Review and comparison of tools for managing X.509 certificates and keys

Oliver Bodnár

November 2016

Contents

Part I

Introduction

Primary goal of this work is to compare and review some of the basic functionalities *(e.g. generation of keys and certificates, ability to create a certificate signing request,...)* of tools working with keys and certificates in X.509 public key infrastructure. Comparison of tools is necessary to understand that ability of a tool to do everything that was tested does not mean that it is the best tool to use in every situation.

Part II

Overview

Chapter 1

Overview Tables

1.1 General

Table 1.1: Overwiew General

Tools	General				
	Type	View information about certificate	License	Operating System	Tool Type
OpenSSL	PKCS12	Yes	Public	All	Command Line
Keytool	JKS	Yes	Public	All	Command Line
GnuPG	None	No	Public	All	Command Line
Keystore Explorer	Both	Yes	Public	Win | OSX	Graphical Interface
XCA	PKCS12	Yes	Public	All	Graphical Interface
GnuTLS	PKCS12	Yes	Public	UNIX	Command Line
Makecert and pvk2pfx	PKCS12	No	Public	Windows	Command Line
Windows PowerShell PKI Module	PKCS12	Yes	Public	Windows	Command Line

1.1.1 Type

This section defines type of storage file in which the certificates or keys are saved. Options are:

- *PKCS#12* - PKCS#12 encoded file, usual extensions are `.pfx` and `.p12`
- *JKS* - Java KeyStore type file, usual extension is `.jks`

1.1.2 View information about certificate

This section shows whether it is possible to view information certificates and keys and additional information.

- *Yes* - viewing information about certificates and keys is availiable. This includes length, used algorithm, fingerprint.
- *No* - tool does not have an option to view any information about certificates or keys.

1.1.3 License

Type of license and possibility of using said tool for testing or production code.

- *Public* - No restrictions for usage, can be used commercialy.

- *Testing* - Tool is free to use non-commercialy. For commercial acquisition of paid license is needed.
- *Paid* - Every type of usage is paid for. Restrictive version of product may be available, but to use all the features and be able to use in commercial enviroment paid license is needed.

Public 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 next chapters under each tool.

1.1.4 Operating Systems

In this part the Operating System under which the tool is implemented is defined.

- *OS X* | *Windows* | *LINUX* - Tool is implemented under any of the mentioned OS, `LINUX` defines most common distributions of Linux systems *(Debian or RPM based)*.
- *UNIX* - Tool is implemented under UNIX-like systems, which is same as *OS X* | *LINUX* option.
- *All* - Tool is implemented under every aforementioned operating system.

1.1.5 Tool Type

This section shows what type of input is used in Tool.

- *Command Line* - tool is used with commands on some sort of console (PowerShell, Command Prompt, Terminal)
- *Graphical Interface* - tool is used with Graphical Interface, main interaction is by user using mouse

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	No	Yes	No	No	No	No
Keystore explorer	Yes	Yes	Yes	Yes	Yes	Yes
XCA	Yes	Yes	Yes	Yes	Yes	Yes
GnuTLS	Yes	Yes	Yes	Yes	Yes	Yes
Makecert and pvk2pfx	Yes	No	Yes	Yes	Yes	Yes
Windows PowerShell PKI Module	Yes	No	No	No	No	No

Generate keys

Self-signed certificate Capability of tool to create a self-signed certificate

- *Yes* - there is a capability of a tool to create self-signed certificate, not necessarily a single command.
- *No* - under no circumstances is it possible to create self-signed certificate using only tool itself.

Certificate Signing Request Capability of tool to create a Certificate Signing Request.

- *Yes* - The tool is capable of creating a certificate signing request, but does not necessarily generates or packs private key with it.
- *No* - The tool is not capable of creating any type of certificate signing request.

Specify length Capability of tool to specify length of created key.

- *Yes* - changing key length is possible either in fixed range or by declaring required length *(in bits)*.
- *No* - not possible to change key length by any means.

Specify algorithm

- *Yes* - changing key or at least hash algorithm is possible .
- *No* - not possible to change any algorithm by any means.

Basic Constraints

Specify Type

- *Yes* - Specification if created certificate will be a certificate authority or end certificate is possible.
- *No* - Created certificate is of fixed type, either end certificate or certificate authority.

Specify Path Length

- *Yes* - Specification of chain length value is available .
- *No* - The chain length has a fixed value.

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	No	No	No	No	No
Keystore explorer	Yes	Yes	Yes	Yes	No
XCA	Yes	Yes	Yes	Yes	No
GnuTLS	Yes	No	Yes	Yes	No
Makecert and pvk2pfx	No	No	Yes	No	Yes
Windows PowerShell PKI Module	No	No	Yes	Yes	Yes

Certificate Signing Request signing *Yes* means that the tool supports signing of certificate signing request. *No* if it does not support signing of certificate signing requests.

Create combination of private key and signed chain *Yes* if it is possible to combine private key and corresponding chain into a single file, *No* - otherwise.

Specify certificate validity *Yes* - Possibility of choosing how long the certificate will be valid. Done by certificate authority. *No* - If there is no option to change how long the certificate will be valid, so the duration is a fixed ammount.

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

- *Yes* - possibility of setting a subject alternative name for certificate. It can be DNS name or IP address.
- *No* - not possible to choose any subject alternative name.

Support for Cryptographic Service Provider

- *Yes* - if there is option to change a cryptographic service provider.
- *No* - the cryptographic service provider canot be changed.

1.3 Conversions

Table 1.4: Conversions

Tools	Conversions			
	Exporting		Direct JKS and PKCS12	Import certificate and private key into a file
	Certificate or chain from file	Private key only		
OpenSSL	Yes	Yes	No	Yes
Keytool	Yes	No	Yes	Yes
GnuPG	No	No	No	No
Keystore explorer	Yes	Yes	Yes	Yes
XCA	Yes	Yes	No	Yes
GnuTLS	Yes	No	No	Yes
Makecert	No	No	No	Yes
Windows PowerShell PKI Module	Yes	Yes	No	Yes

1.3.1 Exporting

Certificate or certificate chain from a file

- *Yes* - capability of tol to exract certificate or certificate chain from a single file.
- *No* - if extracting is not available.

Private key only

- *Yes* - private key can be exported to a single file.
- *No* - private key extracting is not allowed.

1.3.2 Direct conversion between Java Keystore and PKCS#12 file

- *Yes* - if tool has a capability to convert Java Keystore to PKCS#12 file and vica versa.
- *No* - if it doesn't.

1.3.3 Importing of certificates and keys into storage files

- *Yes* - it is possible to import certificates and keys into a tools storage type file *(e. g. PKCS#12 file)*.
- *No* - it is not possible to import a certificate into a file.

Part III

Tools

Chapter 2

OpenSSL

2.1 General

Tested under Fedora 21, OpenSSL version 1.0.1k-fips

OpenSSL heavily relies on a configuration file that needs to be prepared along with a default directory sequence in advance. Hierarchy of that directory sequence can be seen in supplied OpenSSL.cnf file under `[ CA_default ]` part.

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.

2.1.1 Type

OpenSSL uses PKCS12 to store keys and certificates. Certificates and keys generated in OpenSSL are being made into PEM or DER encoded file. PEM or DER encoded keys and certificates can then be stored inside single PKCS#12 file.

2.1.2 View 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. Full text is available at `https://www.openssl.org/source/license.html`

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
```

However for any other type of Key generating algorithm, it is better to create key separately. There are options to create a key in one command, but they are quirky.

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 `<maxChainLengthInteger>` to desired value (e.g. if generated certificate/key should be used only for signing end certificates the value is 0)

2.2.2 Specifications

Certificate Signing Request signing

CSR 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 is chosen by CA during signing the CSR, where in command for signing certificate signing request we choose desired duration in days by changing `-days <velidity_time>` part.

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 noted 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
```

2.4 Note

However using self-signed certificate made with `ca` option as CA certificates is not ideal - 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.*

Chapter 3

Keytool

3.1 General

Tested under Fedora 21 and OpenSUSE 42.1, openjdk version 1.8.0_65

3.1.1 Type

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

3.1.2 View and information

When viewing certificates, command for PEM and DER certificates will return errors with command for PKCS#12 and JKS files.

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 [aliasname] \
        -keystore [keystore_file.jks] -storepass [password] \
        -validity [days] -keysize [length]
```

It should be noted that afterwards, when user is prompted to input First and Last name it is equal to Common Name parameter.

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 preferred.

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 as stated in manual, but i managed to create key lengths of bigger value whilst generating.

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

To choose signing algorithm use `-sigalg` option which can have values of `[SHA1|MD5|SHA256|SHA512]with[RSA|DSA]`. This changes with change of Cryptographic service provider.

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 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 along with a signed chain if it is provided in PKCS#12 file format.

```
keytool -importkeystore -srckeystore [pkcs12_file].[pfx|p12] \
        -srcstoretype pkcs12 -destkeystore <output_name>.jks
```

Specify certificate validity

Specification of certificate validity is set when generating key pair in `-validity [time]` option. But changed by CA when being signed.

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 directly in keytool. It is possible by changing keystore type to PKCS#12 and then using other tool such as OpenSSL.

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

Tested under Fedora 21, OpenSUSE 42.1, Debian Wheezy, gpgsm (GnuPG) 2.0.29, libgcrypt 1.6.3, libksba 1.3.2 and also MS Windows 10 version 1607 Build 14393.447, MS Windows 8.1 Pro 6.3.9600, gpgsm (GnuPG) 2.0.30 (Gpg4win 2.3.3), libgcrypt 1.6.6, libksba 1.3.4

4.1.1 Type

GnuPG can make and view X509 Certificate Signing Requests

4.1.2 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

Certificate Signing Request Possible by using `gpgsm --gen-key > <output_file>` command, but private key is not a part of output CSR.

4.3 Note

GnuPG is unable to do anything besides creating a certificate signing request in X.509 public key infrastructure, that does not include a private key within itself.

Chapter 5

Keystore Explorer

5.1 General

Tested under MS Windows 10 version 1607 Build 14393.447, Keystore Explorer version 5.2.1

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.

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 5.3)*

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. *(Figure 5.4)*

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

Figure 5.1: Choosing type

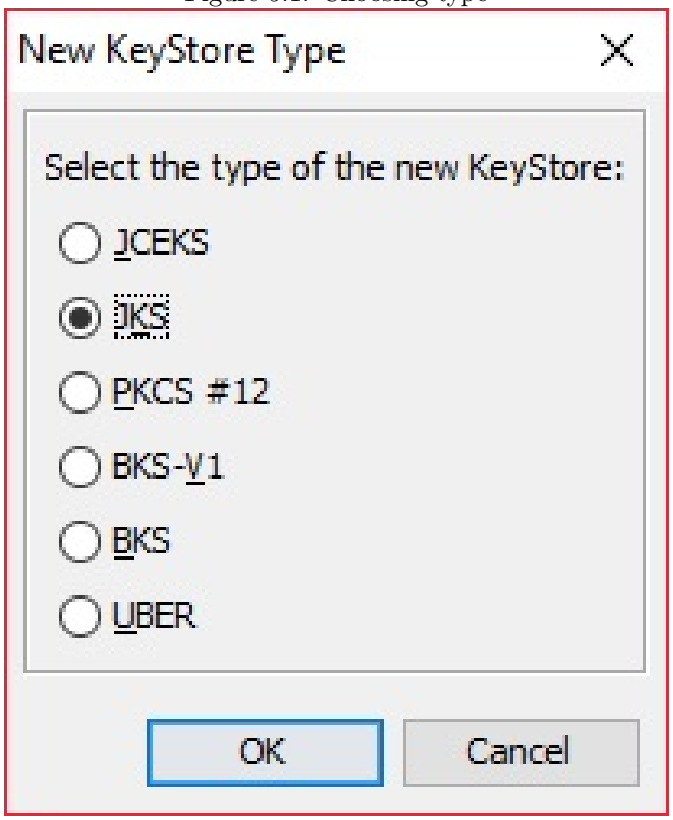


Figure 5.2: Generating keys

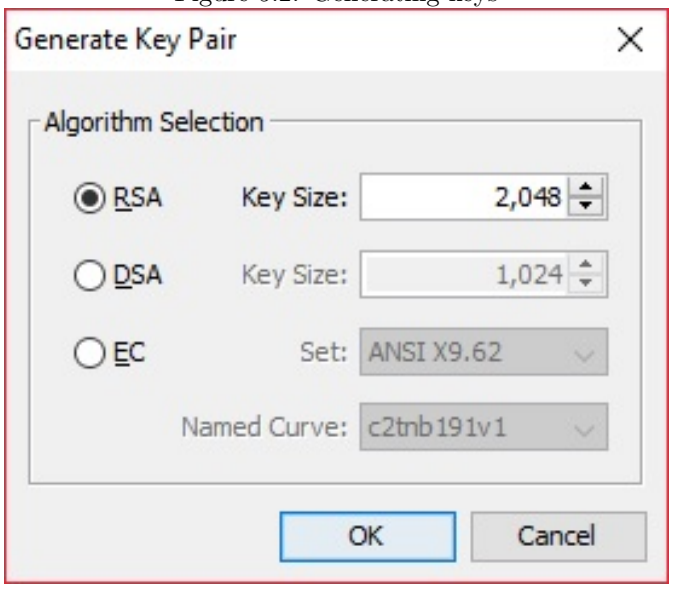


Figure 5.3: Generating certificate

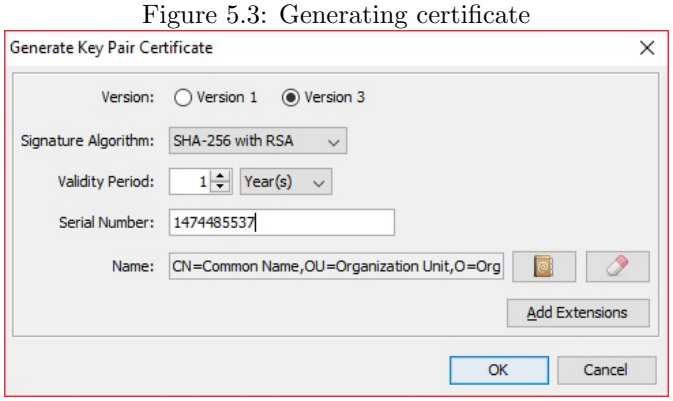
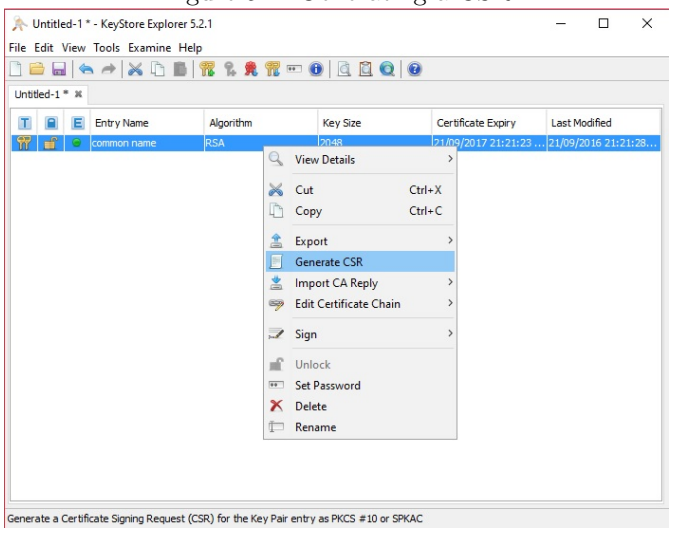


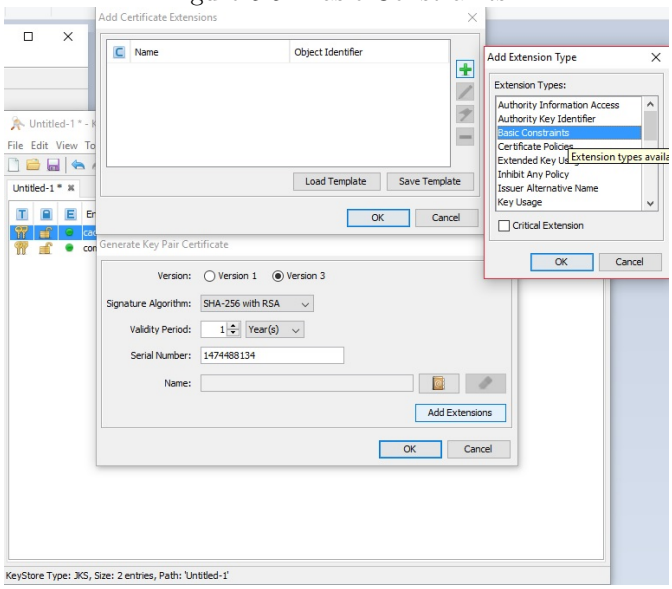
Figure 5.4: Generating a CSR



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

Basic constraints

Figure 5.5: 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.*(Figure 5.5)*

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 and saved in a desired place into a certificate from which the CSR was generated. *(Figure 5.6)*

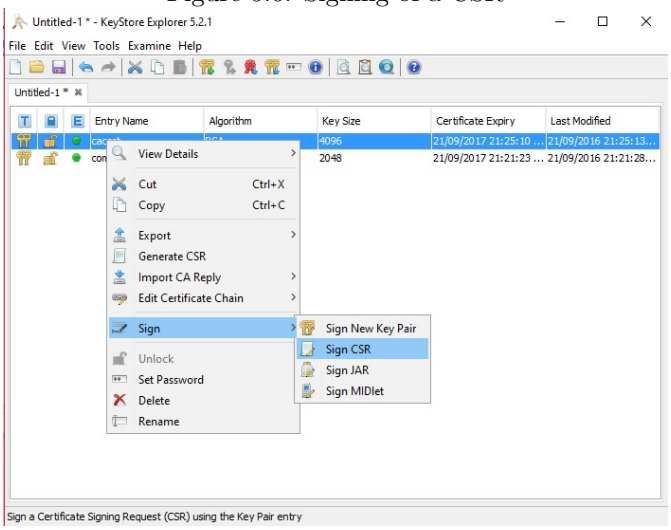
Create combination of private key and signed chain

To create combination of private key and signed chain 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

Figure 5.6: Signing of a CSR



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.

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 certificates and keys by chosing Import option.

Chapter 6

XCA

6.1 General

Tested under Windows 10 version 1607 Build 14393.447 and Fedora 21, XCA version 1.3.2 using OpenSSL version 1.0.2d

6.1.1 Type

XCA works with PKCS#12 type files, but it is neccessary to make their 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 distributed under BSD License - it is free to use for testing and also commercially.

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.*Figure 6.1*

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.

Hash algorithm is one of: `MD5, SHA1, SHA224, SHA256, SHA384, SHA512, RIPEMD160`. It needs to be changed because hash function 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. *(Figure 6.2)*

Figure 6.1: Generation of key for new cert

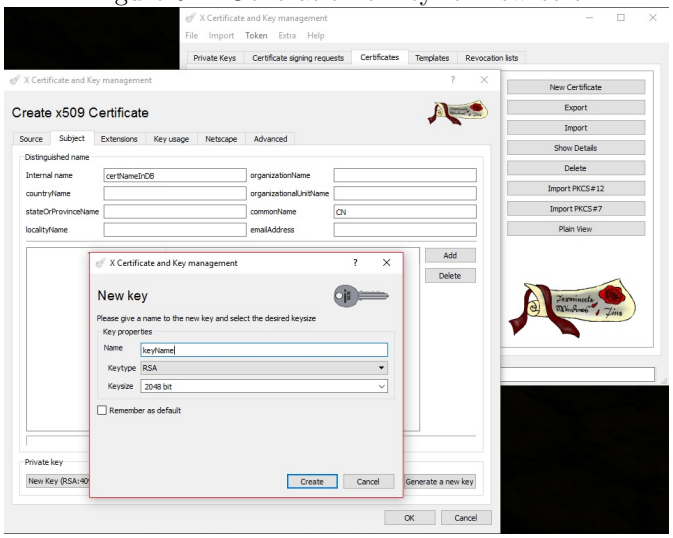
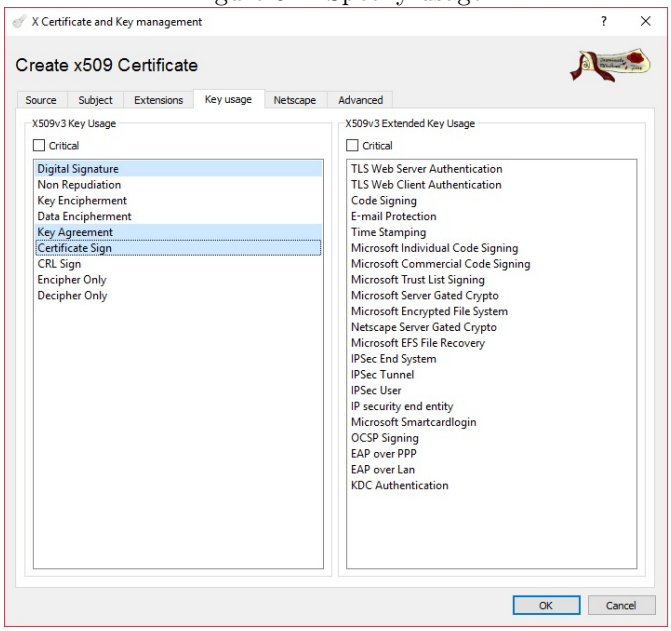


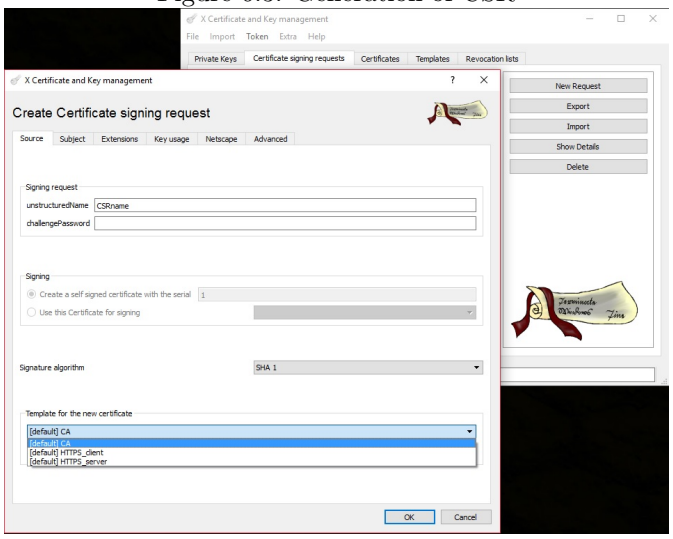
Figure 6.2: Specify usage



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

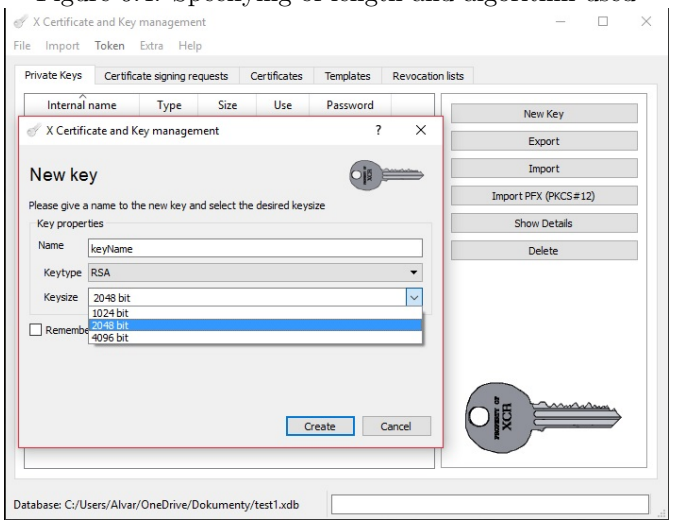
Specify length Length of keys is specified during generation of keys.

Figure 6.3: Generation of CSR



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

Figure 6.4: Specifying of length and algorithm used

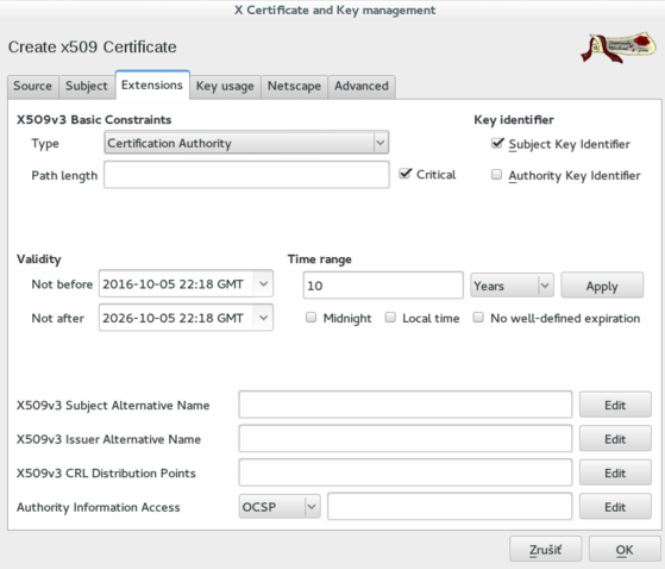


Basic constraints

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

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

Figure 6.5: Basic constraints, validity and SAN



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:

1. using New Certificate option under Certificates tab
2. 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 private key. To create a combination, simply generate new certificate while using said private key.

To export certificate with private key and certificate chain, simply use Export option under *Certificates* tab. There you can choose if you wish to export it in a single *PKCS#12* type file or certificate and key separately, PEM encoded.

Specify certificate validity

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

Setting Subject Alternative Name for end certificates

Subject Alternative Names are specified in appropriate section under an Extensions tab. *(Figure 6.5)*

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 Importing certificates and keys into storage files

Certificates and keys can be Exported and imported to/from PKCS#12 or PEM type files.

Chapter 7

GnuTLS

7.1 General

Tested under Fedora 21, certtool version 3.3.16

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 commercially.

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 private key first. This 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 is in appendix *GnuTLS.cfg* For whole list of options in a config file look up `man certtool`.

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 asks for:

- Common Name, Organizational Unit, Organization, ...
- Subject's domain component
- Subject Alternative Name
- If the certificate will be CA
- What the certificate will be used for *(signing | encryption)*
- Whether the certificate will be TLS web client/server certificate

Table 7.1: Bit length of security parameter options

	low	legacy	medium	high	ultra
RSA	*1024 bit*	*1776 bit*	*2048 bit*	*3072 bit*	*15424 bit*
DSA	*1024 bit*	*2048 bit*	*2048 bit*	*3072 bit*	*3072 bit*
ECDSA	*160 bit*	*192 bit*	*224 bit*	*256 bit*	*512 bit*

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

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

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

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

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 by command:

```
certtool --generate-certificate --load-request <csr_file> \
         --load-ca-certificate <path_to_ca_cert> \
         --load-ca-privkey <ca_cert_privkey_file>
         --outfile <name>.pem
```

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"
   ...
```

7.3 Conversions

7.3.1 Exporting

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 Importing certificates and keys into storage files

To import certificate and key into a PKCS#12 file use command:

```
certtool --load-certificate <in_cert> \
         --load-privkey <in_key> \
         --to-p12 --outder --outfile <name>.[p12|pfx]
```

Chapter 8

Makecert and pvk2pfx

8.1 General

Tested under MS Windows 10 version 14393.447, makecert.exe version 6.3.9600.17298
Makecert is deprecated by Microsoft itself as cited on MSDN MakeCert site.

8.1.1 Type

Makecert can work with PKCS#12 type files, however default output is in `.cer` for certificate and `.pvk` for private key. To tie together key and certificate you need to combine them into PKCS#12 type file with `pvk2pfx` tool (which is also part of Windows SDK). In addition to use the tool you need to start a Developer Command Prompt for Visual Studio (on Windows 10).

8.1.2 License

It is free to use, falls under the Windows SDK.

8.2 Generation and signing of certificates

8.2.1 Keys, certificates and basic constraints

Generate keys

Self-signed certificate Generating key and self-signed certificate that can be used as CA is not straightforward with Makecert.

```
makecert -n "CN=Common Name, OU=Organization Unit, O=..." -cy [authority|end] \
 -a [sha256|sha384|sha512] -sv <key_file_name>.pvk -r \
 -len <int key_length> <cert_name>.cer
```

Where:

- `-n` sets credentials
- `-cy` defines certificate type
- `-a` chooses signature algorithm

- `-sv` chooses name for file where the private key is stored
- `-r` to create a self-signed certificate
- `-len` for key length and without option choose the filename.

After entering this command you will be prompted to enter private key file password.

After creating certificate and private key file successfully to use self-signed certificate as a CA you need to join certificate file and private key into PKCS#12 type file. This is made by using a pvk2pfx tool already implemented in Windows SDK.

```
pvk2pfx -spc <cert_file> -pvk <key_file>.pvk -pfx <pfx_file_name>.pfx \
        [-pi <key_password> -po <pfx_password]
```

Where:

- `-pi` option is used if there is a password on key file
- `-po` is desired pfx file password. If no password for pfx is given makecert defaults it to key file password.

Specify length Length is specified by `-len <int key_length>` option.

Specify algorithm To specify hash algorithm use `-a [sha256|sha384|sha512]` option. Supported are also SHA1 and MD5 but those are considered insecure.

To specify key algorithm you must change CSP.

Basic constraints

Specify Type It is possible with `-cy [authority|end]` option.

Specify path length Path length is specified by `-h <int length>` option while creating a certificate.

8.2.2 Specifications

Specify certificate validity

Defaults from date of creation until 2039, to change use `-b <mm/dd/yyyy -> from> -e <mm/dd/yyyy -> until>` option.

Alternatively you can use `-m <int months>` option where you choose for how many months the certificate will be valid.

Support for Cryptographic Service Provider

There is support for CSP in makecert by using `-sp <provider_name> -sy <provider_type>` defines the CryptoAPI provider for subject.

8.3 Conversions

8.3.1 Exporting

Private key only

Private key is stored and generated while generating a certificate but the type .pvk is uncommon. It is Microsoft's own key storage type more on `http://www.drh-consultancy.demon.co.uk/pvk.html`

8.3.2 Importing certificates and keys into storage files

For importing of keys and certificates into PKCS#12 file, use `pvk2pfx` as in generating self-signed part. There is just option to add 1 certificate and 1 private key into a file.

Chapter 9

Windows PowerShell PKI Module

Tested under MS Windows 10 version 14393.447

9.1 General

It should be noted that to be able to write into internal certificate storage, the PowerShell needs to be run as Administrator.

9.1.1 Type

Windows PowerShell PKI Module works with Windows internal certificate storage but is able to export certificates in PKCS#12 and DER formats.

9.1.2 View and information

To view information about certificates use `Get-ChildItem -Path <path_to_cert>` command where `<path_to_cert>` usually begins with `Cert:\` option.

This commands however show only thumbprint of certificate and a `dns.name`.

9.1.3 License

License is Public, but working Microsoft Windows OS with PowerShell installed is needed.

9.2 Generation and signing of certificates

9.2.1 Keys, certificates and basic constraints

Generate keys

Self-signed certificate To generate self-signed certificate use

```
New-SelfSignedCertificate [-Provider <String>] -Path <storage_path> -DnsName "<cert_name>"
```

Where `<cert_name>` is used as Common Name for certificate.

9.2.2 Specifications

Setting Subject Alternative Name for end certificates

If `-DnsName <cert_name>` is used as list of strings *("dns1","dns2","dns3",...)* where `dns2` and latter are used as a Subject Alternative Name.

Support for Cryptographic Service Provider

Change of Cryptographic Service Provider is made by using `-Provider <String>` option.

9.3 Conversions

9.3.1 Exporting

Certificate or certificate chain from a file

Exporting of certificates is done by sequence:

```
$mypwd = Convert-To-SecureString -String "<password>" -Force -AsPlainText
Get-ChildItem -Path <path> | Export-PfxCertificate -FilePath <where_to_store> \
              [-ChainOption EndEntityCertOnly] [-Noproperties] -Password $mypwd
```

Where:

- `<password>` will be password to created PKCS#12 file as it is needed to export certificates and also needs to be of SecureString type.
- `<path>` can be:
  1. Path to certificate - only a specified certificate will be exported
  2. Directory containing certificates *e.g. Cert:\LocalMachine\My* - It will print all certificates into a single file
- `<where_to_store>` is location where the file will be saved.
- `[-ChainOption EndEntityCertOnly]` if used only End certificates will be outputed and certificate chain will be ommited.
- `[-Noproperties]` no additional properties will be printed out.

9.3.2 Importing certificates and keys into storage files

It is possible in a simmilar way as exporting a certificate into PKCS#12 file. Only difference is that all certificates that should be imported into a single folder and then `<path>` should be set to that directory.

Part IV

Conclusion

Chapter 10

Tools Review

The main thing to understand is, that every tool has had its strong and weak points. Every should be used in complience with required properties of use.

That can mean restrictions on used Operating System *(e.g. customer has only access to Apple computers with OS X operating system)*, or for example restrictions how it should be used on Operating System *(command line vs GUI program)*. In the next part i will try to go over strengths and weaknesses of each tool that I have tested.

10.1 OpenSSL

It is a great command line tool, can work with PEM, DER and PKCS#12 files. Strong point is a easy form of creating a self-signed certificate, however using it for CA certificates with ca option is not ideal - it is noted in man pages and also a O'REILLY OpenSSL book.

On the other hand strong point of OpenSSL is configuration of created certificate, which can be done in file. I tested it with creation of appropriate directories and configs.

10.2 Keytool

Keytool is the only command line tool using Java Keystore to store keys and certificates, which i consider it main strong point. By default the CSP is `java.security.provider` which allows RSA and DSA options and SHA-1,MD5,SHA-256 and SHA-512 hash algorithms. This changes when choosing other Cryptographic Service Provider. Working with it is quite easy and support for exporting as PKCS#12 is usefull addition.

10.3 GnuPG

Tool that can only make end certificates for authentication, CSR is PEM encoded and does not hold generated private key inside. Unreliable as it has been tested on 5 different Operating Systems and worked only on clean install of Windows 8.1.

10.4 Keystore Explorer

A great GUI tool for when you need to work with different types of Java Keystore files. For example, Keystore Explorer supports Bouncy Castle implementation. Even though CSP support is not implicit, there are different choices already implemented. Those can be viewed under `Help->Security Providers` tab.

Main downfall is that keys of length greater than 4096 take longer to generate *(notably RSA of length 16192, it took almost 2 hours to generate).*

10.5 XCA

XCA is a great GUI tool that can work with PEM, DER and PKCS#12 type of files. Main advantage is simplicity and amount of options, basic options are readily availiable as a button. There are many exporting options, for example PEM encoded certificate file with PKCS#8 encrypted key file. Although there is no CSP support, its ease of use makes it a great testing tool.

10.6 GnuTLS

Command line tool, availiable only under Linux. Very friendly to use, many different options in configuration file. Great support for different key and hash algorithm types. Supports key lengths of higher values *(greater than 4096 RSA for example).* Its weaknesses are no support for CSP and availibility only under UNIX like Operating Systems.

10.7 Makecert

Command line tool, not to be used in production code since microsoft has deprecated it. Very unfriendly to use and only way to use it is to download a whole Windows SDK and use it with Visual Studio console. Overall it shouldn't be used.

10.8 Windows PowerShell PKI Module

PowerShell Module on Windows Operating Systems, much easier to use and generate new certificates. Downside is, that certificates made by it are RSAwithSHA256 2048bit and that can only be changed with change of Cryptographic Service Provider. It has replaced deprecated Makecert command line tool.

Chapter 11

Choosing the right tool

The crucial part of deciding which tool to use is to know what is desired. The first choice to make is to decide what file type will be required. In this list there are most useful tools for each type:

- **PKCS#12**
  1. *GUI* - XCA
  2. *Command Line* - OpenSSL
- **Java KeyStore**
  1. *GUI* - Keystore Explorer
  2. *Command Line* - keytool

Other tested tools can be also used, but *OpenSSL, Keytool, XCA* and *Keystore Explorer* can do almost everything, along with their ease of use. My personal favourite has become *XCA* as it is a tool where I confirmed if information about certificates from other tools was as expected.

But only because they can do everything doesn't mean they are best at everything, for example if self-signed certificate is needed only for testing purposes, one can be really quickly made with the help of Windows PowerShell PKI Module where no additional instalations are required.

Considering Java KeyStore tools, only 2 were tested, 1 GUI and 1 Command Line. Nevertheless, even though they have their own upsides and downsides, are easy and quickly ready to use.