

Migrating to EJBCA

A Study Guide

Contents

Migrating to EJBCA	3
Introduction	3
<i>Keon CA</i>	3
<i>EJBCA</i>	3
Detailed HOW-TO	4
<i>Migration of KCAs' signing keys</i>	4
<i>Importing CA</i>	8
<i>Importing user certificates</i>	9
About PrimeKey Solutions AB	9

Migrating to EJBCA

Introduction

This document is written to help illustrate how to migrate from another CA to EJBCA. In order to demonstrate specific steps and to help reader reproduce those steps, one specific CA was chosen - RSA Keon CA. The general idea how to migrate is the same for other CA implementations.

Migration from another CA to EJBCA consists of the following steps:

- Migration of the CAs' signing keys on nCipher HSM, so that the keys can be used by EJBCA
- Import of the CA within EJBCA
- Import of the user certificates in EJBCA

This document outlines how to migrate a simple installation of KCA to EJBCA. We recommend to do first a test migration as to be familiar with the process.

Keon CA

An setup of KCA on a target environment:

- Windows Server 2003
- KCA 6.5
- One root CA – TestKCARootCA
- One sub CA – TestKCASubCA
- Signing keys for the CAs on nCipher nShield PCI card
- 5 users issued by TestKCASubCA

After the installation of this environment, we make a backup of nCipher security world, certificates and user certificates.

EJBCA

A target environment for EJBCA is chosen:

- Ubuntu Linux 7.04 AMD64
- JBoss 4.2.0, MySQL 5.0
- EJBCA 3.5 or later; 3.9 recommended
- One root CA – TestKCARootCA
- One sub CA – TestKCASubCA
- Signing keys for the CAs on nCipher nShield PCI kort
- 5 users issued by TestKCASubCA

Detailed HOW-TO

Migration of KCAs' signing keys

PKCS#11 Configure PKCS#11 for nCipher.

For nCipher, the slot number is some virtual slot, where we do not know the name of it. Fortunately, one can use the default slot by simply omitting to give the slot number. In order to use nCipher PKCS#11, one has to set up some parameters.

Create file /opt/nfast/cknfastrc with this content:

```
CKNFAST_LOADSHARING=1
CKNFAST_NO_ACCELERATOR_SLOTS=1
CKNFAST_NO_UNWRAP=1
CKNFAST_OVERRIDE_SECURITY_ASSURANCES=import
```

Signature Keys

When KCA is installed, it generates keys on nCipher HSM. Those keys are in “native” format and can only be used by KCA. There is no public key associated on the HSM.

We start by listing some general information about the HSM:

```
C:\nfast\bin>enquiry
Server:
  enquiry reply flags  none
  enquiry reply level  Six
  serial number       B1BB-0EE9-3511
  mode                 operational
  version              2.23.6
  speed index         440
  rec. queue           422..622
  level one flags      Hardware HasTokens
  version string       2.23.6cam6, 2.22.43cam8 built on Oct 13 2006
16:16:12
  checked in           00000000431dca98 Tue Sep 06 18:58:00 2005
  level two flags      none
  max. write size      8192
  level three flags    KeyStorage
  level four flags     OrderlyClearUnit HasRTC HasNVRAM HasNS0PermsCmd
ServerHasPollCmds FastPollSlotList HasSEE HasKLF HasShareACL
HasFeatureEnable Has sFileOp HasPCIPush HasKernelInterface HasLongJobs
ServerHasLongJobs AESModuleKeys NTokenCmds LongJobsPreferred
  module type code     0
  product name         nFast server
  device name
  EnquirySix version   4
  impath kx groups
  feature ctrl flags   none
  features enabled     none
  version serial       0

Module #1:
  enquiry reply flags  none
  enquiry reply level  Six
  serial number       B1BB-0EE9-3511
```

```

mode                operational
version             2.22.43
speed index        440
rec. queue         19..152
level one flags    Hardware HasTokens
version string     2.22.43cam8 built on Oct 13 2006 16:16:12
checked in        00000000452f6a4d Fri Oct 13 12:28:29 2006
level two flags    none
max. write size    8192
level three flags  KeyStorage
level four flags   OrderlyClearUnit HasRTC HasNVRAM HasNSOPermsCmd
ServerHasPollCmds FastPollSlotList HasSEE HasKLF HasShareACL
HasFeatureEnable  Has sFileOp HasPCIPush HasKernelInterface HasLongJobs
ServerHasLongJobs AESModuleKeys NTokenCmds LongJobsPreferred
module type code   7
product name       nC1003P/nC3023P
device name        #1 PCI bus 7 slot 1
EnquirySix version 5
impath kx groups   DHPrime1024
feature ctrl flags LongTerm
features enabled   StandardKM
version serial     24
rec. LongJobs queue 18
SEE machine type   PowerPCSYF

```

C:\nfast\bin>nfkminfo.exe

```

World
generation 2
state 0x17270000 Initialised Usable Recovery !PINRecovery
!ExistingClient RTC NVRAM FTO SEEDebug
n_modules 1
hkns0 057731d6c635560900003fed89eba88c5082f2a1
hkm b08e66b01777fcf34dceb77c0684c8d1a071144e (type DES3)
hkmwk 1d572201be533ebc89f30fdd8f3fac6ca3395bf0
hkrc b0f5dbebdc703c6acd7685b261f6748bc4a9535
hkra 7368a945efc76505a19092c5115f493716de3172
hkmc 1d1e3fb769837be06e028bc038d1789ced2ac9e1
hkrtc 55f17755cae9ae49a588f154260306deae1f5cc
hkncv 82674f0623407fbaac5a3aaa0fc08346dff34f37
hkdssee 6df83c268c25939c307f61245235a2ac44e487ae
hkfto cae32a48bd1b9e68e606ae723f8dcb93eb4ee9ab
hknull 1d572201be533ebc89f30fdd8f3fac6ca3395bf0
ex.client none
k-out-of-n 1/1
other quora m=1 r=1 nv=1 rtc=1 dssee=1 fto=1
createtime 2007-07-16 14:58:41
nso timeout 10 min

```

```

Module #1
generation 2
state 0x2 Usable
flags 0x0 !ShareTarget
n_slots 2
esn B1BB-0EE9-3511
hkml 5e43facc6aa39068092762d80a1954bde193b4b

```

```

Module #1 Slot #0 IC 25
generation 1

```

```
phystype      SmartCard
slotlistflags 0x2 SupportsAuthentication
state         0x5 Operator
flags         0x10000 Passphrase
shareno       1
shares        LTU(PIN)
error         OK
Cardset
name          "oper"
k-out-of-n    1/1
flags         NotPersistent PINRecoveryForbidden(disabled) !RemoteEnabled
timeout       none
card names    ""
hkltu         3126f2b3cf7d9d53e3ba278a081ef471644298f8
gentime       2007-07-16 15:00:07
```

```
Module #1 Slot #1 IC 0
generation    1
phystype      SoftToken
slotlistflags 0x0
state         0x2 Empty
flags         0x0
shareno       0
shares        OK
error         OK
```

No Cardset

No Pre-Loaded Objects

One can see which keys are being used by command:

```
C:\nfast\bin>nfkminfo -k
```

Key list - 19 keys

```
AppName rsa-keon-ca-65      Ident 1184599865281000
AppName rsa-keon-ca-65      Ident 1184599867765000
AppName rsa-keon-ca-65      Ident 1184599888484000
AppName rsa-keon-ca-65      Ident 11845998900
AppName rsa-keon-ca-65      Ident 1184599947937000
AppName rsa-keon-ca-65      Ident 1184599969937000
AppName rsa-keon-ca-65      Ident 1184599971843000
AppName rsa-keon-ca-65      Ident 1184599974609000
AppName rsa-keon-ca-65      Ident 1184599977718000
AppName rsa-keon-ca-65      Ident 1184599980515000
AppName rsa-keon-ca-65      Ident 1184599982765000
AppName rsa-keon-ca-65      Ident 1184599985656000
AppName rsa-keon-ca-65      Ident 1184599987625000
AppName rsa-keon-ca-65      Ident 1184599989140000
AppName rsa-keon-ca-65      Ident 1184599991234000
AppName rsa-keon-ca-65      Ident 1184600151640000
AppName rsa-keon-ca-65      Ident 1184600153609000
AppName rsa-keon-ca-65      Ident 1184659106609000
AppName rsa-keon-ca-65      Ident 1184659187875000
```

There are quite some keys, and we can't immediately say which one is the CA's signing key. This is deduced in the following way:

```
C:\nfast\bin>pubkey-find.exe c:\kca.pem
input format cert
nCore hash d0196ea2e070315e9e162c28dc5a524bf6380d

unnamed
appname rsa-keon-ca-65
ident 1184659106609000
```

After which we can add the corresponding public key:

```
C:\nfast\bin>pubkey-find.exe --augment c:\kca.pem
```

And now we "move" the key as to be usable by PKCS#11:

```
C:\nfast\bin>generatekey --retarget -no-verify pkcs11
13:40:30 WARNING: nfgk_debug_output is now deprecated (see manual).
from-application: Source application? (jcecp, pkcs11, rsa-keon-ca-65)
[default jcecp] > rsa-keon-ca-65
from-ident: Source key identifier? (1184599865281000, 1184599867765000,
1184599888484000, 11845998900,
1184599947937000, 1184599969937000,
1184599971843000, 1184599974609000,
1184599977718000, 1184599980515000,
1184599982765000, 1184599985656000,
1184599987625000, 1184599989140000,
1184599991234000, 1184600151640000,
1184600153609000, 1184659106609000,
1184659187875000) [1184599865281000]
```

```
> 1184659106609000
plainname: Key name? [] >
ERROR: plainname: key name unspecified
plainname: Key name? [] > kcaSign
key generation parameters:
operation      Operation to perform      retarget
application    Application                  pkcs11
slot           Slot to read cards from    0
verify         Verify security of key     no
from-application Source application         rsa-keon-ca-65
from-ident     Source key identifier       1184659106609000
plainname      Key name                    kcaSign
*****
* WARNING: will not verify the security of the key *
*****
```

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

```
Key successfully retargetted.
Path to key: c:\nfast\kmdata\local\key_pkcs11_uc3126f2b3cf7d9d53e3ba278a081
ef471644298f8-628484d430c6c0502fdb1a520fb84b9dc73c8372
```

In order to be able to use the public key via PKCS#11, we have to associate a certificate to the key. For this, we take the actual certificate:

```
C:\nfast\bin>ckcerttool.exe -c oper -f c:\ca\kca.pem -k uc3126f2b3cf7d9d53e3ba278a081ef471644298f8-628484d430c6c0502fdb1a520fb84b9dc73c8372 -L kcaSign
```

```
Certificate found, processing...
```

```
Please enter the passphrase for "oper" token (No echo set).
```

```
Passphrase:
```

```
Certificate successfully imported.
```

```
Run cklst to view your certificate object.
```

```
OK
```

At this stage, we proceed to generate two more keys to be used by EJBCA. One of the keys will be used by EJBCA for internal encryption within EJBCA, and the other for testing of the HSM. This generation should be done as described in the EJBCA User Guide, using the regular EJBCA tools. Generate one RSA key of length 2048 bits with alias `kcaDefault` and one RSA key of length 1024 with alias `kcaTest`.

```
keytool -generate -keystore NONE -storetype PKCS11-NFastJava -storepass foo123 -alias kcaDefault -keyalg RSA -keysize 2048
keytool -generate -keystore NONE -storetype PKCS11-NFastJava -storepass foo123 -alias kcaTest -keyalg RSA -keysize 1024
```

We use the EJBCA tools to get automatic association between the private key and a (dummy) certificate, so it can be used through the Java PKCS#11 provider.

Importing CA

After doing *retarget* on the keys for the Root CA and the Sub CA, we can import CA certificates into EJBCA.

KCA does not use the same DN order as EJBCA does by default, so if you want issued certificate to look exactly the same you should uncheck the checkbox Use LDAP DN order in EJBCA after the installation.

Install EJBCA as usual with one AdminCA, after which we can import the CAs from KCA from the help script.

```
bin/ejbca.sh ca importca TestKCARootCA org.ejbca.core.model.ca.catoken.
PKCS11CAToken foo123 rootca.properties TestKCARootCA.pem
```

```
rootca.properties contains:
```

```
defaultKey rootDefault
```

```
certSignKey rootSign
```

```
crlSignKey rootSign
```

```
testKey rootTest
```

```
pin foo123
```

```
sharedLibrary /opt/nfast/toolkits/pkcs11/libcknfast.so
```

We do the same for the SubCA:

```
bin/ejbca.sh ca importca TestKCASubCA org.ejbca.core.model.ca.catoken.
PKCS11CAToken foo123 subca.properties TestKCASubCA-chain.pem
```

```
subca.properties contains:  
defaultKey subDefault  
certSignKey subSign  
crlSignKey subSign  
testKey subTest  
pin foo123  
sharedLibrary /opt/nfast/toolkits/pkcs11/libcknfast.so
```

```
TestKCASubCA-chain.pem is created with:  
cat TestKCASubCA.pem TestKCARootCA.pem > TestKCASubCA-chain.pem
```

After this, EJBCA has one Root CA and one Sub CA that use the same signing keys as KCA.

Importing user certificates

In EJBCA, there is a command to import user certificates:

```
bin/ejbca.sh ca importcert kca1 foo123 TestKCASubCA ACTIVE user1.pem
```

The command does not import information about revocation. Full revocation information is possible to implement in a programmatic way.

You can easily write your own migration tool, speeding up the process by using the command line code as a template. It can be found in `src/java/org/ejbca/ui/cli/CAImportCertCommand.java`.

About PrimeKey Solutions AB

The company's business idea is to provide solutions and services in IT-security with focus on two main areas; Public Key Infrastructure (PKI) and smart card appliances.

Our vision is to provide state-of-the-art solutions that integrate with customer application environments. We help our customers to solve concrete business needs in a cost-effective way, with strong emphasis on security issues. Our goal is to maintain and improve our position as leaders in providing security solutions and services by employing the resources at our disposal combined with our unique expertise.

