VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Samuel Svahn

IMPLEMENTATION OF DIGITAL SIGNATURE USING SIGNSERVER

Tekniikka
2021

VAASAN AMMATTIKORKEAKOULU

Tietotekniikka

TIIVISTELMÄ

Tekijä	Samuel Svahn
Opinnäytetyön nimi	Implementation of Digital Signature Using SignServer
Vuosi	2021
Kieli	englanti
Sivumäärä	40 + 5 liitettä
Ohjaaja	Anna-Kaisa Saari

Tämä opinnäytetyö tutki kuinka digitaalinen allekirjoitus toimii ja miten viitekehys asennetaan. Joidenkin palveluntarjoajien käyttöehdot mahdollistavat oikeuden tehdä muutoksia ilman tiedotuksia asiakkaille. Tästä syystä on joskus hyödyllistä olla riippumaton kolmansista osapuolista tärkeissä toiminnoissa, kuten allekirjoituksissa.

Sähköisen allekirjoituksen tulee olla turvallinen, joten sen taustalla on julkisen avaimen salaus tämän takaamiseksi. Julkisen avaimen salaus on osa julkisen avaimen infrastruktuuria, joka hallinnoin julkisen avaimen sertifikaatteja. Tutkismumateriaali koostuu pääasiallisesti työssä käytettyjen palveluntarjoajien verkkolähteistä.

Järjestelmän asennuksen ja manuaalisen testauksen jälkeen, SignServer vaikutti toimivan täysin verrokkina olleen kaupallisen palvelun tavalla.

Avainsanat Digitaalinen allekirjoitus, SignServer, Kryptografia

VAASAN AMMATTIKORKEAKOULU
VAASA UNIVERSITY OF APPLIED SCIENCES
Tietotekniikka

ABSTRACT

Author	Samuel Svahn
Title	Implementation of Digital Signature Using SignServer
Year	2021
Language	english
Pages	40 + 5 appendices
Name of Supervisor	Anna-Kaisa Saari

This thesis examines how a digital signature functions and how the framework is set up, as it is sometimes beneficial to not have to depend on third parties for vital business operations. Some third party service providers have the option to take their service offline or to change it with no prior notice, according to their terms of service.

The digital signature needs to be secure, so it is supported by a public key encryption to ensure this. A public key encryption is a part of a public key infrastructure which manages public key certificates. The research material consists mostly of web pages of the service providers used in the implementation.

After implementing a proof of concept digital signature framework and doing manual testing, it seems to function exactly like another commercial grade digital signature provider that it was tested against.

Keywords Digital Signature, SignServer, Cryptography

Table of Contents

LIST OF ABBREVIATIONS

PKI	Public Key Infrastructure
TLS	Transport Layer Security
RA	Root Authority
CA	Certificate Authority
CSR	Certificate Signing Request
PDF	Portable Document Format
TSP	Trusted Service Provider
AATL	Adobe Approved Trust List
HSM	Hardware Security Module
SHA	Secure Hashing Algorithm
PKCS	Public Key Cryptography Standards
IETF	Internet Engineering Task Force
SSL	Secure Sockets Layer
ISO	International Organization for Standardization
HTTP	Hypertext Transfer Protocol
SOAP	Simple Object Access Protocol
TSP	Trusted Service Provider
HTML	Hypertext Markup Language
LGPL	GNU Lesser General Public License

CLI	Command Line Interface
WS	Web Services
SOAP	Simple Object Access Protocol
PKCS	Public Key Cryptography Standards

LIST OF FIGURES

LIST OF APPENDICES

1 INTRODUCTION

1.1 Today’s Climate

In todays world, remote work has become more of an accepted way of working, even more so during the Covid-19 pandemic. The digital signature has become an important tool for many businesses. Digital signature is important for many trade based professions where signatures need to be done quickly, for example during a phone call with a client. The digital signature saves also time and resources.

1.2 Aims of the Project

The aim of the project was to implement a proof-of-concept level remote server that can sign PDF documents. This way companies do not have to rely on third party services (and their privacy policy and terms of service), and can customize their own server accordingly. The company suggested SignServer as the solution to start researching. After determining SignServer was suitable for the job, SignServer needed to be configured and supporting applications set up for the pipeline to deliver a signed PDF document.

Since the aim of the project is not directly cryptographic or to implement a commercial grade product, the public key infrastructure here is very simple. Although public key encryption is a vital part of digital signatures, it is far too complicated to implement completely with the resources available.

1.3 Signing Process

The signing process starts when client requests a document to be signed. This request starts a process where a worker inside SignServer receives a signing request. To get a document signed by SignServer, there needs to be a PKI, Public Key Infrastructure in place. A PKI uses public and private keys to encrypt and decrypt data respectively. These keys are used to request a certificate from the Certificate Authority (CA). A CA is the entity that is in charge of issuing certificates. A CA returns a certificate to prove that public key ownership is valid. Other entities can read the public certificate and see that it is vouched by the CA.

An example of a real-life PKI is the government (a CA). The government issues passports, drivers licences and other certificates for individuals, clients. These documents are trusted by virtually anyone and anywhere. They are bound to something that the owner of the document is/has such as a picture, social security number or DNA (a private key).

SignServer needs both public and private keys to complete the cryptographic operations on the document. These keys in turn are a part of the PKI and are bound to the certificates that a CA has provided to the client and to the workers.

2 DIGITAL SIGNATURE AND PORTABLE DOCUMENT FORMAT

2.1 Digital Signature

A digital signature is a way for someone to vouch for their identity. Only it can also be used for services, files and such. This project employs a certificate based digital signature.

Adobe Inc. defines a certificate-based digital signature as follows:

> ”A certificate-based digital signature is a type of e-signature that complies with the strictest legal regulations — and provides the highest level of assurance of a signer’s identity.” /1/

There are multiple benefits to the digital signature according to Adobe Inc.:

- Signatures are issued by trusted providers, so that certain regulations are followed.

- The digital signature and the PDF document are tamper proof.

- The signed document and the digital signature can both be revalidated for more than 10 years. /1/

2.2 Portable Document Format

Adobe Inc. developed the Portable Document Format (PDF) in 1993, it is defined in ISO 32000 /2, vii/. PDF documents are still widely used in digital information exchange.

Adobe Inc. hosts the Adobe Approved Trust List (AATL), which lists Certificate Authorities that are trusted whenever a signed PDF document is opened in Acrobat 9 or Reader 9 and later. These entities are verified by Adobe, and are held to high technical standards /3/. For example, Telia is the only Finnish company on the AATL, maybe these technical standards are expensive to implement /4/. The AATL is important because Adobe provides one of the most popular free PDF viewers, Adobe Reader. Thus it is important to have a signature that complies with Adobes trusted providers. This affects the trust of clients and the image of a company.

3 PUBLIC KEY INFRASTRUCTURE

It might be hard to see how the public key infrastructure (PKI) functions in the project, but without it, none of this would be possible. After the PKI has been set up it is quite passive and acts in the background. The PKI secures most of the information exchange over insecure networks, such as the internet. For example, HTTPS (or HTTP over SSL/TLS) is based on the PKI and almost all websites are using HTTPS today.

3.1 How a PKI Functions

The PKI is a concept that encapsulates a set of rules, policies, software and hardware that are needed to manage digital certificates and public-key cryptography /5/. In short, public key cryptography uses a key pair with a mathematical relationship to perform cryptographic operations on information. The key pair consist of a private key that is kept secret, and a public key which is a certificate, that are then used to encrypt and decrypt information. /6/

A PKI needs an entity to confirm that a public key matches with the private key of the entity distributing it, so that identities cannot be faked. To ensure the security of the system, there needs to be a trusted third party that provides services to authenticate identities of individual entities, a Certificate Authority. A CA can for example provide a certificate to an individual which is mathematically bound to their private key. /6/

3.2 Uses of PKI

PKI is used to support digitally identifying entities, it provides the cryptographic framework for creating and checking for identifying information that services can use. PKIs uses are virtually endless, ranging from signing PDF documents and transactions to securing communications. /5/

3.3 Certificates

Certificates bind a public key to a private key.

> ”In general, a public-key certificate binds a public key held by an entity (such as person, organization, account, device, or site) to a set of information that identifies the entity associated with use of the corresponding private key.” /7/

If the signature is valid and the examiner of the certificate trusts the party that has issued the certificate, the key can be used for secure communication.

3.3.1 Certificate Authority

Certificate Authority acts as the root of trust in a public key infrastructure and provides services that authenticate the identity of individuals, computers, and other entities in a network. It also issues certificates against CSRs. /6/

3.3.2 Certificate Signing Request

Signing request sent from a client application to the CA, and is filled with data about the certificate thats needed to apply for a certificate. The CAs can determine a unique set of attributes that is needed for a certain CA to accepts the CSR. /37/

> “A certification request consists of a distinguished name, a public key, and optionally a set of attributes, collectively signed by the entity requesting certification. Certification requests are sent to a certification authority, which transforms the request into an X.509 public-key certificate.” /37/

3.3.3 X.509

Most public key infrastructures use a standardized machine-readable certificate format for the certificate documents. The standard is called X.509v3. Originally, it was an ISO standard, but today it is maintained by the Internet Engineering Task Force as RFC 3280. /8/

3.4 PKCS 11 & 12

> ”PKCS #12 v1.1 describes a transfer syntax for personal identity information, including private keys, certificates, miscellaneous secrets, and extensions. Machines, applications, browsers, Internet kiosks, and so on, that support this standard will allow a user to import, export, and exercise a single set of personal identity information.” /11/

RFC-7292 gives a good description of the Public Key Cryptography Standards, or PKCS format, even though the status of this RFC is ”Informational” and not an actual standard. /11/

3.5 OpenSSL

> “OpenSSL is a robust, commercial-grade, and full-featured toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. It is also a general-purpose cryptography library.” /9/

OpenSSL was used as a cryptographic toolkit to generate keys, certificates, CSRs and PKCS#12 packages.

3.6 Hardware Security Module

A hardware security module is a tamper-proof piece of hardware that is recommended (arguably necessary) in PKI implementations for storing and protecting cryptographic keys. /10, p. 33/ Even though it is not employed in this project, Primekey advises on its use. The HSM is such an important piece for the security in a PKI that it needs to be mentioned. The only reason it was not implemented here is the relatively high price.

4 COMPONENTS OF THE SYSTEM

This chapter gives an overview of the services and applications used in the system.

In Figure 1, the general architecture of the system is laid out. Wildfly provides a platform for SignServer to run in. The PKI files are stored inside MariaDB and acccessed by SignServer when needed.

Although recommended this project does not use an HSM, rather this project uses software keystores located on the server /12/.

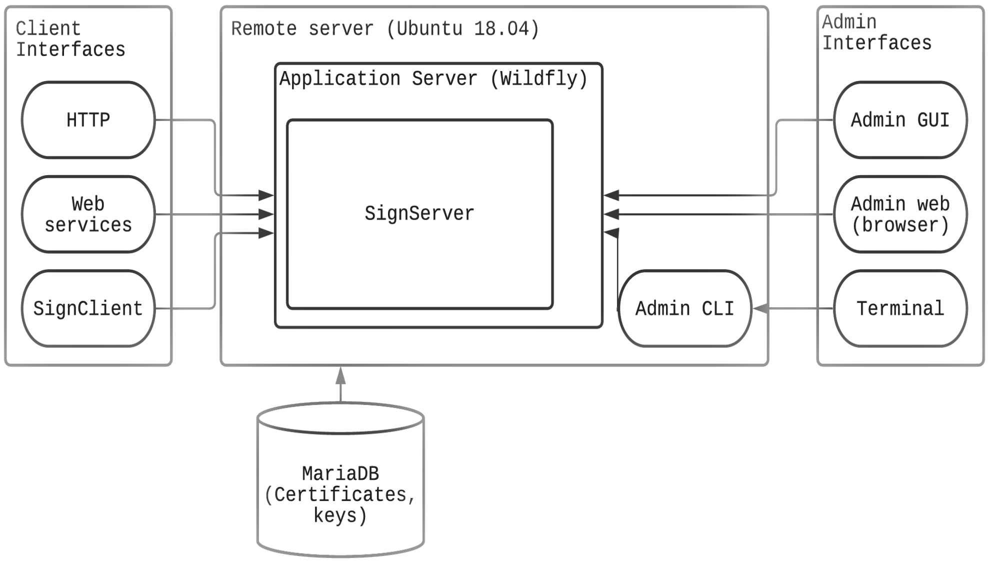


Figure 1. The achitecture of the implemented system.

4.1 Wildfly 20.0.1

Wildfly is an open-source modular application server. It is developed as a community project under the LGPL 2.1 license /14, 15/. An application server is different from a traditional web-server that delivers static content, i.e. pictures, HTML -pages etc. The primary purpose of an application is to provide functionality of the applications that run on it, to the client.

4.2 MariaDB 10.5.8

This project uses a MariaDB database on the server. The database provider was used because Primekey recommends this in use with SignServer. Other database options are also compatible.

4.3 Ubuntu 18.04 LTS

An Ubuntu 18.04 LTS remote server was rented from OVH hosting to host the signing application. It was used remotely through SSH and a command line interface. Ubuntu was chosen because it is one of the most popular, widely used Debian based Linux distributions.

5 SIGNSERVER

5.1 SignServer 5.2.0

> ”The SignServer is an application framework performing cryptographic operations for other applications.” /16/

SignServer is the core of the project. SignServer is a server-side digital signature software based on the PKI. It carries out the cryptographic operations and signs the document with provided keys. In essence this project is infrastructure built around SignServer. /16/

Figure 2 shows a detailed look into the architecture of SignServer. The client communicates with SignServer through one of the several interfaces. The authorization shows some of the options there are to authorize the use of workers. This authorization is applied independently for every worker. The signing column shows some signing options, I.e authenticode which signs Windows executables and shared libraries. The PDFSigner used on this project is not shown here, but it would go into the signing column. Signing then accesses the cryptographic keys from the keystore through the Crypto Token, held by a Crypto worker.

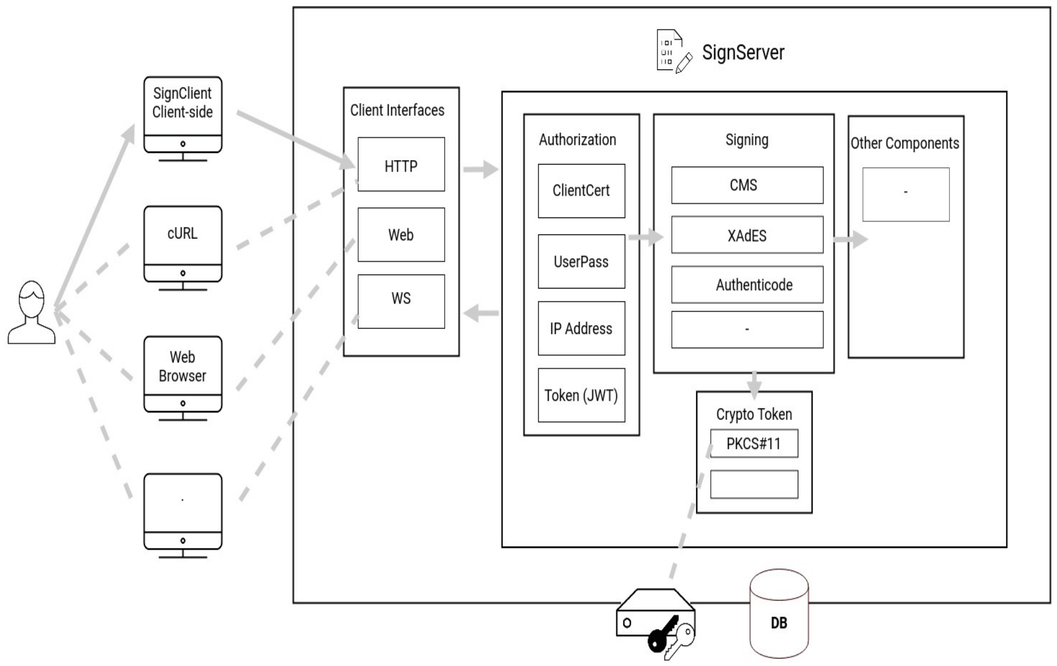


Figure 2. A client communicating with SignServer and the architecture inside SignServer /17/. The database and keystore are also a part of the PKI.

5.2 Features

Since version 3.0 there are three distinct services.

- Signers

- Validation Service

- Timed Service

Signers sign documents and are the only service utilized in this project. Validation Service is used for integrating PKIs into existing applications i.e. certificate revokation. Timed Service provides plug-ins that run at defined intervals. /16/ SignServer also features code signing and user authentication to access workers.

5.3 Interfaces

There are a number of ways of communicating with SignServer. Provided interfaces are:

- Client CLI

- Client HTTP

- Client Web Services (SOAP, that uses a special binary encoding format for requests and responses)

- Apache HTTP Server as reverse proxy

- Admin Web Service

- Administration web services can be used for remote administration through client certificate authenticated HTTPS. /16, 18/

Primarily all of these interfaces provide the same functionality, they are just different channels for transmitting the same signals. The reason there are multiple interfaces is to provide flexible integration options. A client might want to use the HTTP interface through their browser for graphical user interface.

During this project, the admin CLI was mostly used and on the client side, HTTP was used. The CLI is of course the natural interface of remote servers that are accessed through an SSH connection where there is no graphical user interface.

5.4 Workers

Workers are a part of SignServers design, they are modular and independently configured service providers. They can be thought as independent subsystems under SignServer. They are a central part of SignServers functionality. They make SignServer flexible for different environments and configurations as they are configured individually.

Workers can perform certain operations, such as signing a document. This project employs two workers, a PDF signer and a Crypto worker. A Crypto worker does not perform any operations, but holds a Crypto token that can be referenced by other workers to access cryptographic keys and operations that the underlying PKI makes possible, and use it for i.e. signing.

6 IMPLEMENTATION

6.1 Requirements

Requirements for deploying SignServer were following:

- OpenJDK 8 recommended (other Java options include more configuration).
- JBoss EAP or Wildfly application server.
- Multiple database options or NoDB. Apache Ant or Maven for deployment.
- Optional build tool if building from source.
- Compatible versions for these components are provided in the Primekey documentation /19/. Note that some commands might need elevated privileges /20/.

6.2 Installation

A new user, jboss was created to run the Wildfly and to configure SignServer. This is because it is generally safer not to run servers with root privileges. For example, when using the sudo command (which elevates privileges for a single command) the command and arguments are recorded and an audit trail is created /20/. Command 1 below, also specifies the home directory, the default shell and which group the users belongs to.

```
addgroup jboss
adduser --home /home/jboss --shell /bin/bash --ingroup jboss jboss
```

Command 1.

SignServer rights were granted to Jboss using chown and specifying the directory of SignServer recursively, so that Jboss can actually run the service. The asterisk (‘*’) after SignServer is a wildcard character and specifies all files and subdirectories under SignServer. The ‘-R’ flag will recursively go through all aforementioned files and directories applying the ownership to them (Command 2). /21/

```
chown jboss:jboss /opt/signserver* -R
```

Command 2.

6.2.1 OpenJDK8 & Apache Ant

Java and Apache Ant were installed using apt package manager. This command also unzips and installs both of the files. /22, 23/

```
apt-get install unzip openjdk-8-jdk ant
```

Command 3.

6.2.2 SignServer

SignServer was downloaded using wget network downloader. The file was then unzipped into the operating (/opt/) directory using the -d flag. A link between the directories so that it is easier to refer to the directory (Command 4).

```
wget
https://sourceforge.net/projects/signserver/files/signserver/5.2/
signserver-ce-5.2.0.Final-bin.zip

unzip signserver-ce-5.2.0.Final-bin.zip -d /opt

ln -s /opt/signserver-ce-5.2.0.Final-bin.zip /opt/signserver

chown jboss:jboss /opt/signserver* -R
```

Command 4.

6.2.3 MariaDB database

The newest stable release of MariaDB was downloaded using wget network downloader and updated and installed (Command 5).

```
wget
https://downloads.mariadb.com/MariaDB/mariadb-10.5.8/repo/ubuntu/m
ariadb-10.5.8-ubuntu-bionic-amd64-debs.tar

sudo apt update
sudo apt install mariadb-server
```

Command 5.

The database was created according to the database setup instructions in command 6. This setup is used for MariaDB. /24/

```
mysql –u root –p mysql

create database signserver;
grant all on signserver.* to "signserver"@"localhost" identified
by "signserver";
quit
```

Command 6.

Command 7 shows how the Java 8 connector was downloaded from MariaDB resource and placed inside Wildflys directories as mariadb. The connector was moved to the main directory.

```
wget https://downloads.mariadb.com/Connectors/java/connector-java-
2.6.2/mariadb-java-client-2.6.2.jar

mkdir -p /opt/wildfly/modules/system/layers/base/org/mariadb/main/

mv
mariadb-java-client-2.6.2.jar/opt/wildfly/modules/system/layers/ba
se/org/mariadb/main/
```

Command 7.

Configuration for the Wildfly module was created in module.xml. This configuration file specifies the database provider and the path of the database connector (Command 8).

```
nano
/opt/wildfly/modules/system/layers/base/org/mariadb/main/module.xm
l

<?xml version="1.0" encoding="UTF-8"?>

<module xmlns="urn:jboss:module:1.0" name="org.mariadb">

  <resources>

      <resource-root path="mariadb-java-client-2.6.2.jar"/>

  </resources>

  <dependencies>

      <module name="javax.api"/>

      <module name="javax.transaction.api"/>

  </dependencies>

</module>
```

Command 8.

6.2.4 Wildfly Application Server

Using wget the Wildfly package was downloaded from their website /25/. The package was extracted with 'xf' and filtered tar.gz through Gzip with the 'z' flag (Command 9). /26/

```
wget https://download.jboss.org/wildfly/20.0.1.Final/wildfly-
20.0.1.Final.tar.gz

tar xfz wildfly-20.0.1.Final.tar.gz -C /opt

ln -s /opt/wildfly-20.0.1.Final /opt/wildfly

chown jboss:jboss /opt/wildfly* -R
```

Command 9.

SignServer needs to access Wildlflys location, to enable this the location of Wildfly was added as an environment variable to the end of ".bashrc" (Command 10). According to the installation guide, it is not mandatory to set NodeID, but errors will get printed in the log if not set. NodeID is not used as there is only one node (one instance of SignServer running in the network/server) /27/. A large corporation could utilize multiple nodes to provide more capacity, or to provide specific services.

```
nano /home/jboss/.bashrc

export APPSRV_HOME=opt/wildfly-20.0.1.Final
export SIGNSERVER_NODEID=node1
```

Command 10.

6.2.5 Start Wildfly and Configuring MariaDB

A new instance of the SSH terminal was opened and Wildfly was started as user Jboss by running standalone.sh (Command 11). Then another terminal was opened for configuring Wildfly by using the Jboss CLI (Command 12).

```
cd /opt/wildfly/wildfly-20.0.1.Final$
./bin/standalone.sh

cd /opt/wildfly/wildfly-20.0.1.Final$
./bin/jboss-cli.sh
```

Command 11.

```
You are disconnected at the moment. Type 'connect' to connect to
the server or 'help' for the list of supported commands.

[disconnected /]  connect
```

Command 12.

MariaDB connector was registered in the Wildfly CLI. The CLI returns outcome and result parameters for every command (Command 13).

```
/subsystem=datasources/jdbc-
driver=org.mariadb.jdbc.Driver:add(driver-
name=org.mariadb.jdbc.Driver,driver-module-
name=org.mariadb,driver-xa-datasource-class-
name=org.mariadb.jdbc.MySQLDataSource)

[standalone@localhost:9990 /] :reload
{
"outcome" => "success",
"result" => undefined
}
```

Command 13.

The database was configured further in command 14 according to the instructions by Primekey.

```
data-source add --name=signserverds --driver-
name="org.mariadb.jdbc.Driver"
--connection-url="jdbc:mysql://127.0.0.1:3306/signserver" --jndi-
name="java:/SignServerDS" --use-ccm=true --driver-
class="org.mariadb.jdbc.Driver" --user-name="yourUsername" --
password="yourPassword" --validate-on-match=true --background-
validation=false --prepared-statements-cache-size=50 --share-
prepared-statements=true --min-pool-size=5 --max-pool-size=150 --
pool-prefill=true --transaction-
isolation=TRANSACTION_READ_COMMITTED --check-valid-connection-
sql="select 1;"
```

Command 14.

6.3 PKI

Setting up the PKI, the following files were created. Command 15 shows the basic setup for the general directories for the PKI. /28/

```
mkdir /opt/pki
mkdir -p /opt/pki/db/ca.db.certs
mkdir /opt/pki/config
mkdir /opt/pki/certificates
mkdir /opt/pki/requests
echo '01' > /opt/pki/db/ca.db.serial
touch /opt/pki/db/ca.db.index
```

Command 15.

Contents of the CAs config file at /opt/pki/config/ca.config outlines the rules and information for our CA to be used by the CA itself and others interacting with it (Appendix 1).

Private keys need to be generated for every operator in the PKI. Using OpenSSL genrsa command in command 15 /29/. Command arguments define the cipher algorithm DES3, which is a cipher run three times in a row to make the encryption stronger and an output file. The last option specified is the length of the private key generated as bits. In production projects, Triple DES algorithm should not be used as it is considered a weak cipher /30/. For the private key encryption in this case, it is enough to have the process done correctly, the ciphers can always be changed according to specified needs. After executing the command, the prompt will ask to enter a passphrase to decrypt the key.

```
openssl genrsa -des3 -out /opt/pki/certificates/ca.key 2048
```

Command 16.

The req command creates a PCKS#10 certificate signing request. The certificate uses UTF8 encoding. The ‘new’ parameter creates a new certificate and will prompt to fill out the certificate fields as defined in the configured CA policy (file: ca.config). X509 parameter will create a certificate instead of a request.

Days specifies how long the certificate is valid. Finally the CA key location and output file are specified (Command 17). /31/

After executing a prompt will appear and ask for CA private key passphrase specified after the previous command. The necessary information to create the certificate were provided in the prompt.

```
openssl req -utf8 -new -x509 -days 3000 -key
/opt/pki/certificates/ca.key -out /opt/pki/certificates/ca.crt
```

Command 17.

The x509 command can do multiple things such as signing and view contents of a certificate (Command 18). This command has the outform DER argument which specifies the encoding of the certificate to a DER format. DER is a binary certificate format so it does not contain any BEGIN/END statements as other formats such as PEM that are ASCII based. DER is usually used in Java platforms. /32, 33/

```
openssl x509 -in /opt/pki/certificates/ca.crt -outform DER -
out /opt/pki/certificates/ca.der
```

Command 18.

6.3.1 Wildfly Key and Certificate for SSL

The function of an SSL certificate is to provide functional HTTPS communication with an endpoint. The certificate makes SSL/TLS encryption possible. A new key is generated with command 19. A certificate request was configured, creating the file at /opt/pki/requests/https.wildfly.req (Appendix 2). A file was used to create this request, because there needs to be a ‘subjectAltName’ property, otherwise some browsers might not accept the certificate.

```
openssl genrsa -out /opt/pki/certificates/https.wildfly.key 2048
```

Command 19.

A certificate request is generated for Wildfly from the CA. The ‘req’ command here (Command 20) is used without the x509 parameter, so a CSR is created instead of the case above where a certificate was created.

```
openssl req -config requests/https.wildfly.req -days 365 -new key
/opt/pki/certificates/https.wildfly.key -out
/opt/pki/certificates/https.wildfly.csr
```

Command 20.

The CA was requested to sign a certificate according to the CSR (Command 21) /34/. A prompt will ask to review the certificate. The certificate was signed and committed to the database (Attachment 3).

```
openssl ca -config /opt/pki/config/ca.config -out
/opt/pki/certificates/https.wildfly.crt -infiles
/opt/pki/certificates/https.wildfly.csr
```

Command 21.

The PKCS#12 command was used to export the certificate and key together in a bundle format (Command 22). The prompt will then ask to enter a passphrase.

```
openssl pkcs12 -export -in
/opt/pki/certificates/https.wildfly.crt -inkey
/opt/pki/certificates/https.wildfly.key -out
/opt/pki/certificates/wildfly.p12 -name wildfly -chain -CAfile
/opt/pki/certificates/ca.crt
```

Command 22.

6.3.2 Generic Crypto Token

Using the same commands as before to configure the Crypto Worker. ‘my Organization’ and ‘CryptoGeneric’ were used as an organization name and a common name respectively (Command 23).

```
openssl genrsa -out /opt/pki/certificates/crypto.generic.key 1024

openssl req -days 365 -new -key
/opt/pki/certificates/crypto.generic.key -out
/opt/pki/certificates/crypto.generic.csr

openssl ca -config /opt/pki/config/ca.config -out
/opt/pki/certificates/crypto.generic.crt -infiles
/opt/pki/certificates/crypto.generic.csr
openssl pkcs12 -export -in
/opt/pki/certificates/crypto.generic.crt -inkey
/opt/pki/certificates/crypto.generic.key -out
/opt/pki/certificates/crypto.generic.p12 -name generic -CAfile
/opt/pki/certificates/ca.crt
```

Command 23.

6.3.3 Configure Wildfly SSL

Using the Java keytool, the keystore was created. The deststorepass and destkeypass must be the same or Wildfly will not be able to retrieve the key (Command 24). /35/

```
mkdir /opt/wildfly/standalone/configuration/keystore

keytool -importkeystore -deststorepass *passphrase* -destkeypass
*passphrase* -destkeystore
/opt/wildfly/standalone/configuration/keystore/wildfly.keystore.j
ks -srckeystore /opt/pki/certificates/wildfly.p12 -srcstoretype
PKCS12 -srcstorepass *passphrase1* -alias wildfly

keytool -importkeystore -srckeystore
/opt/wildfly/standalone/configuration/keystore/wildfly.keystore.j
ks -destkeystore
/opt/wildfly/standalone/configuration/keystore/wildfly.keystore.j
ks -deststoretype pkcs12
```

```
keytool -alias CA_Own -import -file /opt/pki/certificates/ca.crt
-keystore
/opt/wildfly/standalone/configuration/keystore/truststore.jks


chown jboss:jboss /opt/wildfly* -R
```

Command 24.

Wildfly standalone was started again and connected to Jboss CLI to configure further. A blog by Octopus Deploy helped in the navigation of the Wildfly CLI and checking of the values after executing the commands (Command 25) /36/.

```
/socket-binding-group=standard-sockets/socket-
binding=httpspriv:add(port="8443",interface="httpspriv")

/core-service=management/security-realm=SSLRealm:add()

/core-service=management/security-realm=SSLRealm/server-
identity=ssl:add(keystore-path="keystore/wildfly.keystore.jks",
keystore-relative-to="jboss.server.config.dir", keystore-
password="passphrase", alias="wildfly")

/core-service=management/security-realm=SSLRealm/
authentication=truststore:add(keystore-path="keystore/
truststore.jks", keystore-relative-to="jboss.server.config.dir",
keystore-password="passphrase1")

/subsystem=undertow/server=default-server/https-
listener=httpspriv:add(socket-binding="httpspriv", security-
realm="SSLRealm", verify-client=REQUIRED)
```

Command 25.

6.4 SignServer Deployment

SignServer comes with some default configurations that can be copied and used as a template for unique configurations. The configuration was copied and opened for editing. The lines containing appserver home and database name were activated by uncommenting them (Command 26).

```
cp /opt/signserver/conf/signserver_deploy.properties.sample
/opt/signserver/conf/signserver_deploy.properties

nano /opt/signserver/conf/signserver_deploy.properties


appserver.home=${env.APPSRV_HOME}

database.name=mysql

chown jboss:jboss /opt/signserver* -R
```

Command 26.

To deploy SignServer and the configuration to the application server, using ant ‘deploy’ as a deploy tool. When the server is ready, the version of SignServer can be checked to see if everything went OK (Command 27).

```
cd /opt/signserver

./bin/ant deploy

.bin/signserver getstatus brief all

Current version of server is : SignServer EE 4.0.0
```

Command 27.

6.5 Workers

6.5.1 Crypto Worker

The properties file for Crypto Generic worker was created at /opt/signserver/myconfig/worker-crypto.properties (Attachment 4). After this a keystore was created and imported (Command 30). Defaultkey is the name of the generic certificate imported in the keystore (-alias generic). After this the configuration was imported into SignServer.

```
mkdir /opt/signserver/myconfig

nano /opt/signserver/myconfig/worker-crypto.properties


keytool -importkeystore -deststorepass keystorepwd  -destkeypass
keystorepwd -destkeystore
/opt/signserver-ce-5.2.0.Final/myconfig/worker.crypto.keystore.jk
s -srckeystore /opt/pki/certificates/crypto.generic.p12 -
srcstoretype PKCS12 -srcstorepass cryptobois -alias generic

keytool -importkeystore -srckeystore /opt/signserver-ce-
5.2.0.Final/myconfig/worker.crypto.keystore.jks -destkeystore
/opt/signserver-ce-5.2.0.Final/myconfig/worker.crypto.keystore.jk
s -deststoretype pkcs12
```

Command 28.

```
cd /opt/signserver

./bin/signserver setproperties /opt/signserver/myconfig/worker
```

Command 29.

6.5.2 PDF Signer

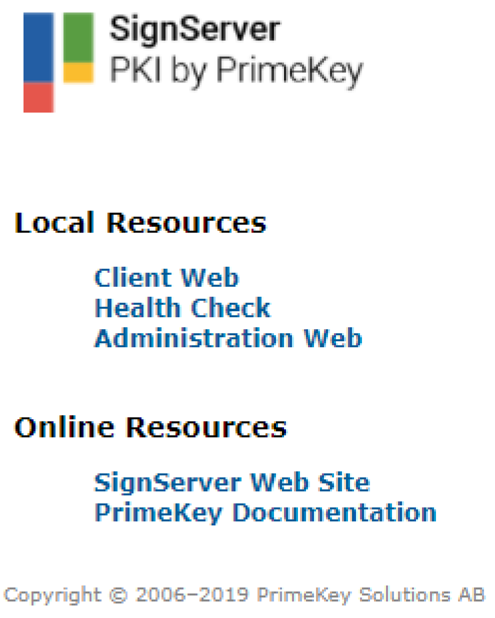
The configuration file was created similarly to the Crypto worker. Different configuration options in the config file attached (Attachment 5).

```
./bin/signserver setproperties myconfig/worker-
pdfsigner.properties
```

Command 30.

6.6 Testing

The deployed server is located at https://51.75.66.251:8442 and SignServer at https://51.75.66.251:8442/signserver/. From the landing page (Figure 3), there are some Administrator options, but the Signing is done in the Client web. To access the administration web, a client certificate is required. The default pages are just basic HTML pages and can be modified and replaced as necessary.


SignServer
PKI by PrimeKey

Local Resources

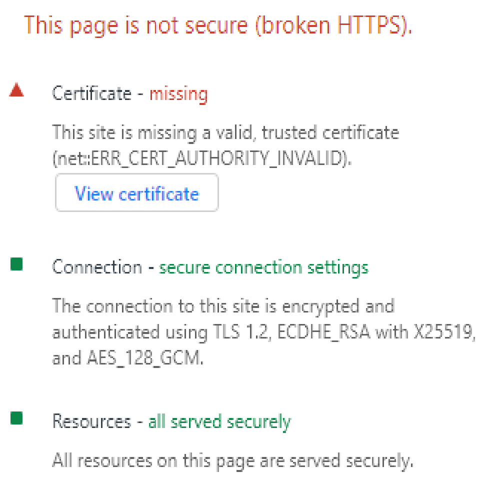
Client Web
Health Check
Administration Web

Online Resources

SignServer Web Site
PrimeKey Documentation

Copyright © 2006-2019 PrimeKey Solutions AB

Figure 3. SignServers landing page.


This page is not secure (broken HTTPS).

▲ Certificate - missing

This site is missing a valid, trusted certificate (net::ERR_CERT_AUTHORITY_INVALID).

View certificate

■ Connection - secure connection settings

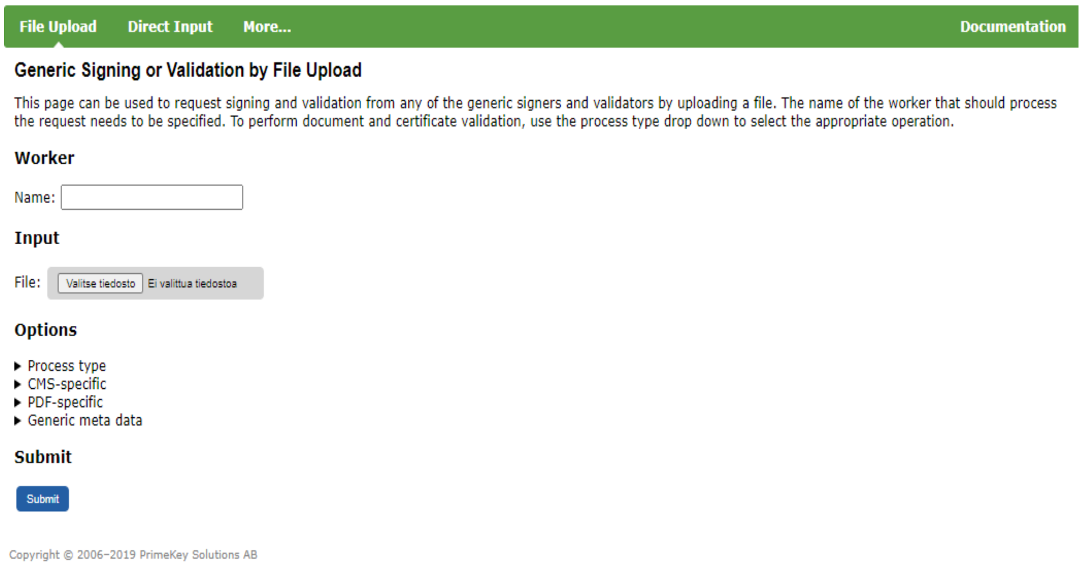
The connection to this site is encrypted and authenticated using TLS 1.2, ECDHE_RSA with X25519, and AES_128_GCM.

■ Resources - all served securely

All resources on this page are served securely.

Figure 4. Security view of the landing page on Google Chrome.

As seen on the security tab viewed in Google Chrome (Figure 4), the CA created is not trusted by the browser. Self-signed certificates are not trusted. Implementations used for production should obtain an SSL/TLS certificate to have browsers trust the domain.



Generic Signing or Validation by File Upload

This page can be used to request signing and validation from any of the generic signers and validators by uploading a file. The name of the worker that should process the request needs to be specified. To perform document and certificate validation, use the process type drop down to select the appropriate operation.

Worker

Name:

Input

File: Valitse tiedosto Ei valittua tiedostoa

Options

- Process type
- CMS-specific
- PDF-specific
- Generic meta data

Submit

Copyright © 2006-2019 PrimeKey Solutions AB

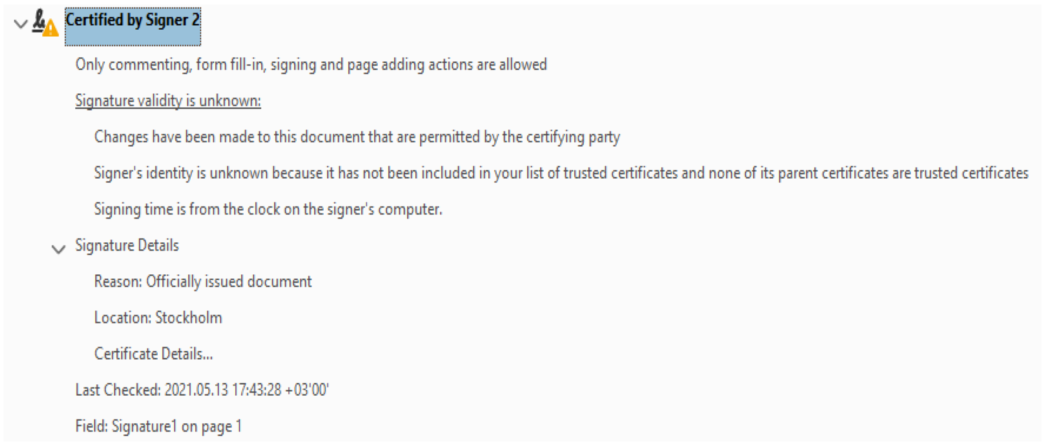
Figure 5. Signing page.

Steps to sign a document:

- Input the name of the worker you want to use
- Select the input file from your computer
- Modify the options if necessary I.e the PDF is password locked
- SignServer sends the signed document to you and it is automatically downloaded, presuming there were no errors in the process.

SignServer provides different kinds of PDF documents to use for testing. Here in testing a file called “sample-certified-signingallowed256.pdf” was used. it is a PDF with Primekey logo and a signature done by Primekey. It can only be signed, form filled, commented or have pages added. After signing it is guaranteed to be tamper proof with its hash. As our CA is not on the AATL, the signatures origin is not trusted, as we see on the Primekeys signature too (Figure 6).

The tests also covered password locked and editing locked documents. In the case of errors, SignServer returned a HTML page with a simple text and a short description of the error.



Certified by Signer 2

Only commenting, form fill-in, signing and page adding actions are allowed

Signature validity is unknown:

Changes have been made to this document that are permitted by the certifying party

Signer's identity is unknown because it has not been included in your list of trusted certificates and none of its parent certificates are trusted certificates

Signing time is from the clock on the signer's computer.

Signature Details

Reason: Officially issued document

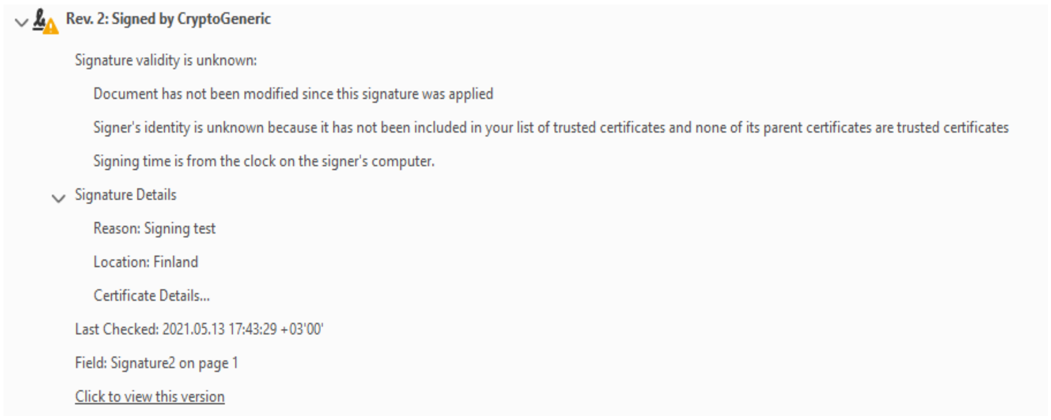
Location: Stockholm

Certificate Details...

Last Checked: 2021.05.13 17:43:28 +03'00'

Field: Signature1 on page 1

Figure 6. The PDF before signing.



Rev. 2: Signed by CryptoGeneric

Signature validity is unknown:

Document has not been modified since this signature was applied

Signer's identity is unknown because it has not been included in your list of trusted certificates and none of its parent certificates are trusted certificates

Signing time is from the clock on the signer's computer.

Signature Details

Reason: Signing test

Location: Finland

Certificate Details...

Last Checked: 2021.05.13 17:43:29 +03'00'

Field: Signature2 on page 1

Click to view this version

Figure 7. After the signing is done a new revision is added, containing the signature information.

Looking at the signature details on Figure 7, the 'reason' field is set as 'Signing test.' As observed during the testing of other signing services, the 'reason' field is used to record the name of the signer.

7 CONCLUSIONS

This thesis was a proof of concept on how to implement SignServer, a simple PKI and the supporting services on a remote server. Since the safety of the implementation cannot be assured, it is not meant to be used in production as is. The mechanics are functional, but a PKI is much more complicated than this. There is no revocation or renewal of certificates, the keys are stored on the server rather than the recommended HSM. HTTPS is not implemented because the CA is self-signed with no root certificate. A proper solution would be to buy an SSL/TLS certificate from a trusted provider, so that users could trust the system.

Manual tests were done against a commercial signature provider and SignServer seems to work very similarly. SignServer could be working in the background in their systems too, so at the least SignServer can do the same as their systems.

Requesting a signature to a PDF and authorizing it through an email message yielded a PDF where the "signer" field was "Signer0002" and the "reason" field was the name in the signature. These commercial companies might run their own scripts to create a new signer, or edit the xml configuration of an existing one when a signature is requested and then just change the "reason" field to show the name of the requestor.

Some of the algorithms, hashes and ciphers are not up to todays security standards, but it is easily implemented when setting up a PKI when the needs of the system are known. Here the PKI was made using a guide /38/, but there are better options for production systems i.e EJBCA also by Primekey /39/, an open source CA solution.

In a production solution, there would probably be better Linux distributions especially made to run servers that have less overhead than Ubuntu. Implementing SignServer into production brings many questions, especially regarding security.

If there are resources to take care of the mentioned concerns, SignServer is a good solution.

REFERENCES

/1/ Adobe Inc. website. Accessed 29.3.2021
https://acrobat.adobe.com/us/en/sign/glossary/digital-signatures.html

/2/ Document management - Portable document format – Part 1: PDF 1.7, First edition 7.1.2008. Accessed 29.3.2021.
http://wwwimages.adobe.com/www.adobe.com/content/dam/acom/en/devnet/pdf/pdfs/PDF32000_2008.pdf

/3/ Adobe AATL Technical requirements Accessed 6.5.2021.
https://helpx.adobe.com/acrobat/kb/approved-trust-list2.html#AATLtechnicalrequirements

/4/ Adobe Approved trust list, Accessed 29.3.2021.
https://helpx.adobe.com/acrobat/kb/approved-trust-list2.html

/5/ Nexus Group, Public key infrastructure (PKI) explained in 4 minutes, 03/09 2018. Accessed 29.3.2021.
https://www.nexusgroup.com/crash-course-pki/

/6/ Microsoft, Public Key Infrastructure 31.5.2018. Accessed 29.3.2021
https://docs.microsoft.com/en-gb/windows/win32/seccertenroll/public-key-infrastructure?redirectedfrom=MSDN

/7/ IETF, RFC 3647 Internet X.509 Public Key Infrastructure November 2003, Accessed 29.3.2021.
https://www.ietf.org/rfc/rfc3647.txt

/8/ IETF, RFC 3280, Accessed 11.5.2021.
https://datatracker.ietf.org/doc/html/rfc3280

/9/ OpenSSL, Accessed 10.5.2021.
https://www.openssl.org/

/10/ Microsoft, Code Signing Best Practices July 25 2007, Accessed 15.4.2021.
http://www.microsoft.com/whdc/winlogo/drvsign/best_practices.mspx

/11/ IETF, RFC 7292 (PKCS#12), Accessed 6.4.2021.
https://tools.ietf.org/html/rfc7292

/12/ Primekey documentation, Accessed 6.4.2021.
https://doc.primekey.com/signserver/signserver-introduction

/13/ Oracle documentation, Java keytool, Accessed 6.4.2021.
https://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html

/14/ GNU, LGPL 2.1 license, Accessed 6.4.2021.
https://www.gnu.org/licenses/old-licenses/lgpl-2.1.en.html

/15/ Github, Wildfly source code, Accessed 6.4.2021.
https://github.com/wildfly/wildfly/

/16/ Primekey documentation, SignServer, Accessed 6.4.2021.
https://download.primekey.se/docs/SignServer-Enterprise/4.0.2/manual/introduction.html

/17/ Primekey documentation, SignServer architecture and concepts, Accessed 15.4.2021.
https://download.primekey.com/docs/SignServer-Enterprise/current/Architecture_and_Concepts.html

/18/ Primekey documentation, SignServer integration, Accessed 6.4.2021.
https://doc.primekey.com/signserver540/signserver-integration

/19/ Primekey documentation, SignServer Prequisites, Accessed 7.4.2021.
https://doc.primekey.com/signserver520/signserver-installation/prerequisites

/20/ Sudo, Sudo in a Nutshell, Accessed 11.5.2021.
https://www.sudo.ws/intro.html

/21/ Linux die, Chown reference, Accessed 11.5.2021.
https://linux.die.net/man/1/chown

/22/ Linux die, Apt-get reference, Accessed 11.5.2021.
https://linux.die.net/man/8/apt-get

/23/ Linux die, Unzip reference, Accessed 11.5.2021.
https://linux.die.net/man/1/unzip

/24/ Primekey, Database setup,Accessed 11.5.2021.
https://doc.primekey.com/signserver520/signserver-installation/database-setup

/25/ Wildfly Downloads, Accessed 13.5.2021.
https://www.wildfly.org/downloads/

/26/ Linux Die, Tar command reference, Accessed 12.5.2021.
https://linux.die.net/man/1/tar

/27/ Primekey documentation, SIGNSERVER_NODEID, Accessed 7.4.2021.
https://download.primekey.com/docs/SignServer-Enterprise/current/Install_SignServer.html

/28/ Linux Die Net, Mkdir command reference, Accessed 10.5.2021.
https://linux.die.net/man/1/rm

/29/ OpenSSL, genrsa command refence, Accessed 10.5.2021.
https://www.openssl.org/docs/man1.0.2/man1/genrsa.html

/30 OpenSSL, The SWEET32 Issue, CVE-2016-2183, Accessed 13.5.2021.
https://www.openssl.org/blog/blog/2016/08/24/sweet32/

/31/ OpenSSL, req command reference, Accessed 10.5.2021.
https://www.openssl.org/docs/manmaster/man1/openssl-req.html

/32/ OpenSSL, x509 command reference, Accessed 10.5.2021.
https://www.openssl.org/docs/man1.0.2/man1/x509.html

/33/ DigiCert, What is DER Format?, Accessed 13.5.2021.
https://knowledge.digicert.com/quovadis/ssl-certificates/ssl-general-topics/what-is-der-format.html

/34/ Linux die, CA reference, Accessed 13.5.2021.
https://linux.die.net/man/1/ca

/35/ Linux die, Java Keytool, Accessed 13.5.2021.
https://linux.die.net/man/1/keytool-java-1.7.0-openjdk

/36/ Octopus Deploy, Using the Wildfly CLI, Accessed 13.5.2021.
https://octopus.com/blog/using-the-wildfly-cli

/37/ IETF, RFC 2968, Accessed 14.5.2021.
https://datatracker.ietf.org/doc/html/rfc2986

/38/ blink38, SignServer guide, Accessed 16.5.2021.
https://github.com/blink38/signserver

/39/ Primekey, EJBCA, Accessed 14.5.2021. https://www.ejbca.org/

APPENDIX 1

```
[ ca ]
default_ca      = CA_OWN


[ CA_own ]
dir             = /opt/pki/db
certs           = /opt/pki/db
new_certs_dir   = /opt/pki/db/ca.db.certs
database        = /opt/pki/db/ca.db.index
serial          = /opt/pki/db/ca.db.serial
RANDFILE        = /opt/pki/db/ca.db.rand
certificate     = /opt/pki/certificates/ca.crt
private_key     = /opt/pki/certificates/ca.key
default_days    = 3000
default_crl_days = 30
default_md      = sha256
preserve        = no
policy          = policy_anything
copy_extensions = copy
x509_extensions = cert_ext


[ policy_anything ]
countryName             = optional
stateOrProvinceName     = optional
localityName            = optional
organizationName        = optional
organizationalUnitName  = optional
commonName              = supplied
```

```
emailAddress            = optional

[cert_ext]
```

APPENDIX 2

```
# OpenSSL configuration file for creating a CSR for a server
certificate

# Adapt at least the FQDN and ORGNAME lines, and then run

# openssl req -new -config myserver.cnf -keyout myserver.key -out
myserver.csr

# on the command line.


# the fully qualified server (or service) name

FQDN = wildfly.domain


# the name of your organization

# (see also https://www.switch.ch/pki/participants/)

ORGNAME = MyOrganization


# subjectAltName entries: to add DNS aliases to the CSR, delete

# the '#' character in the ALTNAMES line, and change the
subsequent

# 'DNS:' entries accordingly. Please note: all DNS names must

# resolve to the same IP address as the FQDN.

ALTNAMES = DNS:$FQDN   # , DNS:bar.example.org ,
DNS:www.foo.example.org


[ req ]

default_bits = 2048

default_md = sha256
```

```
prompt = no

encrypt_key = no

distinguished_name = dn

req_extensions = req_ext


[ dn ]

C = FI

O = $ORGNAME

CN = $FQDN


[ req_ext ]

subjectAltName = $ALTNAMES
```

APPENDIX 3

```
Check that the request matches the signature

Signature ok

The Subject's Distinguished Name is as follows

countryName            :PRINTABLE:'FI'

organizationName       :ASN.1 12:'MyOrganization'

commonName             :ASN.1 12:'wildfly.domain'

Certificate is to be certified until May  5 12:47:09 2029 GMT
(3000 days)

Sign the certificate? [y/n]:Y


1 out of 1 certificate requests certified, commit? [y/n]y

Write out database with 1 new entries

Data Base Updated
```

APPENDIX 4

```
# Type of worker

WORKERGENID1.TYPE=CRYPTO_WORKER



# This worker will not perform any operations on its own and
indicates this by

# using the worker type CryptoWorker

WORKERGENID1.IMPLEMENTATION_CLASS=org.signserver.server.signers.C
ryptoWorker



# Uses a soft keystore:

WORKERGENID1.CRYPTOTOKEN_IMPLEMENTATION_CLASS=org.signserver.serv
er.cryptotokens.KeystoreCryptoToken



# Name for other workers to reference this worker:

WORKERGENID1.NAME=MyCryptoTokenP12



# Type of keystore

# PKCS12 and JKS for file-based keystores

# INTERNAL to use a keystore stored in the database (tied to the
crypto worker)

WORKERGENID1.KEYSTORETYPE=PKCS12



# Path to the keystore file (only used for PKCS12 and JKS)

WORKERGENID1.KEYSTOREPATH=/opt/signserver/myconfig/
worker.crypto.keystore.jks



# Optional password of the keystore. If specified the token is
"auto-activated".

WORKERGENID1.KEYSTOREPASSWORD=keystorepwd
```

```
# Optional key to test activation with. If not specified the
first key found is

# used.

WORKERGENID1.DEFAULTKEY=generic
```

APPENDIX 5

```
## General properties

WORKERGENID1.TYPE=PROCESSABLE

WORKERGENID1.IMPLEMENTATION_CLASS=org.signserver.module.pdfsigner
.PDFSigner

WORKERGENID1.NAME=PDFSigner

WORKERGENID1.AUTHTYPE=NOAUTH


# Crypto token

WORKERGENID1.CRYPTOTOKEN=MyCryptoTokenP12

#WORKERGENID1.CRYPTOTOKEN=CryptoTokenP11


# Using key from sample keystore

WORKERGENID1.DEFAULTKEY=generic

# Key using ECDSA

#WORKERGENID1.DEFAULTKEY=signer00002



## PDFSigner properties


#--------------------------SIGNATURE
PROPERTIES---------------------------------------#
```

```
# specify reason for signing. it will be displayed in signature
properties when viewed

# default is "Signed by SignServer"

#WORKERGENID1.REASON=Signed by SignServer

WORKERGENID1.REASON=Signing test



# specify location. it will be displayed in signature properties
when viewed

# default is "SignServer"

#WORKERGENID1.LOCATION=SignServer

WORKERGENID1.LOCATION=Finland



# digest algorithm used for the message digest and signature
(this is optional and defaults to SHA1)

# the algorithm determines the minimum PDF version of the
resulting document and is documented in the manual.

# for DSA keys, only SHA1 is supported

WORKERGENID1.DIGESTALGORITHM=SHA256



#---------------------------SIGNATURE
VISIBILITY--------------------------------------#



# if we want the signature to be drawn on document page set
ADD_VISIBLE_SIGNATURE to True , else set to False

# default is "False"

#WORKERGENID1.ADD_VISIBLE_SIGNATURE = False

WORKERGENID1.ADD_VISIBLE_SIGNATURE = True



# specify the page on which the visible signature will be drawn

# this property is ignored if ADD_VISIBLE_SIGNATURE is set to
False

# default is "First"
# possible values are :

        # "First" : signature drawn on first page of the
document,

        # "Last"  : signature drawn on last page of the document,

        # page_number : signature is drawn on a page specified by
numeric argument. If specified page number exceeds page count of
the document ,signature is drawn on last page

        # if page_number specified is not numeric (or negative
number) the signature will be drawn on first page

WORKERGENID1.VISIBLE_SIGNATURE_PAGE = 2



# specify the rectangle signature is going to be drawn in

# this property is ignored if ADD_VISIBLE_SIGNATURE is set to
False

# defailt is "400,700,500,800"

# format is : (llx,lly,urx,ury). Here llx =left lower x
coordinate, lly=left lower y coordinate,urx =upper right x
coordinate, ury = upper right y coordinate

#WORKERGENID1.VISIBLE_SIGNATURE_RECTANGLE = 400,700,500,800



# if we want the visible signature to contain custom image ,
specify image as base64 encoded byte array

# alternatively custom image can be specified by giving a path to
image on file system

# note : if specifying a path to an image "\" should be escaped (
thus C:\photo.jpg => "C:\\photo.jpg" )

# note : if specifying image as base64 encoded byte array "="
should be escaped (this "BBCXMI==" => "BBCXMI\=\=")

# if both of these properties are set then
VISIBLE_SIGNATURE_CUSTOM_IMAGE_BASE64 will take priority

# if we do not want this feature then do not set these properties

# default is not set (no custom image)

# these properties are ignored if ADD_VISIBLE_SIGNATURE is set to
False

#WORKERGENID1.VISIBLE_SIGNATURE_CUSTOM_IMAGE_BASE64=
#WORKERGENID1.VISIBLE_SIGNATURE_CUSTOM_IMAGE_PATH=

# if we want our custom image to be resized to specified
rectangle (set by VISIBLE_SIGNATURE_RECTANGLE) then set to True.

# if set to True image might look different that original (as an
effect of resizing)

# if set to False the rectangle drawn will be resized to
specified image's sizes.

# if set to False llx and lly coordinates specified by
VISIBLE_SIGNATURE_RECTANGLE property will be used for drawing
rectangle (urx and ury will be calculated from specified image's
size)

# this property is ignored if ADD_VISIBLE_SIGNATURE is set to
False or if custom image to use is not specified

# default is True

#WORKERGENID1.VISIBLE_SIGNATURE_CUSTOM_IMAGE_SCALE_TO_RECTANGLE =
True



# to create a certifying signature that certifies the document
set the CERTIFICATION_LEVEL

# possible values are: NOT_CERTIFIED, FORM_FILLING,
FORM_FILLING_AND_ANNOTATIONS or NO_CHANGES_ALLOWED

# default is NOT_CERTIFIED

# WORKERGENID1.CERTIFICATION_LEVEL=NOT_CERTIFIED

#--------------------------SIGNATURE
TIMESTAMPING--------------------------------------#



# if we want to timestamp document signature, specify timestamp
authority url, if required bu tsa uncomment tsa username and
password lines and specify proper values

# if we do not want to timestamp document signature , do not set
property



# Worker ID or name of internal timestamp signer in the same
SignServer

# Default: none

#WORKERGENID1.TSA_WORKER=TimeStampSigner
```

```
# URL of external timestamp authority

# note : if path contains characters "\" or "=" , these
characters should be escaped (thus "\" = "\\", "=" =>"\=")

# default is not set (no timestamping)

# WORKERGENID1.TSA_URL =

#WORKERGENID1.TSA_URL=http://tsa.example.com:8080/signserver/tsa?
workerName\=TSA



# if tsa requires authentication for timestamping , specify
username and password

# if tsa does not require authentication, do not set these
properties

# these properties are ignored if TSA_URL is not set (no
timestamping)

# default is not set (tsa does not require authentication)

#WORKERGENID1.TSA_USERNAME=

#WORKERGENID1.TSA_PASSWORD=



#--------------------------EXTRA PROPERTIES [NOT TESTED
YET]---------------------------------------#



#if we want to embedd the crl for signer certificate inside the
signature package set to True, otherwise set to False

#default is False

#WORKERGENID1.EMBED_CRL = False



#if we want to embedd the ocsp responce for signer certificate
inside the signature package set to True, otherwise set to False

#note : issuer certificate (of signing certificate) should be in
certificate chain.

#default is False

#WORKERGENID1.EMBED_OCSP_RESPONSE = False
```