure Channel (ImpathSend)	Ephemeral, stored in volatile RAM

Table 5 – Public asymmetric Critical Security Parameters

2.2 Cryptographic algorithm

The following cryptographic algorithms are licensable and can be activated through the feature 'Enable Feature':

- Elliptic Curve
- EC MQV
- Accelerated Elliptic Curve
- SCA (Side channel attack) protected algorithms

2.2.1 FIPS approved and allowed

This section describes the algorithm used by the module in FIPS approved mode. Non-approved algorithms are not listed as they cannot be used in FIPS mode.

The following acronyms are used in the table below:

- e - encryption
- d - decryption
- KO 1 - Three -key Triple DES

CAVP #	Algorithm	Standard	Details
Boot Loader			
#3130	SHA	FIPS 180-4	**SHA-256** (BYTE-only) **SHA-512** (BYTE-only)
#805	ECDSA	FIPS 186-4	**SigVer**: CURVES(P-521: (SHA-512)) **SHS:** Val#3130
Firmware			

CAVP #	Algorithm	Standard	Details
#3664	AES	FIPS 197 SP800-38A SP800-38D SP800-38B SP800-38F	**ECB** (e/d; 128 , 192 , 256); **CBC** (e/d; 128 , 192 , 256); **CTR** (int only; 256) **CMAC** (Generation/Verification) (KS: 128; Block Size(s): ; Msg Len(s) Min: 0 Max: 2^16 ; Tag Len(s) Min: 16 Max: 16) (KS: 192; Block Size(s): ; Msg Len(s) Min: 0 Max: 2^16 ; Tag Len(s) Min: 16 Max: 16) (KS: 256; Block Size(s): ; Msg Len(s) Min: 0 Max: 2^16 ;Tag Len(s) Min: 16 Max: 16) **GCM** (KS: AES_128(e/d) Tag Length(s): 128 120 112 104 96 64 32)(KS: AES_192(e/d) Tag Length(s): 128 120 112 104 96 64 32) (KS: AES_256(e/d) Tag Length(s): 128 120 112 104 96 64 32); IV Generated: (Internally (using Section 8.2.2)); PT Lengths Tested:(0 , 1024 , 1024); AAD Lengths tested: (1024 , 1024) ; IV Lengths Tested: (96 , 1024) ; 96BitIV_Supported ; OtherIVLen_Supported; GMAC_Supported **DRBG**: Val# 985 **KW** (AE , AD , AES-128 , AES-192 , AES-256 , FWD , 128 , 256 , 192 , 320 , 4096)
#2046	Triple-DES	SP800-67	**TECB**(KO 1 e/d); **TCBC**(KO 1 e/d)
#3082	SHA	FIPS 180-4	**SHA-1** (BYTE-only); **SHA-224** (BYTE-only); **SHA-256** (BYTE-only); **SHA-384** (BYTE-only);**SHA-512** (BYTE-only)
#2414	HMAC with SHA	FIPS 198-1	**HMAC-SHA1** (Key Sizes Ranges Tested:KS=BS) SHS Val#3082; **HMAC-SHA224** (Key Size Ranges Tested:KS=BS) SHS Val#3082; **HMAC-SHA256** (Key Size Ranges Tested:KS=BS) SHS Val#3082; **HMAC-SHA384** (Key Size Ranges Tested:KS=BS) SHS Val#3082; **HMAC-SHA512** (Key Size Ranges Tested:KS=BS) SHS Val#3082
#1897	RSA	FIPS 186-4	**FIPS186-2:** **ALG[ANSIX9.31]:** Key(gen)(MOD: 2048, 3072) **ALG[ANSIX9.31]:** SIG(ver); 1024 , 1536 , 2048 , 3072 , 4096 SHS: SHA-1 Val#3082, SHA-224 Val#3082, SHA-256 Val#3082, SHA-384 Val#3082, SHA-512 Val#3082 **FIPS186-4:** **186-4KEY(gen):** FIPS186-4_Random_e **PGM(ProbRandom:** (2048 , 3072) **PPTT:**(C.3) **PGM(ProvPrimeCondition)** **ALG[RSASSA-PKCS1_V1_5]** SIG(gen) (2048 SHA(224 , 256 , 384 , 512)) (3072 SHA(224 , 256 , 384 , 512)) SIG(Ver) (1024 SHA(1 , 224 , 256 , 384 , 512)) (2048 SHA(1 , 224 , 256 , 384 , 512)) (3072 SHA(1 , 224 , 256 , 384 , 512)) **[RSASSA-PSS]:** Sig(Gen): (2048 SHA(224 SaltLen(28) , 256 SaltLen(32) , 384 SaltLen(48) , 512 SaltLen(64))) (3072 SHA(224 SaltLen(28) , 256 SaltLen(32) , 384 SaltLen(48) , 512 SaltLen(64))) Sig(Ver): (1024 SHA(1 SaltLen(20) , 224 SaltLen(28) , 256 SaltLen(32) , 384 SaltLen(48))) (2048 SHA(1 SaltLen(20) , 224 SaltLen(28) , 256 SaltLen(32) , 384 SaltLen(48) , 512 SaltLen(64))) (3072 SHA(1 SaltLen(20) , 224 SaltLen(28) , 256 SaltLen(32) , 384 SaltLen(48) , 512 SaltLen(64))) SHA Val#3082, DRBG: Val# 985

CAVP #	Algorithm	Standard	Details
#1034	DSA	FIPS 186-4	**FIPS186-4:** **PQG(gen)PARMS TESTED:** [(2048, 224)SHA(224); (2048,256)SHA(256); (3072,256) SHA(256)] **PQG(ver)PARMS TESTED:** [(1024,160) SHA(1); (2048,224) SHA(224); (2048,256) SHA(256); (3072,256) SHA(256)] **KeyPairGen:** [(2048,224) ; (2048,256) ; (3072,256)] **SIG(gen)PARMS TESTED:** [(2048,224) SHA(224 , 256 , 384 , 512); (2048,256) SHA(256 , 384 , 512); (3072,256) SHA(256 , 384 , 512);] **SIG(ver)PARMS TESTED:** [(1024,160) SHA(1 , 224 , 256 , 384 , 512); (2048,224) SHA(224 , 256 , 384 , 512); (2048,256) SHA(256 , 384 , 512); (3072,256) SHA(256 , 384 , 512)] SHS: Val# 3082, DRBG: Val# 985
#771	ECDSA	FIPS 186-4	**FIPS186-4:** **PKG: CURVES**(P-224 P-256 P-384 P-521 K-233 K-283 K-409 K-571 B-233 B-283 B-409 B-571 ExtraRandomBits) **PKV: CURVES**(ALL-P ALL-K ALL-B) **SigGen: CURVES**(P-224: (SHA-224, 256, 384, 512) P-256: (SHA-256, 384, 512) P-384: (SHA-384, 512) P-521: (SHA-512) K-233: (SHA-224, 256, 384, 512) K-283: (SHA-256, 384, 512) K-409: (SHA-384, 512) K-571: (SHA-512) B-233: (SHA-224, 256, 384, 512) B-283: (SHA-256, 384, 512) B-409: (SHA-384, 512) B-571: (SHA-512)) **SigVer: CURVES**(P-192: (SHA-1, 224, 256, 384, 512) P-224: (SHA-224, 256, 384, 512) P-256: (SHA-256, 384, 512) P-384: (SHA-384, 512) P-521: (SHA-512) K-163: (SHA-1, 224, 256, 384, 512) K-283: (SHA-256, 384, 512) K-409: (SHA-384, 512) K-571: (SHA-512) B-163: (SHA-1, 224, 256, 384, 512) B-233: (SHA-224, 256, 384, 512) B-283: (SHA-256, 384, 512) B-409: (SHA-384, 512) B-571: (SHA-512)) SHS: Val#3082, DRBG: Val# 985
#669	Key Agreement CVL certificate for DH, ECDH & EC MQV	SP800-56A	**FFC:** (FUNCTIONS INCLUDED IN IMPLEMENTATION: KPG Partial Validation) **SCHEMES: Ephem:** (KARole: Initiator / Responder) FB FC **OneFlow:** (KARole: Initiator / Responder) FB FC **Static:** (KARole: Initiator / Responder) FB FC DSA Val#1034, SHS Val#3082, DRBG Val#985 **ECC:** (FUNCTIONS INCLUDED IN IMPLEMENTATION: KPG Partial Validation) **SCHEMES: FullMQV:** (KARole: Initiator / Responder) EB: P-224 EC: P-256 ED: P-384 EE:**EphemUnified:** (KARole: Initiator / Responder) EB: P-224 EC: P-256 ED: P-384 EE: P-521 **OnePassDH:** (KARole: Initiator) EB: P-224 EC: P-256 ED: P-384 EE: P-521 **StaticUnified:** (KARole: Initiator / Responder) EB: P-224 EC: P-256 ED: P-384 EE: P-521 ECDSA Val#771, SHS Val#3082, DRBG Val#985
#73	KBKDF	SP800-108	**CTR_Mode:** (Llength(Min16 Max16) MACSupported([CMACAES256]) LocationCounter([BeforeFixedData]) rlength([8])) AES Val#3664, DRBG Val#985
#985	DRBG	SP800-90A	Hash_**Based** DRBG: [Prediction Resistance Tested: Not Enabled (SHA-256) (SHS Val#3082)]
Main Cryptographic Accelerator			

CAVP #	Algorithm	Standard	Details
#3697	AES	SP800-38D	**CMAC** (Generation/Verification) **(KS: 128;** Block Size(s): ; **Msg Len(s)** Min: 0 Max: 2^16 ; **Tag Len(s)** Min: 16 Max: 16) **(KS: 192**; Block Size(s): ; **Msg Len(s)** Min: 0 Max: 2^16 ; **Tag Len(s)** Min: 16 Max: 16) **(KS: 256**; Block Size(s): ; **Msg Len(s)** Min: 0 Max: 2^16 ;**Tag Len(s)** Min: 16 Max: 16) AES Val#3711 **GCM (KS: AES_128**(e/d) Tag Length(s): 128 120 112 104 96 64 32)**(KS: AES_192**(e/d) Tag Length(s): 128 120 112 104 96 64 32) **(KS: AES_256**(e/d) Tag Length(s): 128 120 112 104 96 64 32) **IV Generated:** (Internally (using Section 8.2.2)) ; **PT Lengths Tested:**(0 , 1024 , 1024) ; **AAD Lengths tested:** (1024 , 1024) ; **IV Lengths Tested:** (96 , 1024) ; **96BitIV_Supported** ; **OtherIVLen_Supported** **GMAC_Supported** DRBG: Val# 985
#3711	AES	FIPS 197 & SP800-138	**ECB** (e/d; 128 , 192 , 256); **CBC** (e/d; 128 , 192 , 256); **CTR** (int only; 256) **KW** (AE , AD , AES-128 , AES-192 , AES-256 , FWD , 128 , 256 , 192 , 320 , 4096)
#2073	Triple-DES	SP800-67	TECB(KO 1 e/d); TCBC(KO 1 e/d)
#75	KBKDF	SP800-108	**CTR_Mode:** (Llength(Min16 Max16) MACSupported([CMACAES256]) LocationCounter([BeforeFixedData]) rlength([8])) AES Val#3664, DRBG Val#985
Firmware Side-Channel			
#1917	RSA	FIPS 186-4	**FIPS186-2:** **ALG[RSASSA-PKCS1_V1_5]:** SIG(ver): 1024 , 1536 , 2048 , 3072 , 4096 SHS: SHA-1 Val#3082, SHA-224 Val#3082, SHA-256 Val#3082, SHA-384 Val#3082, SHA-512 Val#3082 **FIPS186-4:** **186-4KEY(gen):** FIPS186-4_Random_e **PGM(ProbRandom:** (2048, 3072) **PPTT:**(C.3) **ALG[RSASSA-PKCS1_V1_5]** SIG(gen) (2048 SHA(224 , 256 , 384 , 512)) (3072 SHA(224 , 256 , 384 , 512)) SIG(Ver) (1024 SHA(1 , 224 , 256 , 384 , 512)) (2048 SHA(1 , 224 , 256 , 384 , 512)) (3072 SHA(1 , 224 , 256 , 384 , 512)) **[RSASSA-PSS]:** Sig(Gen): (2048 SHA(224 SaltLen(28) , 256 SaltLen(32) , 384 SaltLen(48) , 512 SaltLen(64))) (3072 SHA(224 SaltLen(28) , 256 SaltLen(32) , 384 SaltLen(48) , 512 SaltLen(64))) Sig(Ver): (1024 SHA(1 SaltLen(20) , 224 SaltLen(28) , 256 SaltLen(32) , 384 SaltLen(48))) (2048 SHA(1 SaltLen(20) , 224 SaltLen(28) , 256 SaltLen(32) , 384 SaltLen(48) , 512 SaltLen(64))) (3072 SHA(1 SaltLen(20) , 224 SaltLen(28) , 256 SaltLen(32) , 384 SaltLen(48) , 512 SaltLen(64))) SHA Val#3082, DRBG: Val# 985

CAVP #	Algorithm	Standard	Details
#790	ECDSA	FIPS 186-4	**FIPS186-4:** **PKG: CURVES**(P-224 P-256 P-384 P-521 K-233 K-283 K-409 K-571 B-233 B-283 B-409 B-571 ExtraRandomBits) **PKV: CURVES**(ALL-P ALL-K ALL-B) **SigGen: CURVES**(P-224: (SHA-224, 256, 384, 512) P-256: (SHA-256, 384, 512) P-384: (SHA-384, 512) P-521: (SHA-512) K-233: (SHA-224, 256, 384, 512) K-283: (SHA-256, 384, 512) K-409: (SHA-384, 512) K-571: (SHA-512) B-233: (SHA-224, 256, 384, 512) B-283: (SHA-256, 384, 512) B-409: (SHA-384, 512) B-571: (SHA-512)) **SigVer: CURVES**(P-192: (SHA-1, 224, 256, 384, 512) P-224: (SHA-224, 256, 384, 512) P-256: (SHA-256, 384, 512) P-384: (SHA-384, 512) P-521: (SHA-512) K-163: (SHA-1, 224, 256, 384, 512) K-283: (SHA-256, 384, 512) K-409: (SHA-384, 512) K-571: (SHA-512) B-163: (SHA-1, 224, 256, 384, 512) B-233: (SHA-224, 256, 384, 512) B-283: (SHA-256, 384, 512) B-409: (SHA-384, 512) B-571: (SHA-512)) **SHS:** Val#3082, **DRBG:** Val# 985
#696	Key Agreement CVL certificate for ECDH	SP800-56A	**ECC:** (FUNCTIONS INCLUDED IN IMPLEMENTATION: KPG Partial Validation) **SCHEMES: EphemUnified:** (KARole: Initiator / Responder) EB: P-224 EC: P-256 ED: P-384 EE: P-521 **OnePassDH:** (KARole: Initiator / Responder) EB: P-224 EC: P-256 ED: P-384 EE: P-521**StaticUnified:** (KARole: Initiator / Responder) EB: P-224 EC: P-256 ED: P-384 EE: P-521 ECDSA Val#790, SHS Val#3082, DRBG Val#985
Cryptographic Accelerator Library			
#1903	RSA	FIPS 186-4	**FIPS186-2:** **ALG[RSASSA-PKCS1_V1_5]:** SIG(ver): 1024 , 1536 , 2048 , 3072 , 4096 SHS: SHA-1 Val#3082, SHA-224 Val#3082, SHA-256 Val#3082, SHA-384 Val#3082, SHA-512 Val#3082 **FIPS186-4:** **186-4KEY(gen):** FIPS186-4_Random_e **PGM(ProbRandom:** (2048 , 3072) **PPTT:**(C.3) **ALG[RSASSA-PKCS1_V1_5]** SIG(gen) (2048 SHA(224 , 256 , 384 , 512)) (3072 SHA(224 , 256 , 384 , 512)) SIG(Ver) (1024 SHA(1 , 224 , 256 , 384 , 512)) (2048 SHA(1 , 224 , 256 , 384 , 512)) (3072 SHA(1 , 224 , 256 , 384 , 512)) **[RSASSA-PSS]:** Sig(Gen): (2048 SHA(224 SaltLen(28) , 256 SaltLen(32) , 384 SaltLen(48) , 512 SaltLen(64))) (3072 SHA(224 SaltLen(28) , 256 SaltLen(32) , 384 SaltLen(48) , 512 SaltLen(64))) Sig(Ver): (1024 SHA(1 SaltLen(20) , 224 SaltLen(28) , 256 SaltLen(32) , 384 SaltLen(48))) (2048 SHA(1 SaltLen(20) , 224 SaltLen(28) , 256 SaltLen(32) , 384 SaltLen(48) , 512 SaltLen(64))) (3072 SHA(1 SaltLen(20) , 224 SaltLen(28) , 256 SaltLen(32) , 384 SaltLen(48) , 512 SaltLen(64))) SHA Val#3082, DRBG: Val# 985

CAVP #	Algorithm	Standard	Details
#1039	DSA	FIPS 186-4	**FIPS186-4:** **PQG(gen)PARMS TESTED:** [(2048, 224)SHA(224); (2048,256)SHA(256); (3072,256) SHA(256)] **PQG(ver)PARMS TESTED:** [(1024,160) SHA(1); (2048,224) SHA(224); (2048,256) SHA(256); (3072,256) SHA(256)] **KeyPairGen:** [(2048,224) ; (2048,256) ; (3072,256)] **SIG(gen)PARMS TESTED:** [(2048,224) SHA(224 , 256 , 384 , 512); (2048,256) SHA(256 , 384 , 512); (3072,256) SHA(256 , 384 , 512);] **SIG(ver)PARMS TESTED:** [(1024,160) SHA(1 , 224 , 256 , 384 , 512); (2048,224) SHA(224 , 256 , 384 , 512); (2048,256) SHA(256 , 384 , 512); (3072,256) SHA(256 , 384 , 512)] SHS: Val# 3082, DRBG: Val# 985
#776	ECDSA	FIPS 186-4	**FIPS186-4:** **PKG: CURVES**(P-224 P-256 P-384 P-521 K-233 K-283 K-409 K-571 B-233 B-283 B-409 B-571 ExtraRandomBits) **PKV: CURVES**(ALL-P ALL-K ALL-B) **SigGen: CURVES**(P-224: (SHA-224, 256, 384, 512) P-256: (SHA-256, 384, 512) P-384: (SHA-384, 512) P-521: (SHA-512) K-233: (SHA-224, 256, 384, 512) K-283: (SHA-256, 384, 512) K-409: (SHA-384, 512) K-571: (SHA-512) B-233: (SHA-224, 256, 384, 512) B-283: (SHA-256, 384, 512) B-409: (SHA-384, 512) B-571: (SHA-512)) **SigVer: CURVES**(P-192: (SHA-1, 224, 256, 384, 512) P-224: (SHA-224, 256, 384, 512) P-256: (SHA-256, 384, 512) P-384: (SHA-384, 512) P-521: (SHA-512) K-163: (SHA-1, 224, 256, 384, 512) K-283: (SHA-256, 384, 512) K-409: (SHA-384, 512) K-571: (SHA-512) B-163: (SHA-1, 224, 256, 384, 512) B-233: (SHA-224, 256, 384, 512) B-283: (SHA-256, 384, 512) B-409: (SHA-384, 512) B-571: (SHA-512)) **SHS:** Val#3082, **DRBG:** Val# 985
#682	Key Agreement CVL certificate for DH & ECDH	SP800-56A	**FFC:** (FUNCTIONS INCLUDED IN IMPLEMENTATION: KPG Partial Validation) **SCHEMES: Ephem:** (KARole: Initiator / Responder) FB FC **OneFlow:** (KARole: Initiator / Responder) FB FC **Static:** (KARole: Initiator / Responder) FB FC DSA Val#1039, SHS Val#3082, DRBG Val#985 **ECC:** (FUNCTIONS INCLUDED IN IMPLEMENTATION: KPG Partial Validation) **SCHEMES: EphemUnified:** (KARole: Initiator / Responder) EB: P-224 EC: P-256 ED: P-384 EE: P-521 **OnePassDH:** (KARole: Initiator / Responder) EB: P-224 EC: P-256 ED: P-384 EE: P-521**StaticUnified:** (KARole: Initiator / Responder) EB: P-224 EC: P-256 ED: P-384 EE: P-521 ECDSA Val#776, SHS Val#3082, DRBG Val#985
#1111	Key Agreement CVL certificate for EC MQV	FIPS 186-4	**ECC:** (FUNCTIONS INCLUDED IN IMPLEMENTATION: KPG Partial Validation) **SCHEMES: FullMQV:** (KARole: Initiator / Responder) EB: P-224 EC: P-256 ED: P-384 EE: P-521 ECDSA Val#776, SHS Val#3082, DRBG Val#985
Others			
N/A	NDRNG	N/A	Allowed Non-Deterministic Random Number Generator (NDRNG). NDRNG is used to seed the approved DRBG
N/A	PBKDF	SP800-132	Vendor affirmed PBKDF implementation. Algorithm used only for storage of the EncFSKey
N/A	Diffie-Hellman	N/A	CVL Cert. #669, #682, key agreement
N/A	EC Diffie-Hellman	N/A	CVL Cert. #669, #682, #696; key agreement

CAVP #	Algorithm	Standard	Details
N/A	EC MQV	N/A	CVL Cert. #669, #1111; key agreement

Table 6 - Overview of FIPS Approved and allowed algorithms

3. Roles and Services

3.1 Authentication enforcement

The module provides identity-based authentication, as described in chapter 5.1, and furthermore, the strength of the functions involved are described in chapter 5.1.1. A four second delay is implemented upon an unsuccessful authentication attempt, thereby dramatically increasing the time required to brute-force module authentication.

3.2 Roles

This section described the roles supported by the module. Note a user is "Unauthenticated" prior to authentication.

3.2.1 User

User role (USR) corresponds to the FIPS User role, as defined in FIPS 140-2. The exact accreditation and operation required to perform each service is listed in the table of services below.

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. 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 an nShield Security Officer authorizing the action. Some actions including writing tokens always require a certificate.

3.2.2 nShield Security Officer

The nShield Security Officer role (NSO) corresponds to the FIPS Crypto Officer role, as defined in FIPS 140-2. An operator assumes the role of NSO by loading the private half of KNSO and presenting the ObjectID for this key to authorize a command.

The NSO is identified by a key pair, referred to as KNSO. 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 KNSO. The NSO is responsible for creating the authentication tokens (smart cards) for each operator and ensuring that these tokens are physically handed to the correct person.

3.2.3 Junior Security Officer

Junior Security Officer role (JSO) is a delegated role created by the NSO for authorizing an action. In order to assume the role of JSO, the operator loads a key corresponding to a service that is delegated by the NSO and presents the handle of this key, and if required the certificate signed by KNSO that delegates authority to the key, to authorize a command.

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. 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.

3.3 Services vs Roles

The table below presents the Services of the product with a mapping with the Roles.
The Access column presents the access level given to the CSP, R for Read, W for Write, D for Delete

Service	Description	Authorized roles	Access	CSPs
Big number operation Cmd_BignumOp	Performs an operation on an integer.	Unauthenticated	-	None
Buffer operations Cmd_CreateBuffer Cmd_LoadBuffer	Mechanism for loading of data into the module volatile memory. The data can be loaded in encrypted form which can be decrypted inside the module with a key that has been previously loaded.	Unauthenticated	R	Key blob
Bulk channel Cmd_ChannelOpen Cmd_ChannelUpdate	Provides a bulk processing channel for encryption or decryption using symmetric key algorithms.	User	R	Application keys (KA)
Change Share Passphrase Cmd_ChangeSharePIN	Updates the passphrase of a Share.	NSO / JSO / User	W	Passphrase, Share
Check User Action Cmd_CheckUserAction	Determines whether the ACL associated with a Key blob allows a specific operator defined action.	User / JSO / NSO	R	Key blob; KNSO, KJSO; KA
Clear Unit Cmd_ClearUnit	Zeroises all keys, tokens and shares that are loaded into the module. Will cause the module to reboot and perform self-tests. The CSP that are keeping on a smartcard for instance are not erased. When a module is power cycle, a clear unit command is sent to the module.	Unauthenticated	D	Application keys (KA), DRBG internal state, EncFSKey, Impath Keys, Key Blob, KJSO, Logical Token (LT), Passphrase, Remote Administration keys, Share
Derive Key Cmd_DeriveKey	Key wrapping, unwrapping. The ACL needs to authorize this operation.	NSO / JSO / User	R	Application keys (KA)
Destroy Cmd_Destroy	Remove handle to an object in SRAM. If the current handle is the only one remaining, the object is deleted from SRAM. This action deletes the element from the memory	Unauthenticated	D	Application keys (KA), Logical Token, Share, Impath keys
Duplicate key handle Cmd_Duplicate	Creates a second instance of a Key with the same ACL and returns a handle to the new instance. Note that the source key ACL needs to authorize this operation.	User / JSO / NSO	R	Application keys (KA)
Enable feature Cmd_StaticFeatureEnable	Enables the service. This service requires a certificate signed by the Master Feature Enable key.	Unauthenticated	-	None
Encryption / decryption Cmd_Encrypt Cmd_Decrypt	Encryption and decryption using the provided key handle.	User	R	Application keys (KA)
Erase from smartcard /softcard Cmd_EraseFile Cmd_EraseShare	Removes a file or a share and delete the CSP from memory, it does not erase the share keys	NSO / JSO / User	D	Share keys
File operations Cmd_FileCopy Cmd_FileCreate Cmd_FileErase Cmd_FileOp	Performs file operations in the module.	NSO / JSO	-	None
Firmware Authenticate Cmd_FirmwareAuthenticate	Reports firmware version, using a zero knowledge challenge response protocol based on HMAC The protocol generates a random value to use as the HMAC key.	Unauthenticated	-	None

Service	Description	Authorized roles	Access	CSPs
Firmware Update Cmd_ProgrammingBegin Cmd_ProgrammingBeginChunk Cmd_ProgrammingLoadBlock Cmd_ProgrammingEndChunk Cmd_ProgrammingEnd Cmd_ProgrammingGetKeyList	Perform a firmware update. Restricted service to nCipher signed Firmware.	NSO	R	Firmware Confidentiality Key (KFC), Firmware Integrity key (NFIK), KJWAR
Force module to fail Cmd_Fail	Causes the module to enter a failure state.	Unauthenticated	-	None
Foreign Token command Cmd_ForeignTokenCommand	Sends an ISO-7816 command to a smart card over the channel opened by ForeignTokenOpen.	Unauthenticated	R	Logical Token
Foreign Token open Cmd_ForeignTokenOpen	Opens a channel for direct data access to a Smartcard Requires Feature Enabled.	NSO / JSO	R	Logical Token
Format Token Cmd_FormatToken	Formats a smart card or a software token.	NSO / JSO	-	None
Generate Logical Token Cmd_GenerateLogicalToken	Creates a new Logical Token with given properties and secret sharing parameters.	NSO, JSO	W	Module key (KM), Logical Token
Generate prime number Cmd_GeneratePrime	Generates a random prime number.	Unauthenticated	R	DRBG internal state
Generate random number Cmd_GenerateRandom	Generates a random number from the Approved DRBG.	Unauthenticated	R	DRBG internal state
Get ACL Cmd_GetACL	Get the ACL of a given handle.	User	R	Application keys (KA)
Get challenge Cmd_GetChallenge	Get a random challenge that can be used in fresh certificates.	Unauthenticated	R	DRBG internal state
Get key application data Cmd_GetApplicationData	Get the application data field from a key.	User	R	Application keys (KA)
Get Key Information Cmd_GetKeyInfo Cmd_GetKeyInfoEx	Get the type, length and hash of a key.	NSO / JSO / User	R	Application keys (KA)
Get list of module keys Cmd_GetKMList	Get the list of the hashes of all module keys and the KNSO.	Unauthenticated	-	None
Get list of slot in the module Cmd_GetSlotList	Get the list of slots that are available from the module.	Unauthenticated	-	None
Get Logical Token Info Cmd_GetLogicalTokenInfo Cmd_GetLogicalTokenInfoEx	Get information about a Logical Token: hash, state and number of shares.	NSO / JSO / User	R	Logical Token (LT)
Get module signing long term key Cmd_GetKML	Get a handle to the KML public key.	Unauthenticated	R	KML (public)
Get module state Cmd_GetModuleState	Returns unsigned data about the current state of the module.	Unauthenticated	-	None

Service	Description	Authorized roles	Access	CSPs
Get real time clock Cmd_GetRTC	Get the current time from the module Real Time Clock.	Unauthenticated	-	None
Get share access control list Cmd_GetShareACL	Get the Share's ACL.	NSO / JSO / User	R	Share key
Get Slot Information Cmd_GetSlotInfo	Get information about shares and files on a Smartcard that has been inserted in a module slot.	Unauthenticated	-	None
Get Ticket Cmd_GetTicket	Get a ticket (an invariant identifier) for a key. This can be passed to another client or to a SEE World which can redeem it using Redeem Ticket to obtain a new handle to the object.	NSO / JSO / User	-	None
Impath secure channel Cmd_ImpathGetInfo Cmd_ImpathKXBegin Cmd_ImpathKXFinish Cmd_ImpathReceive Cmd_ImpathSend	Support for Impath secure channel. Requires Feature Enabled. Cmd_ImpathKXBegin and Cmd_ImpathKXFinish will create the Impath Key between two modules. Cmd_ImpathGetInfo, Cmd_ImpathReceive and Cmd_ImpathSend uses the impath key	NSO / JSO / User	R, W	Impath key
Initialize Unit Cmd_InitializeUnit Cmd_InitializeUnitEx	Causes a module in the pre-initialization state to enter the initialization state. When the module enters the initialization state, it erases all Module keys (KM). It also erases the module's signing key, KML, and the hash of the Security Officer's keys, HKNSO. It then generates a new KML and KM.	Unauthenticated	D, W	(D, W): Module Key (KM), KML (D): Application keys (KA), Archiving Keys, Impath keys, KJSO, KNSO, Share, Remote Administration Session keys
Insert a Softcard Cmd_InsertSoftToken	Allocates memory on the module that is used to store one or more logical shares or other Token data objects.	Unauthenticated	R	Share, Share key
Key export Cmd_Export	Exports a public key in plain text.	NSO / JSO / User	R	Application keys (KA), KJSO, KNSO, KML
Key generation Cmd_GenerateKey Cmd_GenerateKeyPair	Generates a cryptographic key of a given type with a specified ACL. It returns a handle to the key or a State Blob with the key. Optionally, it returns a KML signed certificate with the ACL information.	NSO, JSO	W	Key blob, DRBG internal state, Application keys (KA), KJSO
Key import Cmd_Import	Loads a plain text key into the module. If the module is initialized in FIPS 140-2 level 3 mode, this service is available for public keys only.	NSO, JSO	R	Key blob, Application keys (KA)
Load Blob Cmd_LoadBlob	Load a Key blob into the module. It returns a handle to the key suitable for use with module services.	NSO / JSO / User	R	Application keys (KA), Archiving keys (KR), Key blob, KJSO, Module key (KM), KNSO
Load Logical Token Cmd_LoadLogicalToken	Initiates loading a Logical Token from Shares, which can be loaded with the Read Share command.	Unauthenticated	R	Module key (KM), Logical Token

Service	Description	Authorized roles	Access	CSPs
Make Blob Cmd_MakeBlob	Creates a Key blob containing the key. Note that the key ACL needs to authorize the operation.	User / JSO / NSO	W	Application keys (KA), KNSO, Archiving keys (KR), Key blob, KJSO, Module key (KM), Logical Token (LT)
Message digest Cmd_Hash	Computes the cryptographic hash of a given message.	Unauthenticated	-	None
Modular Exponentiation Cmd_ModExp Cmd_ModExpCrt Cmd_RSAImmedSignEncrypt Cmd_RSAImmedVerifyDecrypt	Performs a modular exponentiation (standard or CRT) on values supplied with the command.	Unauthenticated	-	None
Module hardware information Cmd_ModuleInfo	Reports detailed hardware information.	Unauthenticated	-	None
No Operation Cmd_NoOp	No operation.	Unauthenticated	-	None
NVRAM Allocate Cmd_NVMemAllocate	Allocation in NVRAM.	NSO	-	None
NVRAM Free Cmd_NVMemFree	Deallocation from NVRAM.	NSO	-	None
Operation on NVM files Cmd_NVMemOp	Returns a list of files in NVRAM.	Unauthenticated	-	None
Operation on NVM list Cmd_NVMemList	Returns a list of files in NVRAM.	Unauthenticated	-	None
Power On	Power on of the module, re initialize the DRBG internal state and reconstruct the EncFSKey	Unauthenticated	W	EncFSKey
Read file Cmd_ReadFile	Reads data from a file on a Smartcard or Softcard. The ACL needs to authorize this operation.	NSO / JSO	-	None
Read share Cmd_ReadShare	Reads a share from a Smartcard or Softcard. Once a quorum of shares have been loaded, the module re-assembles the Logical Token.	NSO / JSO / User	R	Passphrase, Share Keys, Logical Token (LT)
Receive share from remote slot Cmd_ReceiveShare	Receives a Share encrypted with the Impath session keys by a remote module.	NSO / JSO / User	R	Impath key, passphrase, Application keys (KA), Share Keys
Redeem Ticket Cmd_RedeemTicket	Gets a handle in the current name space for the object referred to by a ticket created by Get Ticket.	NSO / JSO / User	R	Key blob, Logical Token (LT), Impath key
Remote Administration Cmd_DynamicSlotCreateAssociation Cmd_DynamicSlotExchangeAPDUs Cmd_DynamicSlotsConfigure Cmd_DynamicSlotsConfigure Cmd_DynamicSlotsConfigureQuery Cmd_VerifyCertificate	Provides remote presentation of Smartcards using a secure channel between the module and the Smartcard. The KJWAR is used in the operation (R), the remote administration session keys are created (W)	NSO / JSO / User	R & W	Remote administration session keys, KJWAR

Service	Description	Authorized roles	Access	CSPs
Remove a Softcard Cmd_RemoveSoftToken	Removes a Softcard from the module. It returns the updated shares and deletes them from the module's memory. The removal does not erase the CSP from the softcard.	Unauthenticated	D	Share, Share key
Report statistics Cmd StatGetValues Cmd_StatEnumTree	Reports the values of the statistics tree.	Unauthenticated	-	None
Secure Execution Environment Cmd_CreateSEEWorld Cmd_GetWorldSigners Cmd_SEEJob Cmd_SetSEEMachine Cmd_TraceSEEWorld	Creation and interaction with SEE machines.	NSO	-	None
Send share to remote slot Cmd_SendShare	Reads a Share and encrypts it with the Impath session keys for transmission to the peer module.	NSO / JSO / User	R	Impath key, Share Keys
Set ACL Cmd_SetACL	Replaces the ACL of a given Key blob with a new ACL. The ACL needs to authorize this operation.	NSO / JSO / User	R	Application keys (KA)
Set key application data Cmd_SetAppData	Writes the application information field of a key.	User	W	Application keys (KA)
Set Module Key Cmd_SetKM	Allows a Key blob to be stored internally as a Module key (KM) value. The ACL needs to authorize this operation.	NSO / JSO	R	Key blob, Module key (KM)
Set NSO Permissions Cmd_SetNSOPerm	Sets the NSO key hash and which permissions required a Delegation Certificate.	NSO	R	KNSO
Set real time clock Cmd_SetRTC	Sets the Real-Time Clock value.	JSO	-	None
Show Status Cmd_NewEnquiry	Report status information.	Unauthenticated	-	None
Sign Module State Cmd_SignModuleState	Returns a signed certificate that contains data about the current configuration of the module.	Unauthenticated	R	KML
Signature generation Cmd_Sign	Generate a digital signature or MAC value.	NSO / JSO / User	R	Application keys (KA)
Signature verification Cmd_Verify	Verifies a digital signature or MAC value.	NSO / JSO / User	R	Application keys (KA)
Write file Cmd_WriteFile	Writes a file to a Smartcard or Softcard.	Unauthenticated	-	None
Write share Cmd_WriteShare	Writes a Share to a Smartcard or Softcard.	NSO / JSO	W	Share

Table 7 - Services vs Roles

4. Physical Security

4.1 Physical security overview

The product is a multiple-chip embedded cryptographic module as described in FIPS 140-2. It is design to meet FIPS 140-2 Security level 3 requirement. All components within the defined cryptographic boundary of the module, except the physical interfaces, are covered by an opaque epoxy resin, and opaque solid metal heat sink.

PCI card on which the module resides, has a clear button. Pressing this button puts the module into the self-test state, clearing all application keys (KA), stored Key blobs, Logical Tokens and Impath keys and running all self-tests. The NSO's key, module keys and module signing key (KML) can be cleared by returning the module to the factory state, as described above.

4.2 Checking the module

To ensure physical security, the following checks are required to be made regularly:

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

5. Rules

5.1 Identification and authentication

Communication with the modules is performed via a server program running on the host machine, using standard inter process communication, sockets on UNIX and pipes on Windows. The operator must log on to the host computer and start an nShield enabled application to use the module. The application connects to the hard 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 vs Roles”. An operator cannot access any service that accesses CSPs without first presenting a smartcard, or software token.

5.1.1 Strength of authentication

KNSO, KJSO and application keys are stored encrypted with Module key (KM) and encrypted with an associated logical token for storage on the host side. The associated logical token is encrypted with KM and divided into shares and then encrypted by a share key on smartcard or softcard plus an optional passphrase for each share. The share is distributed to the quorum of users.

The authentication requires the following:

- Presentation of the quorum of shares to the HSM by the quorum of smartcards or softcard holders to reconstruct the encrypted logical token. Each share is protected by an AES-128 key, optional passphrase and associated MAC values.
- The encrypted logical token is then decrypted by the HSM to validate that the logical token belongs to the security world.
- The usage of the key is granted.

The protection of the authentication mechanism meets the requirement of 10^6 resistance for authentication mechanisms. In effect, an attacker would need to brute force the MAC protection (2^160) and brute force the share which is protected by an AES-key (2^128).

5.1.2 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 logical token used to encrypt this blob.

The Key blob also contains an ACL 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 a client, identified by a unique ClientID. 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 clients.

5.1.3 Access Control List

An Access Control List or ACL is associated to all Key blobs and describes the operations that can be performed by its associated key. Created by the operator during the generation or import of the key and stored with the key enciphered and MAC when it is on the blob format.

When the blob is reloaded the ACL applies to the new Key blob created. It 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 blob created when you reload the blob. ACL design is complex - operators will not normally need to write ACLs themselves. nCipher provide tools to generate keys with strong ACLs.

Usage	Description
Limits on operation	The ACL can specify limits on operation or groups of operations. Those limits may be global limits or per authorization limits: - If a global limit is exceeded, then the key cannot be used for that operation - If a per authorization limit is exceeded then the Logical Token protecting the key must be reloaded before the key can be reused
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
List operator defined actions	These actions do not permit any operations within the module, but can be tested with the **CheckUserACL** service. This enables SEE programs to make use of the ACL system for their own purposes
Specify a host service identifier	An ACL can also specify a host service identifier. In which case the ACL is only met if the hard 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.

Table 8 - ACL usage

5.1.4 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.

5.1.5 Error conditions

If the module encounters an unrecoverable error it enters in the error state. The error state is indicated by the status LED flashing in the Morse pattern SOS. As soon as the unit enters the error state all processors stop processing commands and no further replies are returned. In the error state the unit does not respond to commands. The unit must be reset. Upon reset, the keys and CSPs are zeroized.

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

5.1.6 Status information

The module sends a signal to the status LED on the PCI card that indicates the overall state of the module. The module also returns a status message in the reply to every command. This indicates the status of that command.

There are a number of services that report status information (e.g. **new enquiry**), for more information refer to the services table.

5.1.7 Create a new operator

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 passphrases used and the Key blob

nCipher supplies a graphical user interface, called KeySafe, and a command line tool called new-world, that can help automate these steps.

5.1.8 Authorize the operator to create keys

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 NSO's key that:
   - includes the hash of this key as the certifier
   - authorizes the action **GenerateKey** or **GenerateKeyPair** depending on the type of key required
4. If the operator needs to create permanent - as opposed to session - keys, the certificate must also include an entry that enables the action **MakeBlob.** The certificate can restrict the operator to only making blobs protected by their Operator Card Set by including the hash of its Logical Token
5. Give the operator the Key blob and certificate

nCipher supplies a graphical user interface, called KeySafe, and a command line tool called new-world, that can help automate these steps.

5.1.9 Authorize an operator to act as a Junior Security Officer

To authorize an operator to act as a Junior Security Officer:

1. Generate a Logical Token to use to protect the JSO's key
2. Write one or more shares of this token onto software tokens
3. Create a new key pair
   - Give the private half an ACL that permits Sign and UseAsSigningKey
   - Give the public half an ACL that permits ExportAsPlainText
   - Export the private half of the JSO's key as a Key blob under this token
   - Export the public half of the JSO's key as plain text
4. Create a certificate signed by the NSO's key that includes the hash of this key
   - Authorizes the actions GenerateKey, GenerateKeyPair
   - Authorizes the actions **GenerateLogicalToken**, **WriteShare** and **MakeBlob**, these may be limited to a particular module key
5. Give the JSO the software token, any passphrases used, the Key blob and certificate

nCipher supplies a graphical user interface, called KeySafe, and a command line tool called new-world, that can help automate these steps.

5.1.10 Authenticate an operator to use a stored key

To authenticate an operator to use a stored key:

1. Use the **LoadLogicalToken** service to create the space for a Logical Token
2. Use the **ReadShare** service to read each share from the software token
3. Use the **LoadBlob** service to load the key from the Key blob

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

To assume NSO role, load the NSO's key using this procedure. The NSO's key can then be used in certificates authorizing further operations.

nCipher supplies a graphical user interface, called KeySafe, and a command line tool called new-world, that can help automate these steps.

5.1.11 Authenticate an operator to create a new key

To authenticate an operator to create a new key:

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

nCipher supplies a graphical user interface, called KeySafe, and a command line tool called new-world, that can help automate these steps.

5.2 Delivery and operation

5.2.1 Delivery

Solo XC modules are sent to the customers using a standard carrier service. After accepting delivery of the module, the Crypto Officer shall perform an inspection of the module as per section 4.2 of this Security Policy. This inspection is done to ensure that the module has not been tampered with during transit. If the inspection results indicate that the module has not been tampered with, the Crypto Officer can then proceed with installation and configuration of the module.

The module must be installed and configured according to the instructions provided in Section 5.2.3 of this security policy document.

Once the module hardware has been installed, the user must install the support software and initialize the module as described in the User Guide. There are separate versions of this guide for every operating system.

For detailed information on how to configure the module, including how to upgrade its software please refer to the User Guide. For information on how to develop applications including SEE (Secure Execution Engine) applications, please refer to the CodeSafe Developers Guide.

5.2.2 MOI switch

The MOI switch facilitates specific services of the module depending on the switch setting:

- **Initialization**
  - The module is in 'Initialization', by physically moving the switch to the 'I' setting and use the Clear Unit command / service to clear the module, or invoke the Clear Unit command / service using a command line utility specifying Initialization as a parameter. In order to restore the module to 'Operational' you must put the switch back to 'O'
- **Operational**
  - Normal operation of the module
- **Monitor**
  - To put the module in 'Monitor' you must have physical access to the module and put the switch in 'M' setting. In order to restore the module to 'Operational', the module must be reinitialized and then return it to 'Operational'

5.2.3 Initialization procedures

The nShield enabled application must perform the following services to initialize a module to comply with FIPS 140-2 Level 3 (for more information refer to the nShield User Guide):

1. Put the mode switch into the initialization position and restart the module
2. Use either the graphical user interface tool called KeySafe, (ensuring to set the "FIPS 140 Mode level 3 compliant" flag to Yes) or the command line tool **new-world**, specifying the -F flag.
3. Using either tool specify the number of cards in the ACS and specify the encryption algorithm to use as AES.
4. The tool will prompt you to insert cards and enter passphrases for each ACS card.
5. When you have created all the cards, reset the mode switch into the operational position and restart the module.

5.2.4 FIPS mode verification

An operator can verify the initialization status of the module as if a module is initialized in level 3 mode, in the following ways:

- Using Keysafe:
  - Keysafe displays "Strict FIPS 140-2 Level 3 = Yes" in the information panel for that module
- Command line:
  - The command line tool **nfkminfo** includes **StrictFIPS** in the list of flags for the module

5.2.5 Return a module to factory state

To return a module to the factory state, perform the following steps:

1. Put the mode switch into the initialization position
2. Pull the Initialization pin high and restart the module
3. Use the Initialize command to enter the Initialization state
4. Load a random value to use as the hash of the NSO's key
5. Set NSO service to set the NSO's key and the operational policy of the module
6. Put the mode switch into the operational position Pull the Initialization pin low and restart the module

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

Placing the module in factory state, has the following impact:

- Destroys any loaded Logical tokens, Share Keys, Impath keys, Key blobs, Session keys
- Erases the current Module Signing Key (KML) and generates a fresh one
- Erases all current Module Keys, except the Well Known Module Key
- Generates a new Module Key Zero
- Sets NSO's key to a known value

Placing the module in factory state prevents the module from loading any keys previously associated with the module, as it no longer possesses the decryption key. Returning the module to factory state does not erase the Firmware Confidentiality Key or the public halves of the Firmware Integrity Key.

nCipher supplies a graphical user interface, called KeySafe, and a command line tool called new-world, that can help automate these steps.

6. Self-tests

6.1 Power up self-test

When power is applied to the module it enters the self-test state. The module also enters the self-test state whenever the unit is reset, by pressing the clear button or by sending the Clear Unit command, with no further intervention from the operator. During self-test, cryptographic operations are prevented and data output is inhibited.

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

- Operational test on hardware components (memory tests and real time clock tests)
- KAT test for the bootloader library (See **Known Answer Tests**)
- Integrity check of the following image:
  - Secure bootloader - SHA512
  - Firmware - ECDSA P521 NIST
  - Security processor - SHA 256
- DRBG health-checks tests on the random number generator
- Cryptographic algorithm **Known Answer Tests** self-tests as required by FIPS 140-2 (see next paragraph)

6.2 KAT Test

The module performs the following **KAT tests** - If any of these test fails, the module enters in the error state.

Certificates	Algorithm	KAT test
Boot Loader		
#3130	SHA256 & SHA512	KAT tests for SHA256 and SHA512
#805	ECDSA	KAT test verify only for ECDSA P521
Firmware		
#3664	AES	KAT AES ECB encryption and decryption for all keys sizes
#2046	Triple DES 3K & 2K	KAT TDES ECB encryption and decryption
#3082	SHA	SHA1 KAT test, other size are tested along with KAT HMAC
#2414	HMAC with SHA	All KAT HMAC implemented
#1897	RSA	RSA sign and verify 2048 bits
#1034	DSA	DSA KAT tests signature and verification key sizes 2048
#771	ECDSA	ECDSA KAT tests signature and verification for prime and binary curves
#669	Key Agreement	Primitive Z for point multiply (ECDH) and modular exponentiation (DH) KDF KAT covered by #3082 DSA KAT covered by #1034 and ECDSA KAT covered by #771
#73	KBKDF	Covered by AES KAT tests #3664 and DRBG #985
#985	DRBG	SP 800-90A health tests
Main Cryptographic Accelerator		
#3711	AES	KAT AES ECB encryption and decryption for all keys sizes
#2073	Triple-DES 3K & 2K	KAT TDES ECB encryption and decryption
#75	KBKDF	Covered by AES KAT tests and DRBG
Firmware Side-Channel		
#1917	RSA	RSA sign and verify 2048 bits
#790	ECDSA	ECDSA KAT tests signature and verification for prime and binary curves
#696	Key Agreement	Primitive Z for point multiply (ECDH) KDF KAT covered by #3082 ECDSA KAT covered by #790

Certificates	Algorithm	KAT test
Cryptographic Accelerator Library		
#1903	RSA	RSA sign and verify 2048 bits
#1039	DSA	DSA KAT tests signature and verification key sizes 2048
#776	ECDSA	ECDSA KAT tests signature and verification for prime and binary curves
#682	Key Agreement	Primitive Z for point multiply (ECDH) and modular exponentiation (DH) KDF KAT covered by #3082 DSA KAT covered by #1039 and ECDSA KAT covered by #776
#1111	CVL	KAT test covered by #776 KAT tests
Other		
	NDRNG	Continuous randomicity check

Table 9 – KAT Tests table

6.3 Pairwise consistency tests

Whenever the **GenerateKeyPair** command is used (DSA, RSA and ECDSA), the module performs a pairwise consistency check as part of the key generation process.

The private half of the key is used to create a signature which is then verified using the public half. If the signature verification fails, the **GenerateKeyPair** command returns command block with the status Fail and no data.

6.4 Firmware Load Test

The firmware is operating in the approved mode before firmware is allowed to be loaded. When new firmware is loaded, 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 NFIK
- Verify the package using the public portion of the KFC
- Verify the Secure Boot image integrity (if present in package) using the public portion of the NFIK
- Verify the Firmware image integrity using the public portion of the NFIKs
- The image is encrypted with the KFC stored in the module
- The Version Security Number for the image is at least as high as the stored value

Only if all tests pass is the new firmware written to permanent storage. Updating the firmware clears the NSO's key and all stored module keys. The module will not re-enter operational mode until the Crypto Officer has correctly re-initialized it. Note that if the module's firmware is updated to a different version, this results in the loss of the current CMVP validation of the module.

When firmware is updated, the module verifies an ECDSA P521 (certificate #771) signature on the new firmware image before it is written to flash.

Contact Us

Web site: https://www.ncipher.com
Help Centre: https://help.ncipher.com
Email Support: support@ncipher.com

Depending on your geographic location, you can also contact us as follows:

Americas	Asia Pacific	Europe, Middle East and Africa
nCipher Security LLC Sawgrass Corporate Center, Building A 13800 Northwest 14th Street, Suite 130 Sunrise, FL 33323	**nCipher Security (Hong Kong) Limited** 9/F, V-Point 18 Tang Lung Street Causeway Bay, Hong Kong	**nCipher Security Ltd** One station Square Cambridge, UK CB1 2GA
Toll Free: +1 833 425 1990 Fort Lauderdale: +1 954 953 5229	Hong Kong: +852 3461 3088 Japan: +81 50 3196 4994	United Kingdom: +44 1223 723 711

About nCipher Security

Today's fast moving digital environment enhances customer satisfaction, gives competitive advantage and improves operational efficiency. It also multiplies the security risks. nCipher Security, a leader in the general purpose hardware security module (HSM) market, empowers world-leading organizations by delivering trust, integrity and control to their business critical information and applications.

Our cryptographic solutions secure emerging technologies – cloud, IoT, blockchain, digital payments – and help meet new compliance mandates, using the same proven technology that global organizations depend on today to protect against threats to their sensitive data, network communications and enterprise infrastructure. We deliver trust for your business critical applications, ensuring the integrity of your data and putting you in complete control – today, tomorrow, at all times. www.ncipher.com