Application Notes

nShield Warrants

14 January 2025

Table of Contents

A warrant, in nShield terminology, is a certificate chain that attests to the identify of a device.

Unless otherwise specified, the following notation will be used in this appnote for DDDS-supported value types:

- `[…]` - list
- `{…}` - map
- `'…'` - symbol
- `"…"` - string
- `ByteBlock(…)` - byte-block
- `int(…)` - integer

For further information on the DDDS format, see Dynamically Defined Data Structures.

1. Warrant format

1.1. Overall format

A warrant is a marshalled DDDS list containing, in order:

- A root key name which defines which key must be used to verify the first certificate
- Zero or more Warrant Delegation Certificates
- One of either a Smartcard Information Certificate or Module Information Certificate

The root key name is a DDDS symbol. Each individual certificate is a DDDS map.

1.2. Warrant certificates

All warrant certificates have the following structure.

```
{
  'Signature': ByteBlock(...),
  'Payload': ByteBlock(...)
}
```

The value of `'Signature'` is a signature on the payload. The cryptographic key and mechanism used to verify this signature is not encoded in the certificate itself; it must be known out-of-band. For nShield warrants, only ECDSA signatures on NISTP521 are used, represented by the concatenation of the signature components in bigendian format, i.e. `I2OSP(r, 66) || I2OSP(s, 66)`. `I2OSP(n, l)` refers to the representation of integer n as a big-endian l-length octet string.

The value of `'Payload'` is itself a marshalled DDDS map. All nShield warrant certificates are identified by the presence of the `'WarrantCertificateType'` member in the payload with the value of either `'Delegation'`, `'SmartcardInformation'`, `'ModuleInformation'` or `'FieldUpgradeModuleInformation'`. Note that any `'FieldUpgradeModuleInformation'` certificates depend on signatures made using legacy DSA-1024 keys, limiting their security. The remaining certificate types are defined in further sections.

1.2.1. Warrant Delegation Certificates

A Warrant Delegation Certificate is a certificate whose payload is:

```
{
  'WarrantCertificateType': 'Delegation',
  'DelegateKey': <keydata>,
```

```
  'SigMech': <mech>
}
```

'DelegateKey' and 'SigMech' define the key and mechanism used for verification of the next certificate in the warrant.

For nShield warrants, only NISTP521 ECC public keys are used as the 'DelegateKey', which have the following format:

```
['ECDSA', 'Public', 'NISTP521', [ByteBlock(x), ByteBlock(y)]]
```

For nShield warrants, only ECDSA with SHA512 is used for the 'SigMech', which has the following format:

```
['ECDSA', ['EMSA1', 'SHA512']]
```

1.2.2. Smartcard Information Certificates

A Smartcard Information Certificate is a certificate whose payload is:

```
{
  'WarrantCertificateType': 'SmartcardInformation',
  'CardPub': <keydata>,
  'CardMech': <mech>,
  'SerialNumber': ByteBlock(...)
}
```

'CardPub' is the public half of a key held by the smartcard.

'SerialNumber' is the 9-byte serial number of the smartcard.

'CardMech' is the key agreement mechanism the card will use. As in the Warrant Delegation Certificates, the only mechanism used in nShield warrants is ECDSA with SHA512.

1.2.3. Module Information Certificates

A Module Information Certificate is a certificate whose payload is:

```
{
  'WarrantCertificateType': 'ModuleInformation',
  'KLF<KLF number>pub': <keydata>,
  'KLF<KLF number>mech': <mech>,
  'Approvals': [<approval>, ...],
  'ElectronicSerialNumber': "ABCD-ABCD-ABCD",
  'PhysicalSerialNumber': "01-234567"
}
```

'KLF<KLF number>pub' is the public half of the module's long-term signing key. Existing warrant formats always use NISTP521 ECC public keys.

'KLF<KLF number>mech' is the signature mechanism to be used when verifying messages signed by a module's KLF<KLF number>. Existing warrant formats always use ECDSA with SHA512.

'ElectronicSerialNumber' and 'PhysicalSerialNumber' are the electronic and physical serial numbers of the module to which the warrant refers.

'Approvals' is a possibly empty list of approvals which declare the security properties of the hardware platform associated with the given module.

<KLF number> can be either 2 or 3, for example:

```
{
  'WarrantCertificateType': 'ModuleInformation' ,
  'KLF2pub': <keydata>,
  'KLF2mech': <mech>,
  'Approvals': [<approval>, ...],
  'ElectronicSerialNumber': "ABCD-ABCD-ABCD",
  'PhysicalSerialNumber': "01-234567"
}
```

1.2.3.1. Hardware approvals

A single module approval is a DDDS list whose first item is a symbol defining the remaining items of the list. Only one symbol is defined so far: 'FIPS140'. Such an approval has the following form:

```
['FIPS140', int(version), int(level), '<kind>']
```

version indicates the version of the targetted FIPS140 standard.

level indicates the FIPS140 security level the hardware platform can achieve.

kind indicates the FIPS140 hardware profile which best describes the platform. For current hardware, this is either 'MultiChipEmbedded', or 'MultiChipStandalone'.

Therefore ['FIPS140' , int(2), int(3), 'MultiChipEmbedded'] declares that, at the time the warrant was created, the hardware was approved to FIPS140-2 level 3.

Other kinds of approval are to be defined.

1.3. Verifying a warrant

A real nShield HSM warrant might look like the DDDS-marshalled form of the following:

```
[
  'KWARN-1',                                              # root key name
  {'Signature': ByteBlock(...), 'Payload': ByteBlock(...)}, # warrant delegation certificate
  {'Signature': ByteBlock(...), 'Payload': ByteBlock(...)}  # module information certificate
]
```

Verifying any nShield warrant means working through the certificates in order. The first must be verified under the root key specified, the second is verified under the key from the payload of the first, and so on. After the last has been verified, the device public key and identity can be extracted from it, and the relationship between them can be trusted (to the extent that you trust the root key). The root key used for nShield module warrants is defined in nShield root key.

2. nShield root key

nShield module warrants are rooted at key KWARN-1:

```
KeyData.type = ECDSAPublic
       .data.curve.name = NISTP521
            .Q.flags = 0x0
              .x =
0x1d21dfde6d7e001c5a4f78ae8d2f799e0caf79c60d673d0da88b206a3ba52f20dce0956ce02f01af32736767c8b9feff398c29e02085273
71856aa2f40fcae61d96
              .y =
0x1641ca5472f06257de815ae06b33e1b868c149645f55fb7c91738014d3d235e7b247649cc2f7d1075e9b4d4388661e754a7fc386913c33d
dd60208bded5301a1b77
```

Its nCore key hash is `679eae63 cd8b4e63 a57afbf4 ac21417c e3578955`.

3. Python APIs

The following Python libraries are available for processing nShield warrants:

- `nfddds` implements encoding and decoding of DDDS structures
- `nfwar` provides warrant support in Python, including verification with or without using an HSM
- `nchex` allows for parsing warrant files in hex format

For an example using `nfwar` to parse a warrant without access to an HSM, see `$NFAST_HOME/python3/examples/offline_warrant_verification.py`.

4. Warrant example

This secton outlines the process of reading a nShield KLF2 module warrant file and extracting information from it. Warrants are typically encoded in hex format, which can be processed using the nchex library. The outputs have been reformatted slightly for readability.

We can unpack the warrant with the nfddds module:

```
> import nfddds
> import nchex
> file_data = open("/opt/nfast/kmdata/warrants/ABCD-ABCD-ABCD", "rb").read()
> (klftype, warrant_data) = nchex.parse(str(file_data, "UTF-8"))
> warrant = nfddds.decode(warrant_data)
> print(klftype)
KLF2Warrant
> print(warrant)
[
  'KWARN-1',
  {
    'Payload':
      """b33b4465 6c656761 74654b65 79943545 43445341 36507562 6c696338 4e495354
      50353231 92f4c542 01ce51e1 6a7187d2 9a61842f 271be758 36d00405 e24ef3e9
      88a2e49d 8ad6e653 1ddd7cb9 bb0c62c9 1536dc16 f183c10e 4ec53de9 66404026
      6b86b472 42433f09 4829f4c5 410c97f2 5beebdee dc81aa2e 147159df 495ae0aa
      3ba20d22 2fcbb271 b4e62d68 bd0b18ee 6007a610 cf356f2f 7e9d94c7 337c818a
      ddc2ccbf 1934898e 699b3ff5 cc733753 69674d65 63689235 45434453 41923545
      4d534131 36534841 353132c4 16576172 72616e74 43657274 69666963 61746554
      7970653a 44656c65 67617469 6f6e""",
    'Signature':
      """00d01425 60724466 7afccb06 e168c6d3 77be0623 d7253fd4 0de6bffc 1a394eaf
      a1bda0af e850eca6 c0587eb1 f6c271dd 2602f6ec 16e2963c 281118c5 67817c01
      aaa901ad 6f5c9427 7342bd4b 59ade2b9 d7f6d86d 6e46a445 7d201d56 3537390c
      b0979cfc ef5baa20 95f6c4bc af35c1b9 17a051de c5889620 6cf5dec4 44d8407e
      ad7b89f8"""
  },
  {
    'Payload':
      """b6394170 70726f76 616c7391 94374649 50533134 300203c4 114d756c 74694368
      6970456d 62656464 6564c416 456c6563 74726f6e 69635365 7269616c 4e756d62
      65722e41 4243442d 41424344 2d414243 44384b4c 46326d65 63689235 45434453
      41923545 4d534131 36534841 35313237 4b4c4632 70756294 35454344 53413650
      75626c69 63384e49 53545035 323192f4 c5410f45 cd4f61d5 318de1ff 4bc56405
      d6d36a3a c3ab966e ddc7f228 ff71a23e 929a10e3 513c4d2b bcc51e5f a1ef5129
      57c66cfb a2731e44 7894e470 a315bbe2 8acb59f4 c5412838 e5947737 0789ea09
      d1b83542 a2d4aaf8 3db54847 275cba0f ebb773d8 c45a60be 6fd6efee 0656cafe
      e0b580e0 96e5baf1 bdfb81e5 8df760ee dd2f8865 36727cc4 14506879 73696361
      6c536572 69616c4e 756d6265 72293031 2d323334 353637c4 16576172 72616e74
      43657274 69666963 61746554 797065c4 114d6f64 756c6549 6e666f72 6d617469
      6f6e""",
    'Signature':
      """018f257b 8adfa961 2223d4e3 abd0921e 00b79749 cd49a320 bf5990f5 a4f325ea
      2913c2ef f19d0544 f36c9310 4dade3b3 10fe0779 819a3ea6 6b2be718 46acaceb
      b90f0106 13c23118 bb4f23db 7d12b6b4 8fc3c7ec 4c452cf5 022bef2e c0bc24d1
      6dd5d57c abf9d77c 6a769b4b 0cc6ee78 9cfe39c6 7280eca9 baf4317c fee59ff3
      a951beb0"""
  }
]
```

The basic structure is as described in the Warrant format section: a list consisting of a root

key name, one delegation certificate and a module information certificate.

We can unpack the payload of the delegation certificate:

```
> delegation_cert = warrant[1][nfddds.Symbol('Payload')]
> print(nfddds.decode(delegation_cert))
{
  'DelegateKey': [
    'ECDSA', 'Public', 'NISTP521',
    [

619869569498291931339665867925139435365659916621719747957505084594535642957795503109526140600452686417106163642774260002418856254768643229567689380381073616 9,

168851790370479162603327826894762014276575457671531073025230026616996041091987680751434932025755737810417296484254014405915751348478117063721036878705970291
    ]
  ],
  'SigMech': ['ECDSA', ['EMSA1', 'SHA512']],
  'WarrantCertificateType': 'Delegation'
}
```

Similarly for the module information certificate:

```
> module_cert = (warrant[2][nfddds.Symbol('Payload')])
> print(nfddds.decode(module_cert))
{
  'ElectronicSerialNumber': 'ABCD-ABCD-ABCD',
  'PhysicalSerialNumber': '01-234567',
  'Approvals': [['FIPS140', 2, 3, 'MultiChipEmbedded']],
  'WarrantCertificateType': 'ModuleInformation',
  'KLF2pub': [
    'ECDSA', 'Public', 'NISTP521',
    [

204772945937072010226860998619571494264264799010175099002158797896809774651276304246398533131554228909781887253956881741534001231085622044477444946293082969,

539292244234422539754545398145607083325398812601271805090001462425588071401713689513940584061317769068587981621988409982003420521563484566472379191365366396
    ]
  ],
  'KLF2mech': ['ECDSA', ['EMSA1', 'SHA512']]
}
```