nShield Key Attestation

nShield Key Attestation Verifier v1.0.2 Application Note

30 January 2024

Table of Contents

1. Introduction

Key attestation refers to a way of cryptographically proving to a third party that a key is gen erated in the nShield HSM and cannot be exported in clear text.

The nShield Key Attestation Verifier allows a user to generate a JSON bundle containing all necessary certificates and information about a key and HSM to verify its protection and use constraints enforced by the HSM. nShield attestation relies on a KLF2 warrant, a certificate chain which links the HSM to its ESN. Verification of the bundle can be done without access to an HSM.

The `nfkmattest` tool can be installed as part of the nShield Security World software or as a standalone package.

2. Installing the nShield Key Attestation Verifier

Always download the nShield Key Attestation Verifier from a trusted source. Verify the integrity after it has been downloaded. You can verify the integrity by using the hash provided at the software download, or obtained from a trusted source.

Before you install the nShield Key Attestation Verifier:

- See the latest *Release Notes* at https://nshieldsupport.entrust.com/hc/en-us/sections/360001115837-Release-Notes for hardware and software compatibility, and known and fixed issues.
- Determine whether the nShield Key Attestation Verifier will be installed as a standalone tool, or installed alongside an existing Security World software installation.
- If you have any instances of the nShield Key Attestation Verifier currently installed, remove them as described in Uninstall the nShield Key Attestation Verifier.

nShield Security World software v13.5 onward includes the `nfkmattest` tool as part of the main installation. Use the steps on this page for stand alone installation or if installing on top of Security World v13.4.

2.1. Install the nShield Key Attestation Verifier

If performing a standalone installation, the following paths should not already exist:

- On Windows: `C:\Program Files\nCipher\nfast`
- On Linux: `/opt/nfast`

2.1.1. Windows:

To install the nShield Key Attestation Verifier on Windows:

1. Download and mount `keyattest-Common-<version>.iso`.
2. In an administrator command prompt, change to where the ISO is mounted.
3. Run `nShieldKeyAttestSetup.bat`, specifying: `-s` (or `--standalone`) for a standalone installation, or `-n` (or `--nshield-upgrade`) to install alongside an existing Security World software installation. For example, to install as a standalone installation:

```
nShieldKeyAttestSetup.bat -s
```

On completion, `nfkmattest` will exist in `C:\Program Files\nCipher\nfast\bin`.

2.1.2. Linux:

To install the nShield Key Attestation Verifier on Linux:

1. Download and mount `keyattest-Common-<version>.iso`.
2. In a command prompt, change to where the ISO is mounted.
3. Run `nShieldKeyAttestSetup.sh`, specifying: `-s` (or `--standalone`) for a standalone installation, or `-n` (or `--nshield-upgrade`) to install alongside an existing Security World software installation. For example, to install alongside an existing Security World software installation:

```
sudo ./nShieldKeyAttestSetup.sh -n
```

On completion, `nfkmattest` will exist in `/opt/nfast/bin`.

2.2. Uninstall the nShield Key Attestation Verifier

Remove the nShield Key Attestation Verifier with the `nShieldKeyAttestSetup` script in the version that you are uninstalling. You cannot uninstall this tool using the script from a different release.

Performing a standalone uninstall will remove the following:

- On Windows: `C:\Program Files\nCipher\nfast`
- On Linux: `/opt/nfast`

Files which need to be retained should be backed up before uninstalling.

2.2.1. Windows

To uninstall the nShield Key Attestation Verifier on Windows:

1. In an administrator command prompt, change to the installation script location:
   - Standalone: `C:\Program Files\nCipher\nfast\python3\nfkmattest.uninstall`
   - Alongside an existing Security World: `%NFAST_HOME%\python3\nfkmattest.uninstall`
2. Run `nShieldKeyAttestSetup.bat` with the `--uninstall` option - specifying: `-s` (or `--standalone`) if installed as a standalone installation, or `-n` (or `--nshield-upgrade`) if

installed alongside an existing Security World software installation. For example, to uninstall a standalone installation:

```
nShieldKeyAttestSetup.bat -s --uninstall
```

2.2.2. Linux:

To uninstall the nShield Key Attestation Verifier on Linux:

1. In a command prompt, change to `opt/nfast/python3/nfkmattest.uninstall`.
2. Run `nShieldKeyAttestSetup.sh` with the `--uninstall` option - specifying: `-s` (or `--standalone`) if installed as a standalone installation, or `-n` (or `--nshield-upgrade`) if installed alongside an existing Security World software installation. For example, to uninstall if installed alongside an existing Security World software installation:

```
sudo ./nShieldKeyAttestSetup.sh -n --uninstall
```

3. Generating an attestation bundle

An attestation bundle can be generated for a key as follows.

```
$ nfkmattest bundle [OPTIONS] APPNAME IDENT
```

The set of certificates and relevant data fields is returned in a JSON-formatted file `key_APPNAME_IDENT.att`. An alternative output file path can be specified with the option `--output PATH`. If the HSM warrant is stored in a non-default directory, its path can be specified with the option `--warrants DIR`. If no HSM warrant is found, see Getting missing warrant.

When generating keys, the `APPNAME` is the section of the key file name as it appears in the `opt/nfast/kmdata/local` (**Linux**) or `C:\ProgramData\nCipher\Key Management Data\local` (**Windows**) filesystem, and the `IDENT` is the last section of the key file name as it appears on the local filesystem. If you generate a `pkcs11`, `custom`, or `embed` key, the `IDENT` is different to that of a `plainname` key generated with `generatekey`. Key file output examples include:

`key_custom_0140c376b9dd2655ae75c99d940e3477408aef14` `key_embed_5fe6c9e346b4d-d2ea35e1de9049861fe97888b5c` `key_pkcs11_ua5fe6c9e346b4dd2ea35e1de9049861fe97888b5c`

Bundle generation is supported for asymmetric (public/private) key pairs only. Symmetric keys are not supported for bundle generation.

3.1. Bundle details

The possible bundle fields are outlined below.

Field	Presence	Description
pubkeydata	Always	Public key material in nCore format (including any domain parameters)
kcmsg	Always	The key generation certificate body
kcsig	Always	The signature on the key generation certificate under KML
modstatemsg	Always	A module state certificate
modstatesig	Always	The signature on the module state certificate under KLF2.
warrant	Always	The D3S encoding of the generating HSM's warrant.

Field	Presence	Description
root	Always	The name of the warranting root used in this certificate. This will always be KWARN-1 for nShield HSMs.
knsopub	Persistent keys	KNSO public key
hkre	Recoverable keys	Hash of KRE
hkra	Recoverable keys	Hash of KRA
hkfips	Persistent keys in FIPS worlds	Hash of KFIPS
hkmc	Persistent keys	Hash of KMC
hkm	Persistent keys	Hash of KM
CertKMaKMCbKNSO	Persistent keys in non-FIPS worlds	Signature on world binding cert
CertKMaKMCaKFIPSbKNSO	Persistent keys in FIPS worlds	Signature on world binding cert
CertKREaKRAbKNSO	Recoverable keys	Signature on world binding cert
ciphersuite	Persistent keys	Ciphersuite name for security world from the NFKM_CipherSuite enumeration (e.g. DLf3072s256mAESc-SP800131Ar1)

4. Verifying an attestation bundle

An attestation bundle can be verified as follows.

```
$ nfkmattest verify PATH
```

This will output information about the key in JSON format. The fields are:

Key	Syntax	Meaning
`path`	string	The path of the bundle file
`protection`	string	Type of protection, either `module`, `softcard` or `cardset`
`recovery`	boolean	Whether key recovery is enabled for the key, if available
`permissions`	list	Key usage permissions
`esn`	string	Electronic Serial Number (ESN) of the HSM used to generate the key
`hknso`	string	The hash of the nShield Security Officer key (KNSO) for the Security World used to generate the key
`k`	object	Public key parameters (a more detailed breakdown can be seen in the section below)

4.1. Private key operations

The `permissions` field is a list of the permitted private key operations. The following are the possible options.

Permission	Description
`decrypt`	Key can decrypt messages, yielding plaintext
`unwrap`	Key can decrypt messages, yielding a key inside the HSM (this includes loading of key blobs)
`sign`	Key can sign messages

4.2. Public key parameters

The `k` field is the public key is an nCore `M_KeyData` structure in JSON format. The key object has two fields.

Key	Syntax	Meaning
`k.type`	string	The public key type, described in sections below
`k.data`	object	Public key material

In almost all cases, integers are represented as the RFC4648 section 4 base64 encoding of the big-endian form of the integer value. The representation is normally minimal, meaning that a value of 0 is represented by the empty string.

The fields in `k.data` will depend on the key type.

4.2.1. RSA public keys

The key type is `RSAPublic`. The data object has two fields.

Key	Syntax	Meaning
`k.data.n`	base64(integer)	Public modulus
`k.data.e`	base64(integer)	Public exponent

4.2.2. DSA and KCDSA public keys

The key types are `DSAPublic` and `KCDSAPublic`. The data object has the following fields.

Key	Syntax	Meaning
`k.data.dlg`	object	Container for domain parameters
`k.data.dlg.p`	base64(integer)	Field modulus
`k.data.dlg.q`	base64(integer)	Subgroup order
`k.data.dlg.g`	base64(integer)	Subgroup generator
`k.data.y`	base64(integer)	Public key

4.2.3. ECC public keys

The key types are `ECDSAPublic` (signature only), `ECDHPublic` (key agreement only) and `ECPublic`. The data object has the following fields.

Key	Syntax	Meaning
`k.data.curve`	object	Container for domain parameters
`k.data.curve.name`	string	Domain parameters of curve (see below for supported values)
`k.data.Q`	object	Public point
`k.data.Q.flags`	list	Always empty
`k.data.Q.x`	base64(integer)	X coordinate of public point
`k.data.Q.y`	base64(integer)	Y coordinate of public point

4.2.3.1. Supported values for `k.data.curve.name`

The supported named curves are as follows.

`NISTP192`	`NISTP224`	`NISTP256`
`NISTP384`	`NISTP521`	`NISTB163`
`NISTB233`	`NISTB283`	`NISTB409`
`NISTB571`	`NISTK163`	`NISTK233`
`NISTK283`	`NISTK409`	`NISTK571`
`BrainpoolP160r1`	`BrainpoolP160t1`	`BrainpoolP192r1`
`BrainpoolP192t1`	`BrainpoolP224r1`	`BrainpoolP224t1`
`BrainpoolP256r1`	`BrainpoolP256t1`	`BrainpoolP320r1`
`BrainpoolP320t1`	`BrainpoolP384r1`	`BrainpoolP384t1`
`BrainpoolP512r1`	`BrainpoolP512t1`	`ANSIB163v1`
`ANSIB191v1`	`SECP160r1`	`SECP256k1`

If this field is `Custom` or `CustomLCF`, the full domain parameters of the curve are given in further fields. These are described in the nCore API documentation.

4.2.4. Ed25519 public keys

The key type is `Ed25519Public`. The data object has the following field.

Key	Syntax	Meaning
`k.data.k`	base64(bytes)	RFC8032-format public key

5. Getting a missing warrant

To use the `nfkmattest` tool to generate an attestation bundle, the HSM used must have a KLF2 warrant installed in the appropriate location, or an alternative search directory specified with the `--warrants DIR` option.

If a warrant can't be found locally but has been installed on a different server, it can be copied over a secure connection. By default, these warrants are stored in `NFAST_KMLOCAL/warrants/` for Solo + or Solo XC, or `NFAST_KMDATA/hsm-<ESN>/warrants/` for Connect + or Connect XC modules. nShield 5s and nShield 5c are supplied with the required warrants pre-installed and stored within the module. These will be fetched by the Security World software when necessary.

If no warrants are installed, complete the steps in the relevant nShield User Guide to request one from Entrust.

6. Worked examples

Below is an example generating a key, creating a bundle and verifying the bundle for a recov erable RSA key.

```
$ generatekey -b simple protect=token type=RSA ident=rsaexample
key generation parameters:
 operation    Operation to perform              generate
 application  Application                        simple
 protect      Protected by                      token
 slot         Slot to read cards from           0
 recovery     Key recovery                      yes
 verify       Verify security of key            yes
 type         Key type                          RSA
 size         Key size                          2048
 pubexp       Public exponent for RSA key (hex)
 ident        Key identifier                    rsaexample
 plainname    Key name
 nvram        Blob in NVRAM (needs ACS)         no

Loading `sampleocs':
 Module 1: 0 cards of 1 read
 Module 1 slot 0: `sampleocs' #1
 Module 1 slot 0:- passphrase supplied - reading card
Card reading complete.

Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_simple_rsaexample
$ nfkmattest bundle simple rsaexample
$ nfkmattest verify key_simple_rsaexample.att
{
  "path": "key_simple_rsaexample.att",
  "protection": "cardset",
  "recovery": true,
  "type": "RSAPublic",
  "permissions": [
    "sign",
    "decrypt",
    "unwrap"
  ],
  "esn": "A89B-485C-A955",
  "hknso": "06669505 feaa2de2 5e94940b d2ac1341 a6e2b475",
  "k": {
    "type": "RSAPublic",
    "data": {
      "e": "AQAB",
      "n":
"514JPs/SdZ7viCuXidF/IkI/13PLsu3GfKp8YgmQ5P5qK/mWRcMPeQ0ZO8SQK9BsoKf+/Shhxn081TxP3n8U4o7D94BxRfcpht02nk3mmQvDm0aN
dzV9cBBec7Jk0ipegAgjQm+KfF8dbWtCbmvki7Eg2jcscCaT5qo9n0XhwXLYhmVG8CdqGrPYQR3CVstzjv+uTc+vofmii29S6D4uYG/z9kWDyym3X
UKmvjwGAEt2kyZ7BVxeP+tahIkLnvglFJuYKIEF3I86+2UKem8hJa1tTxkXsWuGA0ShsXikV67uJmXMG0Ablx9HmmFdUQ5FL/Gs9ETEA1ZMjX2WEi
dx3w=="
    }
  }
}
```

Below is an example of the same process for a non-recoverable PKCS#11 ECDSA key. The key can be generated using the nShield PKCS#11 API but `generatekey` is used here for brevity.

```
$ generatekey -b pkcs11 protect=token type=ECDSA plainname=ecdsaexample
```

```
key generation parameters:
 operation    Operation to perform       generate
 application  Application                pkcs11
 protect      Protected by               token
 slot         Slot to read cards from    0
 recovery     Key recovery               no
 verify       Verify security of key     yes
 type         Key type                   ECDSA
 plainname    Key name                   ecdsaexample
 nvram        Blob in NVRAM (needs ACS)  no
 curve        Elliptic curve             NISTP256

Loading `sampleocs':
 Module 1: 0 cards of 1 read
 Module 1 slot 0: `sampleocs' #1
 Module 1 slot 0:- passphrase supplied - reading card
Card reading complete.

Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_uc3f8abff09207a68ead2a0176ba7aee425370eab1-
04b5c0582d4371e4ac7e370723398e469441427c
$ nfkmattest bundle pkcs11 uc3f8abff09207a68ead2a0176ba7aee425370eab1-04b5c0582d4371e4ac7e370723398e469441427c -o
key_pkcs11_ecdsaexample.att
$ nfkmattest verify key_pkcs11_ecdsaexample.att
{
  "path": "key_pkcs11_ecdsaexample.att",
  "protection": "cardset",
  "recovery": false,
  "type": "ECDSAPublic",
  "permissions": [
    "sign"
  ],
  "esn": "A89B-485C-A955",
  "hknso": "06669505 feaa2de2 5e94940b d2ac1341 a6e2b475",
  "k": {
    "type": "ECDSAPublic",
    "data": {
      "curve": {
        "name": "NISTP256"
      },
      "Q": {
        "flags": [],
        "x": "EhCTAIWyYL38wdhHM8x60fKIp6rQ3wWp6hj9SWWiW+k=",
        "y": "r1YAfJjH50goy2Ja7u80y1UZwiv7LT84rRH+7p/2EVg="
      }
    }
  }
}
```