Report on Certificate Tools

Oliver Bodnár

August 2016

Contents

Part I

Overview

Chapter 1

Overview Tables

1.1 General

Table 1.1: Overwiew General

Tools	General			
	Type	View + info	License	Operating System
OpenSSL	PKCS12	Yes	Public	Both
Keytool	JKS	Yes	Public	Both
GnuPG				
Keystore Explorer	Both	Yes	Public	Win — OS X
XCA	PKCS12	Yes	Public	Both
GnuTLS	PKCS12	Yes	Public	LINUX

1.1.1 Type

This section defines type of storage file in which the certificates or keys are saved. Most common are PKCS#12 (.pfx and .p12 extentions) and JKS (Java KeyStore).

1.1.2 View and information

This section shows whether it is possible to view certificates or keys and additional information. Yes means that at least viewing is supported while not necessary meaning possibility of viewing more information about certificate.

1.1.3 License

Type of license and possibility of using said tool for testing or production code. Public means that license is not requiered for use in production code but does not mean that it should be used as such. Definition if it is advised to use said tool in production code or only in testing enviroment will be talked about in the next chapter under each tool.

1.1.4 Operating Systems

In this part the Operating System where the tool is imported is defined. Both means that it is implemented under UNIX like systems and Windows systems. Otherwise the Operating System is defined (Windows — UNIX).

1.2 Generation and signing of certificates

1.2.1 Keys, certificates and basic constraints

Table 1.2: Generation of keys and certificates and basic constrints

Tool	Generate keys				Basic Constraints	
	+ self-signed certificate	+ CSR	Specify length	Specify algorithm	Specify Type	Specify path length
OpenSSL	Yes	Yes	Yes	Yes	Yes	Yes
Keytool	Yes	Yes	Yes	Yes	Yes	Yes
GnuPG						
Keystore explorer	Yes	Yes	Yes	Yes	Yes	Yes
XCA	Yes	Yes	Yes	Yes	Yes	Yes
GnuTLS	Yes	Yes	Yes	Yes	Yes	Yes

Generate keys

Self-signed certificate possibility of using 1 command to generate key pair and self-signed certificate

Certificate Signing Request possibility of using a command to generate key pair and certificate signing request to certificate authority.

Specify length possibility to specify the length of output key

Specify algorithm possibility to choose between different types of algorithms for key generation

Basic Constraints

Specify Type specify if generated certificate will belong to certificate authority or whether it will be end certificate

Specify path length specify the maximum length of certificate authority chain

1.2.2 Specifications

Table 1.3: Specifications

Tool	CSR signing	Privkey + signed chain	Specify certificate validity	SAN for end certificates	Support for CSP
OpenSSL	Yes	Yes	Yes	Yes	Yes
Keytool	Yes	Yes	Yes	Yes	Yes
GnuPG					
Keystore explorer	Yes	Yes	Yes	Yes	No
XCA	Yes	Yes	Yes	Yes	???
GnuTLS	Yes	???	Yes	Yes	No

Certificate Signing Request signing possibility of signing a certificate signing request with certificate authority's key

Create combination of private key and signed chain possibility of generating private key and chain signed by certificate authority that will be outputed to a single file

Specify certificate validity possibility of choosing how long will the certificate be valid. This should be done by certificate authority.

Setting Subject Alternative Name for end certificates possibility of choosing Subject Alternative Name for end certificates. That should be done by IP's or DNS addresses.

Support for Cryptographic Service Provider whether the use, choosing and changing of Cryptographic Service Provider is supported.

1.3 Conversions

Table 1.4: Conversions

Tools	Conversions			
	Exporting		Direct JKS and PKCS12	Import certificate and private key into a file
	Certificate/chain only from file	Private key only		
OpenSSL	Yes	Yes	No	Yes
Keygen	Yes	No	Yes	Yes
GnuPG				
Keystore explorer	Yes	Yes	Yes	Yes
XCA	Yes	Yes	No	Yes
GnuTLS	Yes	No	No	Yes

1.3.1 Exporting

Certificate or certificate chain from a file

Possibility of extracting certificate or certificate only from a file. Choice of Yes based on possibility of extracting either from a file.

Private key only

Possibility of extracting private key from tool's file storage type of choice.

1.3.2 Direct conversion between Java Keystore and PKCS#12 file

Possibility of direct conversion (by a command of tested tool) between Java KeyStore and PKCS#12 type file.

1.3.3 Importing of certificates and keys into storage files

Possibility of importing (additional?) certificates and keys into storage files of said tool. Yes if it is possible to import or add another certificate or key into storage.

Part II

Tools

Chapter 2

OpenSSL

2.1 General

2.1.1 Type

OpenSSL uses PKCS12 to store keys and or certificates. Certificates and keys made in OpenSSL however are being made into PEM or DER encoded file. Said keys/certificates can then be stored inside single .pfx file. Yes

2.1.2 View and information

Viewing stored certificates

PEM encoded certificates (.pem|.cer|.crt):

```
openssl x509 -in sample_cert.extention -text -noout
```

DER encoded certificates (.der):

```
openssl x509 -in certificate.der -inform der -text -noout
```

Importing PEM or DER encoded keys or certificates into PKCS#12 file:

```
openssl pkcs12 -export -in file.pem \
        -inkey privateKeyFile.key -out file.p12 \
        -name "My Certificate" -certfile othercerts.pem
```

certfile option is used only if importing more certificates into a single PKCS#12 file is wanted.

2.1.3 License

OpenSSL is free to use commercialy, however creating CA is not advised in O'REILLY's Network Security with OpenSSL by J. Viega, M. Messier and P. Chandra on page 59,

> Since OpenSSL's command-line CA functionality was intended primarely as an example of how to use OpenSSL to build a CA, we don't recommend that you attempt to use it in a large production environment.

and also hinted in manual pages:

> The ca command is quirky and at times downright unfriendly.
>
> The ca utility was originally meant as an example of how to do things in a CA . It was not supposed to be used as a full blown CA itself: nevertheless some people are using it for this purpose.
>
> The ca command is effectively a single user command: no locking is done on the various files and attempts to run more than one ca command on the same database can have unpredictable results.

2.2 Generation and signing of certificates

2.2.1 Keys, certificates and basic constraints

Generate keys

Self-signed certificate It is possible to generate private key and self signed certificate with

```
openssl req -x509 -sha256 -nodes -days 365 \
-newkey rsa:[length] -keyout privateKey.key \
        -out certificate_name.crt
```

Yes

Certificate Signing Request It is possible to generate private key and CSR with

```
openssl req -out CSR.csr -new -newkey rsa:[length] \
        -nodes -keyout privateKey.key
```

Specify length Length is specified while generating key/certificate.

Specify algorithm Currently OpenSSL supports Public-key cryptography algorithms: RSA, DSA, Diffie-Hellman key exchange, Elliptic Curve

In the past also support for GOST R 34.10-2001 but as of January 2016 deprecated (https://mta.openssl.org/pipermail/commits/2016-January/003023.html)

Basic constraints

Basic constraints are defined in CA's openssl.cnf `[ v3_req ]` part.

```
[ v3_req ]

basicConstraints=critical,CA:<BOOL_VAL>, pathlen:<maxChainLengthInteger>
```

Specify Type To specify if generated certificate belongs to CA or is end certificate you choose `<BOOL_VAL>` true or false respectively.

Specify path length Specifying of path length is made by setting a `<maxCheinLengthInteger>` to desired value (e.g. if generated certificate should be used only for generating end certificates the value is 0)

2.2.2 Specifications

Certificate Signing Request signing

SR signing can be done by CA created in openSSL by command

```
openssl ca -config ca/openssl.cnf \
        -extensions server_cert -days <validity_time>  \
-notext -md [message digest alg] -in ca/csr/www.example.com.csr.pem \
        -out ca/certs/www.example.com.cert.pem
```

Create combination of private key and signed chain

First creation of the request is needed, after that request must be signed. To combine them together the creation of PKCS#12 file is needed with commands:

```
openssl pkcs12 -export -out outfilename.p12 \
        -in signed_certificate.crt -inkey privateKey.key \
        -chain -CAfile ca-all.crt -password pass:PASSWORD
```

Specify certificate validity

Time how long the certificate will be valid can be given in csr but only as a suggestion. Ultimately the real time of validity is always given by CA signing the CSR.

Setting Subject Alternative Name for end certificates

The Subject Alternative Name is also given in `[ v3_req ]` part of openssl.cnf that CA uses. It can be given as an IP address but also as a DNS name.

```
[ v3_req ]
subjectAltName = @alt_names

[alt_names]
DNS.1 = example1.com
DNS.2 = example2.com
DNS.<next_number> = dns_webaddress.com
```

Support for Cryptographic Service Provider

OpenSSL has native support for Cryptographic Service Provider since version 0.9.7 by creating PKCS#12 file using -CSP option to set the cspname parameter

2.3 Conversions

2.3.1 Exporting

Certificate or certificate chain from a file

To export certificate from a PKCS#12 file use:

```
openssl pkcs12 -in name.[pfx|p12] -nokeys -clcerts -out name.pem
```

To export certificate chain from a PKCS#12 file use:

```
openssl pkcs12 -in name.[pfx|p12] -nokeys -cacerts -out CAchain.pem
```

It should be mentioned that if there are multiple certificates in a chain they are all going to be in same output file.

Private key only

To export an encrypted private key use:

```
openssl pkcs12 -in name.[pfx|p12] -nocerts -out name.encrypted.priv.key
```

To export an unencrypted private key use:

```
openssl pkcs12 -in name.[pfx|p12] -nocerts -nodes -out name.unencrypted.priv.key
```

2.3.2 Direct conversion between Java Keystore and PKCS#12 file

Direct conversion between Java Keystore and PKCS#12 type file is not supported by OpenSSL

2.3.3 Importing certificates and keys into storage files

To import certificate and private key into a single PKCS#12 type file use:

```
openssl pkcs12 -export -out certificate.pfx \
        -inkey privateKey.key -in certificate.crt \
        -certfile CACert.crt
```

Chapter 3

Keytool

3.1 General

3.1.1 Type

Keytool has native support for both PKCS#12 and Java KeyStore type files

3.1.2 View and information

To view information about PEM or DER encoded certificate use:

```
keytool -printcert -file certificate.extention
```

To view information about Jave KeyStore or PKCS#12 encoded file use:

```
keytool -list -v -keystore store_file.[pfx|p12|jks]
```

Viewing Public key part of certificate is not availiable in keytool.

3.1.3 License

Keytool is free to use in production code and is licensed under Apache Licence Version 2.0. For more detailed description look at http://www.apache.org/licenses/LICENSE-2.0.txt

3.2 Generation and signing of certificates

3.2.1 Keys, certificates and basic constraints

Generate keys

Self-signed certificate To generate new key pair and self-signed certificate use:

```
keytool -genkey -keyalg [RSA|DSA] -alias [alisaname] \
        -keystore [keystore_file] -storepass [password] \
        -validity [time] -keysize [length]
```

Keytool natively supports only DSA of length in between 512 and 1024, and is considered to be dangerous to use. RSA with key length of at least 2048 is preffered.

Certificate Signing Request To generate a Certificate Signing Request generation of a key pair with alias is needed:

```
keytool -genkey -alias namealias -keyalg [RSA|DSA] \
        -keysize [length] -keystore [keystore_path | new_keystore_name]
```

To create a CSR you then need to do:

```
keytool -certreq -keyalg [keyalg_from_genkey] -alias [alias_from_genkey] \
        -file certreq.csr -keystore [jks_from_genkey]
```

Where you need to set -keyalg -alias and -keystore as chosen when generating key pair earlier.

Specify length Length is specified when generating keys, RSA support for up to 4096, DSA support from 512 up to 1024.

Specify algorithm Algorithm is specified while generating key pair, currently support for DSA, RSA. EC is needed to be implemented.

Basic constraints

Specify Type With keytool you can make a self-signed CA certificate by using `-ext BC=ca:[true|false]` option.

Specify path length Specifying of path length is possible while making a certificate by using `-ext BC=pathlen:[int path length]` option.

3.2.2 Specifications

Certificate Signing Request signing

Keytool can sign a CSR if a CA certificate and a keystore containing it are avaliable:

```
keytool -gencert -rfc -keystore [root_keystore.jks] \
        -alias [CA_cert_alias] -storepass [store_password] \
        -infile [CSR_file.csr] -validity <int_length> \
        -outfile [output_certificate.crt] -ext BC=ca:[true|false] \
        -ext SAN=ip:<IP_address>,dn:<DNS_address>
```

Create combination of private key and signed chain

You can import a key pair into a Java KeyStore file along with signed chain if the signed chain is provided and a CA certificate is imported into said Java Keystore.

Specify certificate validity

Specification of certificate validity is set when generating key pair in `-validity [time]` option.

Setting Subject Alternative Name for end certificates

To set a Subject Alternative Name for certificate use `-ext san=dns:www.dnsexample.com,ip:1.1.1.1` option which will add SAN to certificate.

Support for Cryptographic Service Provider

Support for Cryptographic Service Provider is native but there is need for a definition of path where the CSP is located.

```
-storetype [keystore_type_of_implementation]
-provider [provider_implementation_name]
-providerpath [provider_implementation_type]
```

3.3 Conversions

3.3.1 Exporting

Certificate or certificate chain from a file

To export certificate with alias `myalias` use:

```
keytool -export -alias myalias -file [output.crt] -keystore [source_keystore.jks]
```

Exported certificate will be DER encoded.

Exporting of certificate chain from a Java KeyStore is not availiable using keytool.

Private key only

Exporting of private key only is not availiable in keytool.

3.3.2 Direct conversion between Java Keystore and PKCS#12 file

Conversion between Java KeyStore and PKCS#12 file is availiable. To convert Java KeyStore file to PKCS#12 use:

```
keytool -importkeystore -srckeystore [existing_keystore.jks] \
        -destkeystore [destination_filename.p12|pfx] -srcstoretype JKS \
        -deststoretype PKCS12 -srcstorepass [.jks password] \
        -deststorepass [same as .jks password] -srcalias [source_alias] \
        -destalias [source_alias] -srckeypass [source_key_password] \
        -destkeypass [source_key_password] -noprompt
```

To convert PKCS#12 to JKS use:

```
keytool -importkeystore -srckeystore [PKCS12_file.pfx|.p12] \
        -srcstoretype pkcs12 -destkeystore [dest_keystore_name.jks] \
        -deststoretype JKS
```

3.3.3 Importing certificates and keys into storage files

To import certificate use:

```
keytool -import -alias [certificate_chosen_alias] \
        -file [certificate_file.ext] -keystore [target_keystore]
```

Importing of keys alone is not supported, you need to fold them into a Java KeyStore or PKCS#12 file first.

Chapter 4

GnuPG

4.1 General

4.1.1 Type

The type of file with which GnuPG works is a .gpg encoded in OpenPGP standard. As I understood the OpenPGP is competing standard to X509 standard.

4.1.2 View and information

4.1.3 License

GnuPG is free to use both for testing and in production code.

4.2 Generation and signing of certificates

4.2.1 Keys, certificates and basic constraints

Generate keys

Self-signed certificate Generation of keys and self-signed certificate is not supported. In older versions it was possigle with `gpgm`

Certificate Signing Request

Specify length

Specify algorithm

Basic constraints

Specify Type

Specify path length

4.2.2 Specifications

Certificate Signing Request signing

Create combination of private key and signed chain

Specify certificate validity

Setting Subject Alternative Name for end certificates

Support for Cryptographic Service Provider

4.3 Conversions

4.3.1 Exporting

Certificate or certificate chain from a file

Private key only

4.3.2 Direct conversion between Java Keystore and PKCS#12 file

4.3.3 Importing certificates and keys into storage files

Chapter 5

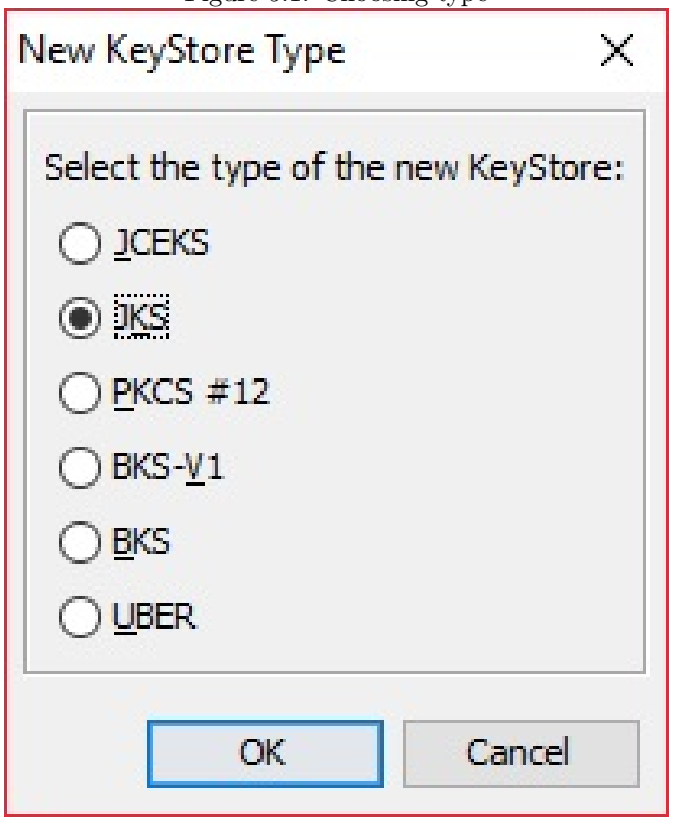
Keystore Explorer

5.1 General

5.1.1 Type

Keystore Explorer supports both Java Keystore and PKCS#12 type files, in addition it has a support for alternatives to JKS such as JCEKS, BKS, BKS-V1, UBER.

Figure 5.1: Choosing type



5.1.2 View and information

You can view certificates and additional information with Keystore Explorer, view public and private key parts of certificates and other info such as subject alternative names by right clicking a certificate and choosing viable option from View option.

5.1.3 License

Keystore Explorer is free to use utility, License is public.

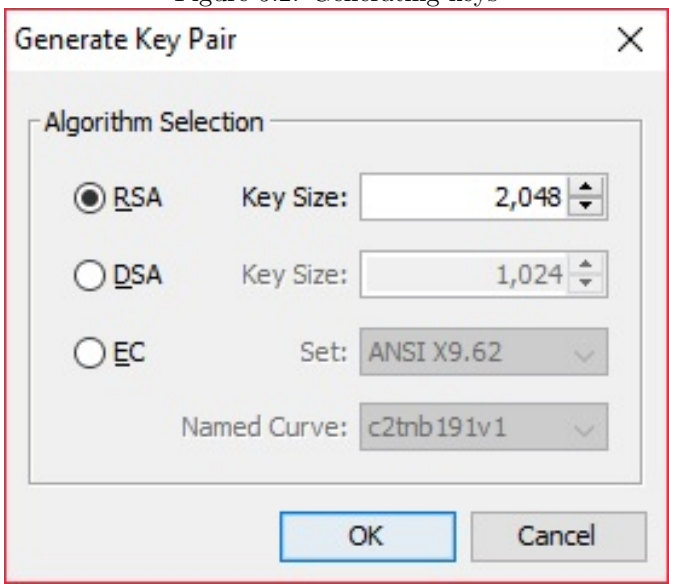
5.2 Generation and signing of certificates

5.2.1 Keys, certificates and basic constraints

Generate keys

Self-signed certificate It is possible to generate a key pair and self-signed certificate, simply click on icon or double click on blank space, choose algorithm and key length. Then you can choose all the other options such as Basic Constraints and Subject Alternative Name.

Figure 5.2: Generating keys



You can also set up default pre-fill of Common Name, Organisational Unit, Organisation etc.

Certificate Signing Request It is possible to make Certificate Signing Request in Keystore Explorer. Simply generate key pair certificate and then right click the generated certificate and choose Generate CSR option.

Specify length Length is specified at the moment of key pair generation.

Specify algorithm Algorithm of keys is specified when generating keys and the certificate hash function is chosen by certificate authority.

Figure 5.3: Generating certificate

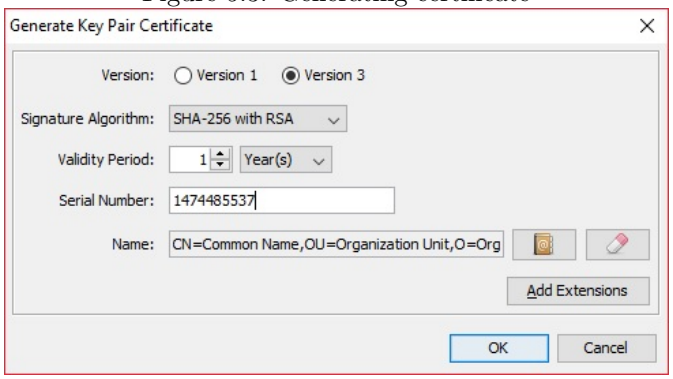
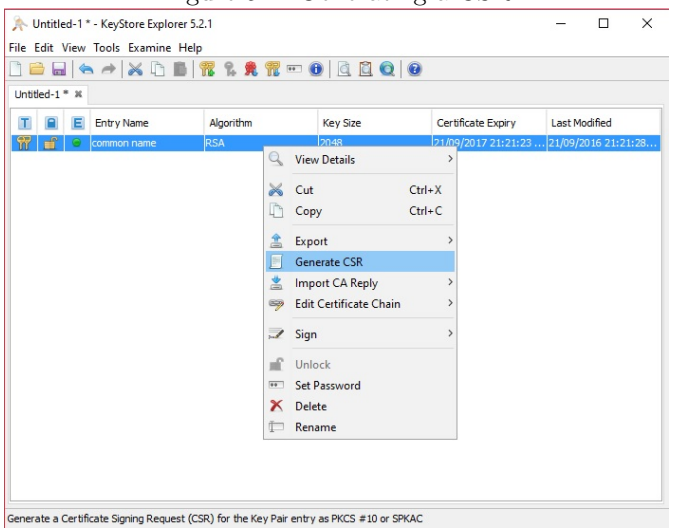


Figure 5.4: Generating a CSR



Basic constraints

Specify Type Specifying type is made while generating keys and certificate by changing v3 portion when asked during generation.

Specify path length Specifying of max path length is made also by changing v3 part of x509 certificate during generation under Basic Constraints type.

5.2.2 Specifications

Certificate Signing Request signing

To sign a Certificate Signing Request simply right-click a Certificate Authority certificate and use the sign-¿CSR option. Then you need to import CA response that was generated in a desired place into a certificate from which the CSR was generated.

Figure 5.5: Basic Constraints

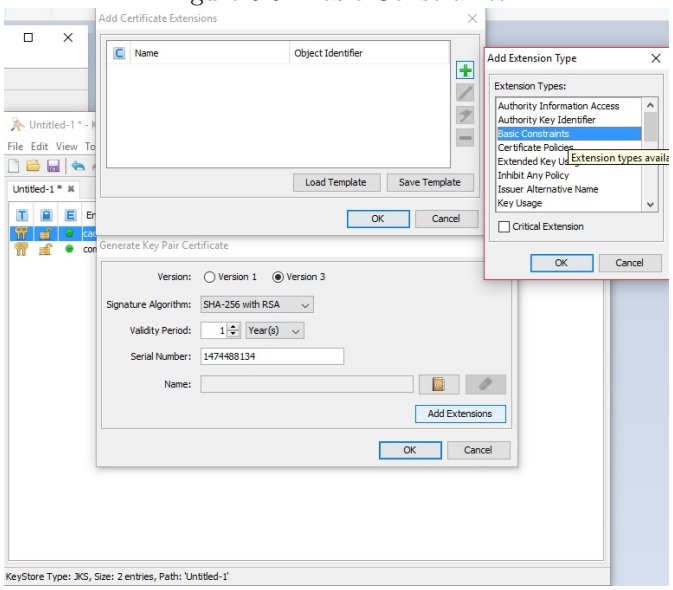
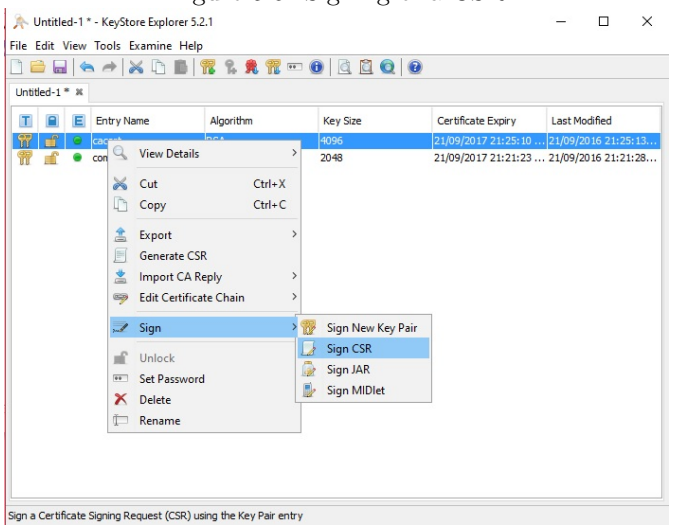


Figure 5.6: Signing of a CSR



Create combination of private key and signed chain

To create combination of private key simply generate new key pair certificate, export private key part, import it back and then import to new certificate of private key a trusted chain.

Specify certificate validity

Desired duration of certificate validity is asked during generation, but given whilst signing a Certificate Signing Request by Certificate Authority

Setting Subject Alternative Name for end certificates

Setting of Subject Alternative Name for certificate is made during generation of certificate, under the add extentions option under the Subject Alternative Name part. Keystore Explorer allows to add both DNS and IP formats of SAN.

Support for Cryptographic Service Provider

As Far as i know there is not support for using CSP.

5.3 Conversions

5.3.1 Exporting

Certificate or certificate chain from a file

It is possible to export a certificate or certificate chain from a keystore by simply right-clicking the chosen certificate and chosing export option.

Private key only

It is possible to export a private key from a keystore by simply right-clicking the chosen certificate and chosing export-¿private key option.

5.3.2 Direct conversion between Java Keystore and PKCS#12 file

To change a type of file simply right-click on the blank space and choose change type option. In addition to changing a pkcs12 file type to JKS it is possible to change type of files between all supported types of Keystore Explorer.

5.3.3 Importing certificates and keys into storage files

It is possible to import a certificate and/or key by chosing Import option.

Chapter 6

XCA

6.1 General

6.1.1 Type

XCA works with PKCS#12 type files, but it is neccessary to make theirs database file which stores and to which you can import certificates, certificate signing requests or keys.

6.1.2 View and information

You can view information about keys, certificates and CSRs by double clicking a chosen file.

6.1.3 License

XCA is under BSD License - it is free to use for testing and also commercialy.

6.2 Generation and signing of certificates

6.2.1 Keys, certificates and basic constraints

Generate keys

Self-signed certificate You can generate key pair by itself and then use it for generation of certificate. If you do not generate key in advance there is option to generate new key whilst generating a certificate. To generate a certificate click on Certificates tab and press New Certificate. There are many options for generating a self-signed certificate, under Source tab you choose a signing algorithm (`MD5|SHA1|SHA224|SHA256|SHA384|SHA512|RIPEMD160`). What must be changed is default hash function because it is set to SHA-1 which is no longer considered safe. Under Subject tab, you shoose Common Name, Organisation, etc. fields of certificate and also choose private key. If you didn't generate key in advance then you can generate a new private key (`RSA|DSA|EC`). For certificate to be accepted by all operating systems you need to choose `Digital Signature & Key Agreement & Certificate Sign` under Key usage tab.

Certificate Signing Request Certificate Signing Request is generated simmilarly as a certificate itself, only difference is that you go under Certificate signing request tab and press New Request.

Specify length Length of keys is specified during generation of keys.

Figure 6.1: Generation of key for new cert

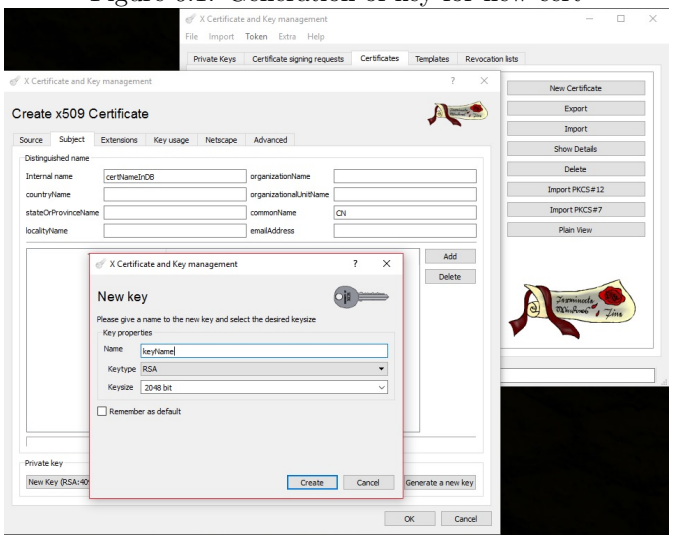
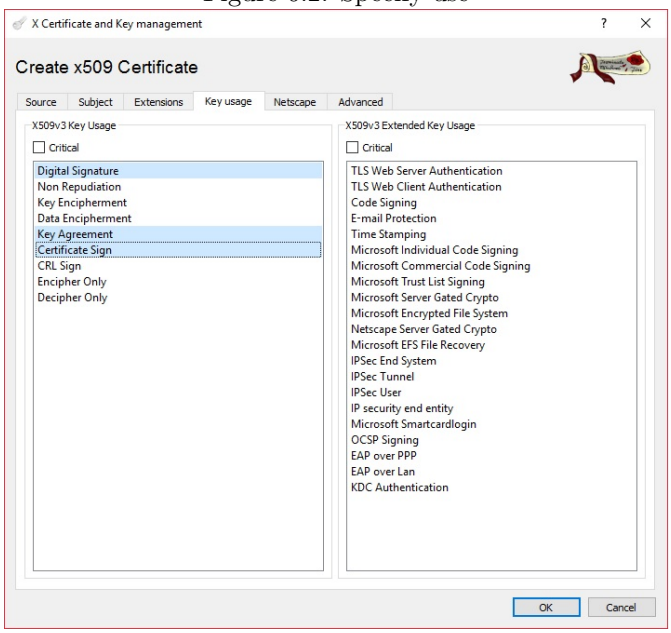


Figure 6.2: Specify use



Specify algorithm Algorithm used for key generation is specified during generation of keys (`RSA|DSA|EC`).

Basic constraints

Specify Type Type of a certificate is chosen under Extensions tab.

Specify path length Path Length is also specified under Extensions tab.

Figure 6.3: Generation of CSR

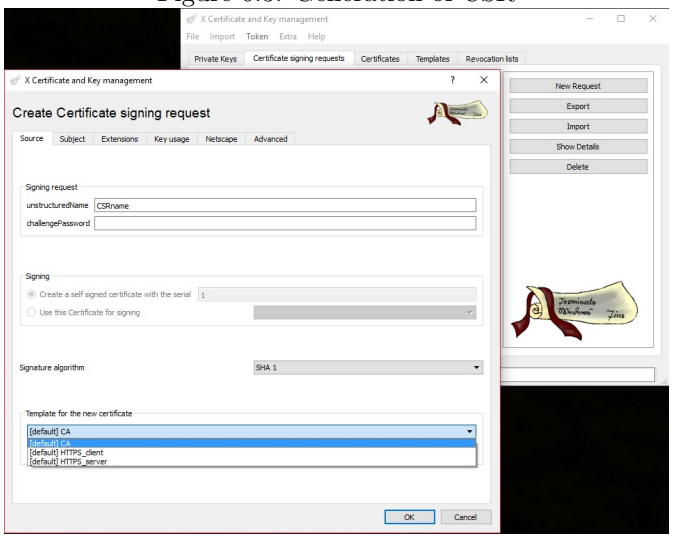
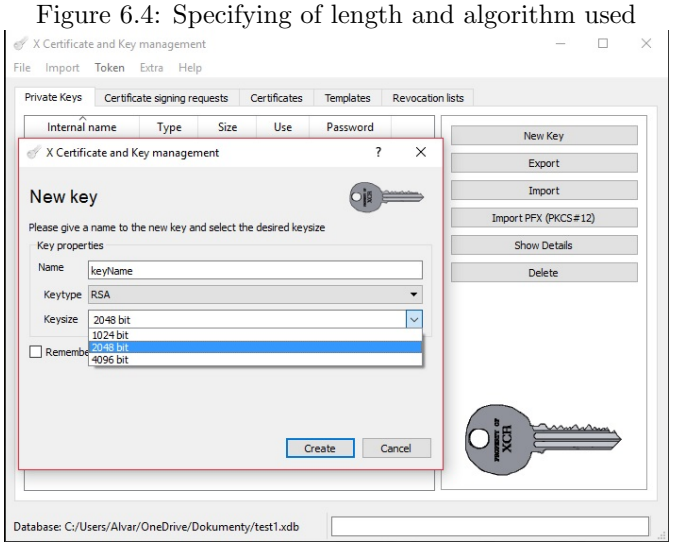


Figure 6.4: Specifying of length and algorithm used



6.2.2 Specifications

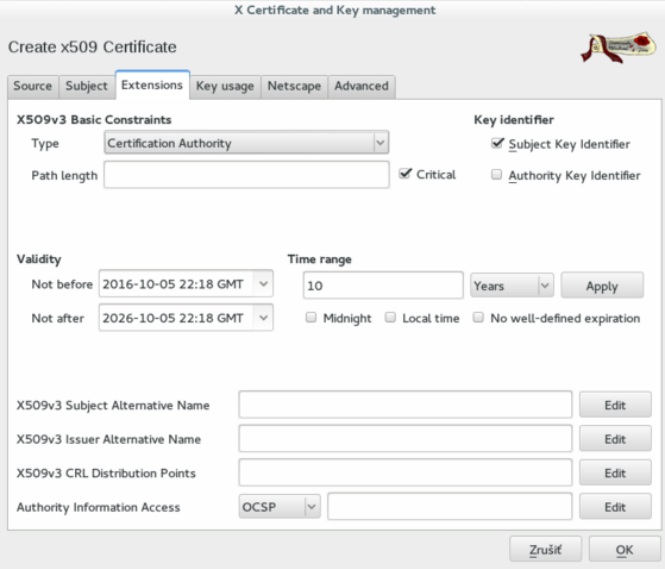
Certificate Signing Request signing

To sign CSR you must have a CA certificate already in XCA's database. If a CA is availiable then you can sign CSR by using New Certificate option under Certificates tab or right click on CSR under Certificate signing requests tab and choose Sign option. Both options are equivalent.

Create combination of private key and signed chain

If the private key is provided PEM encoded it can be imported to XCA's database. If not you need to generate new key pair, then use export option to export private part of a key, delete key pair from database and then import

Figure 6.5: Basic constraints, validity and SAN



private key. to create a combination, simply generate new certificate while using said private key.

Specify certificate validity

Time period during which is a certificate valid is defined under Extensions tab.

Setting Subject Alternative Name for end certificates

Subject Alternative Names are specified in appropriate section under a Extensions tab.

Support for Cryptographic Service Provider

As far as i know there is no support for CSP.

6.3 Conversions

6.3.1 Exporting

Certificate or certificate chain from a file

Possible by using export option, choosing in which format you want to export it.

Private key only

Exporting of private key only directly is availiable.

6.3.2 Direct conversion between Java Keystore and PKCS#12 file

Conversion between PKCS#12 and JKS is not possible.

6.3.3 Importing certificates and keys into storage files

Certificates and keys can be Exported and imported to/from PKCS or PEM type files.

Chapter 7

GnuTLS

7.1 General

7.1.1 Type

GnuTLS works with PKCS#12 PEM encoded files

7.1.2 View and information

To view information about certificate use `certtool --certificate-info --infile [file.pem]` option.

7.1.3 License

GnuTLS is under Gnu Public License free to use commercialy.

7.2 Generation and signing of certificates

7.2.1 Keys, certificates and basic constraints

Generate keys

Self-signed certificate To generate a self-signed certificate it is needed to generate a a private key. Generating of key is done by following command:

```
certtool --generate-privkey --outfile [file.pem] --[rsa|dsa|ecc] \
         --sec-param=[low|legacy|medium|high|ultra]
```

Where `sec-param=<length_alias>` and will be discussed in next parts.

To create a self-signed certificate with created private key use:

```
certtool --generate-self-signed --load-privkey ca-key.pem \
         [--template cert.cfg] --hash=[look@algorithm_part] --outfile ca-cert.pem
```

More options when generating self-signed certificate (or a CSR) you need to create a config file which has structure as follows (copied from a man page):

```
# X.509 Certificate options
#
# DN options

# The organization of the subject.
organization = "Koko inc."

# The organizational unit of the subject.
unit = "sleeping dept."

# The locality of the subject.
# locality =

# The state of the certificate owner.
state = "Attiki"

# The country of the subject. Two letter code.
country = GR

# The common name of the certificate owner.
cn = "Cindy Lauper"

# A user id of the certificate owner.
#uid = "clauper"

# Set domain components
#dc = "name"
#dc = "domain"

# If the supported DN OIDs are not adequate you can set
# any OID here.
# For example set the X.520 Title and the X.520 Pseudonym
# by using OID and string pairs.
#dn_oid = "2.5.4.12 Dr."
#dn_oid = "2.5.4.65 jackal"

# This is deprecated and should not be used in new
# certificates.
# pkcs9_email = "none@none.org"

# An alternative way to set the certificate's distinguished name directly
# is with the "dn" option. The attribute names allowed are:
# C (country), street, O (organization), OU (unit), title, CN (common name),
# L (locality), ST (state), placeOfBirth, gender, countryOfCitizenship,
# countryOfResidence, serialNumber, telephoneNumber, surName, initials,
# generationQualifier, givenName, pseudonym, dnQualifier, postalCode, name,
# businessCategory, DC, UID, jurisdictionOfIncorporationLocalityName,
# jurisdictionOfIncorporationStateOrProvinceName,
# jurisdictionOfIncorporationCountryName, XmppAddr, and numeric OIDs.
```

```
#dn = "cn = Nikos,st = New\, Something,C=GR,surName=Mavrogiannopoulos,2.5.4.9=Arkadias"

# The serial number of the certificate
# Comment the field for a time-based serial number.
serial = 007

# In how many days, counting from today, this certificate will expire.
# Use -1 if there is no expiration date.
expiration_days = 700

# Alternatively you may set concrete dates and time. The GNU date string
# formats are accepted. See:
# http://www.gnu.org/software/tar/manual/html_node/Date-input-formats.html

#activation_date = "2004-02-29 16:21:42"
#expiration_date = "2025-02-29 16:24:41"

# X.509 v3 extensions

# A dnsname in case of a WWW server.
#dns_name = "www.none.org"
#dns_name = "www.morethanone.org"

# An othername defined by an OID and a hex encoded string
#other_name = "1.3.6.1.5.2.2 302ca00d1b0b56414e5245494e2e4f5247a11b3019a006020400000002a10f300d1b047
#other_name_utf8 = "1.2.4.5.6 A UTF8 string"
#other_name_octet = "1.2.4.5.6 A string that will be encoded as ASN.1 octet string"

# Allows writing an XmppAddr Identifier
#xmpp_name = juliet@im.example.com

# Names used in PKINIT
#krb5_principal = user@REALM.COM
#krb5_principal = HTTP/user@REALM.COM

# A subject alternative name URI
#uri = "http://www.example.com"

# An IP address in case of a server.
#ip_address = "192.168.1.1"

# An email in case of a person
email = "none@none.org"

# TLS feature (rfc7633) extension. That can is used to indicate mandatory TLS
# extension features to be provided by the server. In practice this is used
# to require the Status Request (extid: 5) extension from the server. That is,
# to require the server holding this certificate to provide a stapled OCSP response.
# You can have multiple lines for multiple TLS features.
```

```
# To ask for OCSP status request use:
#tls_feature = 5

# Challenge password used in certificate requests
challenge_password = 123456

# Password when encrypting a private key
#password = secret

# An URL that has CRLs (certificate revocation lists)
# available. Needed in CA certificates.
#crl_dist_points = "http://www.getcrl.crl/getcrl/"

# Whether this is a CA certificate or not
#ca

# Subject Unique ID (in hex)
#subject_unique_id = 00153224

# Issuer Unique ID (in hex)
#issuer_unique_id = 00153225

#### Key usage

# The following key usage flags are used by CAs and end certificates

# Whether this certificate will be used to sign data (needed
# in TLS DHE ciphersuites). This is the digitalSignature flag
# in RFC5280 terminology.
signing_key

# Whether this certificate will be used to encrypt data (needed
# in TLS RSA ciphersuites). Note that it is preferred to use different
# keys for encryption and signing. This is the keyEncipherment flag
# in RFC5280 terminology.
encryption_key

# Whether this key will be used to sign other certificates. The
# keyCertSign flag in RFC5280 terminology.
#cert_signing_key

# Whether this key will be used to sign CRLs. The
# cRLSign flag in RFC5280 terminology.
#crl_signing_key

# The keyAgreement flag of RFC5280. It's purpose is loosely
# defined. Not use it unless required by a protocol.
#key_agreement

# The dataEncipherment flag of RFC5280. It's purpose is loosely
```

```
# defined. Not use it unless required by a protocol.
#data_encipherment

# The nonRepudiation flag of RFC5280. It's purpose is loosely
# defined. Not use it unless required by a protocol.
#non_repudiation

#### Extended key usage (key purposes)

# The following extensions are used in an end certificate
# to clarify its purpose. Some CAs also use it to indicate
# the types of certificates they are purposed to sign.


# Whether this certificate will be used for a TLS client;
# this sets the id-kp-serverAuth (1.3.6.1.5.5.7.3.1) of
# extended key usage.
#tls_www_client

# Whether this certificate will be used for a TLS server;
# This sets the id-kp-clientAuth (1.3.6.1.5.5.7.3.2) of
# extended key usage.
#tls_www_server

# Whether this key will be used to sign code. This sets the
# id-kp-codeSigning (1.3.6.1.5.5.7.3.3) of extended key usage
# extension.
#code_signing_key

# Whether this key will be used to sign OCSP data. This sets the
# id-kp-OCSPSigning (1.3.6.1.5.5.7.3.9) of extended key usage extension.
#ocsp_signing_key

# Whether this key will be used for time stamping. This sets the
# id-kp-timeStamping (1.3.6.1.5.5.7.3.8) of extended key usage extension.
#time_stamping_key

# Whether this key will be used for email protection. This sets the
# id-kp-emailProtection (1.3.6.1.5.5.7.3.4) of extended key usage extension.
#email_protection_key

# Whether this key will be used for IPsec IKE operations (1.3.6.1.5.5.7.3.17).
#ipsec_ike_key

## adding custom key purpose OIDs

# for microsoft smart card logon
# key_purpose_oid = 1.3.6.1.4.1.311.20.2.2

# for email protection
```

```
# key_purpose_oid = 1.3.6.1.5.5.7.3.4

# for any purpose (must not be used in intermediate CA certificates)
# key_purpose_oid = 2.5.29.37.0

### end of key purpose OIDs

### Adding arbitrary extensions
# This requires to provide the extension OIDs, as well as the extension data in
# hex format. The following two options are available since GnuTLS 3.5.3.
#add_extension = "1.2.3.4 0x0AAB01ACFE"

# As above but encode the data as an octet string
#add_extension = "1.2.3.4 octet_string(0x0AAB01ACFE)"

# For portability critical extensions shouldn't be set to certificates.
#add_critical_extension = "5.6.7.8 0x1AAB01ACFE"

# When generating a certificate from a certificate
# request, then honor the extensions stored in the request
# and store them in the real certificate.
#honor_crq_extensions

# Alternatively only specific extensions can be copied.
#honor_crq_ext = 2.5.29.17
#honor_crq_ext = 2.5.29.15

# Path length contraint. Sets the maximum number of
# certificates that can be used to certify this certificate.
# (i.e. the certificate chain length)
#path_len = -1
#path_len = 2

# OCSP URI
# ocsp_uri = http://my.ocsp.server/ocsp

# CA issuers URI
# ca_issuers_uri = http://my.ca.issuer

# Certificate policies
#policy1 = 1.3.6.1.4.1.5484.1.10.99.1.0
#policy1_txt = "This is a long policy to summarize"
#policy1_url = http://www.example.com/a-policy-to-read

#policy2 = 1.3.6.1.4.1.5484.1.10.99.1.1
#policy2_txt = "This is a short policy"
#policy2_url = http://www.example.com/another-policy-to-read

# Name constraints
```

```
# DNS
#nc_permit_dns = example.com
#nc_exclude_dns = test.example.com

# EMAIL
#nc_permit_email = "nmav@ex.net"

# Exclude subdomains of example.com
#nc_exclude_email = .example.com

# Exclude all e-mail addresses of example.com
#nc_exclude_email = example.com

# IP
#nc_permit_ip = 192.168.0.0/16
#nc_exclude_ip = 192.168.5.0/24
#nc_permit_ip = fc0a:eef2:e7e7:a56e::/64


# Options for proxy certificates
#proxy_policy_language = 1.3.6.1.5.5.7.21.1


# Options for generating a CRL

# The number of days the next CRL update will be due.
# next CRL update will be in 43 days
#crl_next_update = 43

# this is the 5th CRL by this CA
# Comment the field for a time-based number.
#crl_number = 5

# Specify the update dates more precisely.
#crl_this_update_date = "2004-02-29 16:21:42"
#crl_next_update_date = "2025-02-29 16:24:41"

# The date that the certificates will be made seen as
# being revoked.
#crl_revocation_date = "2025-02-29 16:24:41"
```

Certificate Signing Request Creating of CSR is made by:

```
certtool --generate-request --load-privkey [keyfile.pem] \
         [--template cert.cfg]--outfile [csrfilename.pem]
```

when not using template it uses default values

Specify length Length is specified when generating keys by `--sec-param=[low|legacy|medium|high|ultra]` where length is determined by alias and algorithm. For RSA:

```
low    = 1024 bit
legacy = 1776 bit
medium = 2048 bit
high   = 3072 bit
ultra  = 15424 bit
```

For DSA:

```
low    = 1024 bit
legacy = 2048 bit
medium = 2048 bit
high   = 3072 bit
ultra  = 3072 bit
```

For ECDSA:

```
low    = 160 bit
legacy = 192 bit
medium = 224 bit
high   = 256 bit
ultra  = 512 bit
```

Also, to choose exact bit length you can use `--bits=<length>` option - tested on `--rsa --bits=32768` option which generated 32kbit RSA key.

To define a hash algorithm use `--hash=[SHA1|RMD160|SHA256|SHA384|SHA512` option while generating a certificate.

Specify algorithm Algorithm for key generation is specified in `--[rsa|dsa|ecc]` option where rsa = RSA, dsa = DSA, ecc = ECDSA.

Basic constraints

Specify Type To specify whether certificate should be CA or End cert write `ca` into the cert.cfg file and use that as a template for generation.

Specify path length Path length is specified by `path_len=<value>` option in cert.cfg file, where value is max chain length (use -1 for undefined value).

7.2.2 Specifications

Certificate Signing Request signing

Possible

Create combination of private key and signed chain

When you create a certificate by signing a request to sign key only, you can validate chain.

Specify certificate validity

Validity is specified by changing cert.cfg file used while generating a certificate, the part `expiration_days = <value>` Alternatively, you cen set NOT BEFORE and NOT AFTER times by changing `activation_date = <NOT_BEFORE_STRING` and `expiration_date = <NOT_AFTER_STRING>` where the times are written in GNU date string format.

Setting Subject Alternative Name for end certificates

SAN is set by adding options to cert.cfg :

```
dns_name = "www.dnsname1.com"
dns_name = "www.dnsname2.com"
  ...
dns_name = "www.lastdns.com"

uri = "uri1"
...

ip_address = "0.0.0.1"
ip_address = "0.0.0.2"
   ...
```

Support for Cryptographic Service Provider

GnuTLS does not have support for CSP.

7.3 Conversions

7.3.1 Exporting

Certificate or certificate chain from a file

Private key only

It is not possible to export private key only, however for proper use of GnuTLS keys need to be stored on drive or imported to PKCS#12 file.

7.3.2 Direct conversion between Java Keystore and PKCS#12 file

Not possible.

7.3.3 Importing certificates and keys into storage files

Possible. - need to run up some facts.

Chapter 8

Tool Page Template

8.1 General

8.1.1 Type

8.1.2 View and information

8.1.3 License

8.2 Generation and signing of certificates

8.2.1 Keys, certificates and basic constraints

Generate keys

Self-signed certificate

Certificate Signing Request

Specify length

Specify algorithm

Basic constraints

Specify Type

Specify path length

8.2.2 Specifications

Certificate Signing Request signing

Create combination of private key and signed chain

Specify certificate validity

Setting Subject Alternative Name for end certificates

Support for Cryptographic Service Provider

8.3 Conversions

8.3.1 Exporting

Certificate or certificate chain from a file

Private key only

8.3.2 Direct conversion between Java Keystore and PKCS#12 file

8.3.3 Importing certificates and keys into storage files