EJBCA SignServer

Manual

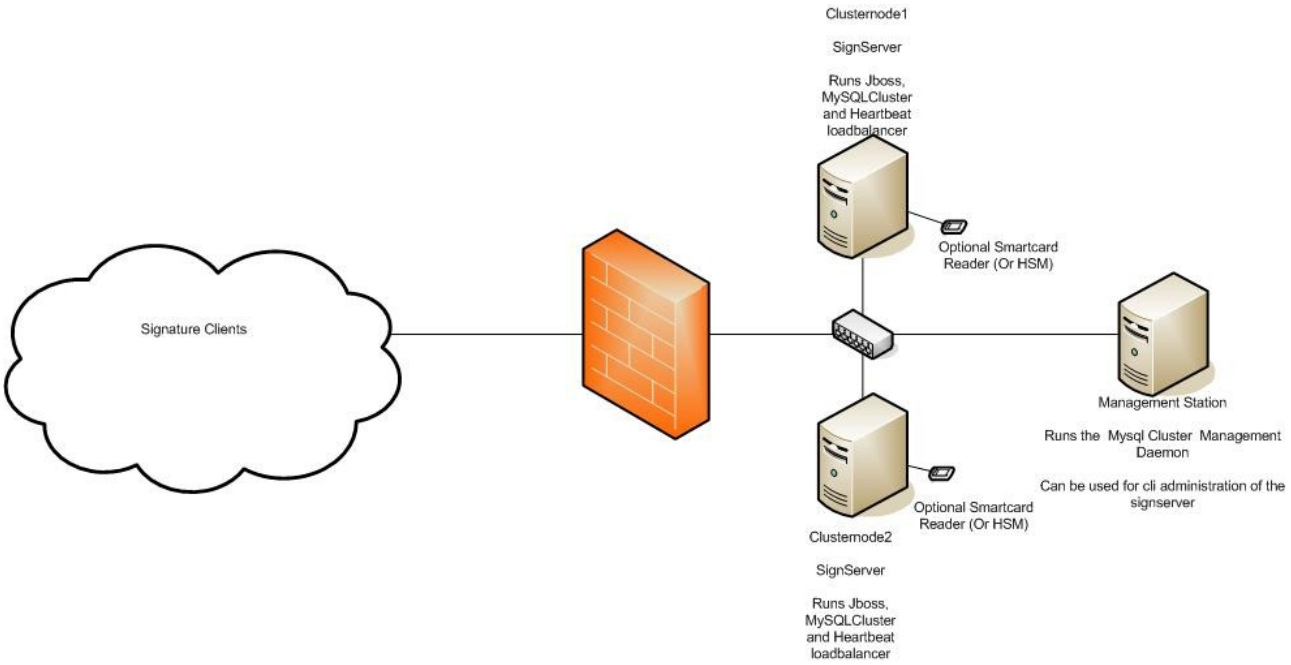
06-07-22

1 Introduction/Scope

The EJBCA SignServer is an application framework performing signatures for other applications. It's intended to be used in environments where keys are supposed to be protected in hardware but there isn't possible to connect such hardware to existing enterprise applications. Another usage is to provide a simplified method to provide signatures in different application managed from one location in the company.

The SignServer have been designed for high-availability and can be clustered for maximum reliability.

The SignServer comes with a RFC 3161 compliant TimeStamp signer serving requests through http or client-authenticated https.



Drawing 1: Overview of a possible set up of a highly available SignServer solution

2 Document History

Version	*Date*	*Name*	*Comment*
0.1	2006-06-04	Philip Vendil	Initial version of this document.
1.0 RC1	2006-08-22	Philip Vendil	Prerelease version

Table of Contents

1 Introduction/Scope....2
2 Document History....2
3 Quick-start of a Simple Time-stamp Server....4
3.1Required Software....4
3.2Installation Steps....4
4 Overall Architecture....6
5 Available Signers....7
5.1Time-stamp Signer....7
5.1.1Overview....7
5.1.2Available Properties....7
5.2MRTD Signer....8
5.2.1Overview....8
5.2.2Available Properties....8
6 Available SignTokens....9
6.1P12SignToken....9
6.1.1Overview....9
6.1.2Available Properties....9
6.2PrimeCardHSMSignToken....9
6.2.1Overview....9
6.2.2Available Properties....9
7 Setting Authorization Type....10
8 Building and Deploying the SignServer....10
9 Administrating the SignServer....11
10 Making the SignServer highly-available....12
10.1HTTP access requires a load balancer....12
10.2Setting up a MySQL Cluster....12
11 Testing....15
11.1Automatic Junit Tests....15
11.2The Test Client....15
11.3Manual Tests....15
12 For Developers....15
12.1The ISigner Interface....15
12.2The ISignToken Interface....15
13 References....15

3 Quick-start of a Simple Time-stamp Server

This section will show how to set up a quick and simple standalone time-stamp server, accepting time-stamp requests over plain HTTP.

3.1 Required Software

Java 1.5 (or 1.4) (http://java.sun.com)
Jboss-4.0.4.GA (http://www.jboss.org)
ant version 1.6.5 (http://ant.apache.org)

SignServer-1.0-Beta1 (http://www.ejbca.org)

1 web-server key-store in Java key-store format (JKS), (make sure that the server certificate have the right hostname in it's CN)
1 Root certificate of the web-server in DER encoding

1 Time-stamp key-store in PKCS12 format

3.2 Installation Steps

1. First make sure that ant, Java and Jboss is installed properly.

2. Set the JAVA_HOME and JBOSS_HOME environment variables.

3. Unzip the SignServer package and go to it's home directory.

4. Rename the web server keystore to tomcat.jks at put it in a 'p12' subdirectory. Also place the webserver root certificate in DER encoding in the same directory, call it rootcert.cer

5. Then edit the signserver_build.properties file. At least configure the httpsserver.password property.

6. Do 'ant deploy' and then start Jboss (JBOSS_HOME\bin\run.sh) in another console.

7. Edit the signserver_cli.properties and set the hostname.* properties.

8. Use the signserver cli to set the following properties:
   (if it's not executive use chmod +x bin/signserver.sh)

```
bin\signserver.sh setProperty 1 DEFAULTTSAPOLICYOID "<default policyid>"
bin\signserver.sh setProperty 1 KEYSTOREPATH "<path to time stamp P12>"
bin\signserver.sh setProperty 1 KEYSTOREPASSWORD "<keystore password>"
bin\signserver.sh setProperty 1 AUTHTYPE NOAUTH
```

   (In the path section, use '\\' for '\' in windows environment.)

Then run

```
bin\signserver.sh getConfig 1
```

And double-check the configuration with the propety reference. (Important, the properties are case sensitive).

Finally run

```
bin\signserver.sh reload
```

To activate the configuration.

9. Run the test-client to see that everything is up.

```
cd dist-client
java -jar timeStampClient.jar "http://localhost:8080/signserver/tsa?signerId=1"
```

The message "TimeStampRequest Validated" should appear once a second.
Also check JBOSS_HOME/server/default/log/server.log that successful messages appear.

4 Overall Architecture

The SignServer is a framework designed to perform different kind of digital signatures for different applications.

To access the SignServer there are different options depending on the signer used. The time-stamp signer uses HTTP for remote access and the MRDT Signer uses RMI-SSL.

There are two main concepts to the SignServer and that is Signers (which performs the actual signature and are identified by a signerId) and SignToken which is an entity that keeps track of the signing keys (can be either software or hardware based).

The applications i administrated through a command-line interface, where the properties and access control can be configured.

The currently exists two authentication modes, one using client authenticated SSL where the serial number of each signer client have to be in the signers access list for the signature to be performed. The other is an unauthenticated mode where there isn't any requirements on who requests a signature.

One SignServer can have multiple signers for different purposes.

5 Available Signers

There currently exists two types of signers. The first one is the time stamp signer generating RFC 3161 compliant timestamps using the Bouncycastle library. The other is a MRTD signer creating 'Machine Reader Travel Document' signatures using the RSA algorithm from pre-padded data.

5.1 Time-stamp Signer

The time-stamp signer have the classpath: se.primeKey.signserver.server.signers.MRTDSigner

5.1.1 Overview

The time stamp server generates time stamp tokens and have the support for the following options:

- Set of accepted policies
- Set of accepted algorithms
- Set of accepted extensions
- Accuracy microseconds
- Accuracy milliseconds
- Accuracy seconds
- Included certificate chain (currently doesn't include CRLs)
- Ordering
- TSA name

The time stamp signer currently don't support:

- CRL inclusion
- Signed attributes
- Unsigned attributes

Timestamps requests are served through a http service at the URL:
'http://<hostname>/signserver/tsa?signerId=<signer Id>'
If no 'signerId' parameter is specified then will the id of 1 be used as default.

The time-stamp signer requires a time-stamp certificate with the extended key usage 'time-stamp' only.

5.1.2 Available Properties

The following properties can be configured with the signer:

TIMESOURCE = property containing the classpath to the ITimeSource implementation that should be used. (OPTIONAL, default LocalComputerTimeSource)

ACCEPTEDALGORITHMS = A ';' separated string containing accepted algorithms, can be null if it shouldn't be used. (OPTIONAL, Strongly recommended)

Supported Algorithms are: *GOST3411, MD5, SHA1, SHA224, SHA256, SHA384, SHA512, RIPEMD128, RIPEMD160, RIPEMD256*

ACCEPTEDPOLICIES = A ';' separated string containing accepted policies, can be null if it shouldn't be used. (OPTIONAL, Recommended)

ACCEPTEDEXTENSIONS = A ';' separated string containing accepted extensions, can be null if it shouldn't be used. (OPTIONAL)

DEFAULTTSAPOLICYOID = The default policy ID of the time stamp authority (REQUIRED, if no policy OID is specified in the request then will this value be used.)

ACCURACYMICROS = Accuracy in micro seconds, Only decimal number format, only one of the accuracy properties should be set (OPTIONAL)

ACCURACYMILLIS = Accuracy in milliseconds, Only decimal number format, only one of the accuracy properties should be set (OPTIONAL)

ACCURACYSECONDS = Accuracy in seconds. Only decimal number format, only one of the accuracy properties should be set (OPTIONAL)

ORDERING = The ordering (OPTIONAL), default false.

TSA = General name of the Time Stamp Authority. (OPTIONAL)

5.2 MRTD Signer

The MRTD signer have the class path: se.primeKey.signserver.server.signers.MRTDSigner

5.2.1 Overview

The MRTD Signer performs a RSA signing operation on incoming data. The data should already be padded. This signer i used to sign 'Machine Readable Travel Documents' i.e. electronic passports.

The signatures are requested through RMI-SSL and PrimeCard is required to build it. No load balancer is required to make it redundant, this is done with the client libs.

5.2.2 Available Properties

No configuration properties exists.

6 Available SignTokens

There exists two types of sign tokens, one storing the keys in software and one on Smartcards. See the developer section for information about developing support for other HSMs.

6.1 P12SignToken

The P12SignToken signer have the classpath:
se.primeKey.signserver.server.signertokens.P12SignToken

6.1.1 Overview

A SignToken using a PKCS 12 key-store in the file-system. Can only contain one signing key.

In a clustered environment must the key store be at the same location at all nodes.

6.1.2 Available Properties

KEYSTOREPATH : The full path to the key-store to load. (required)

KEYSTOREPASSWORD : The password that locks the key-store. Used for automatic activation.

6.2 PrimeCardHSMSignToken

6.2.1 Overview

Using PrimeCardHSM it's possible to use a Smartcard to generate 2048-bit signatures. The Smartcard can perform about one signature a second.

PrimeCardHSM requires PCSCD software and Smartcard drivers. See separate documentation about installing PrimeCardHSM.

6.2.2 Available Properties

defaultKey = Hash value of the signing key on the card. See PrimeCardHSM documentation for more information.(Required)

authCode = Authentication code for automatic activation (Optional).

7 Setting Authorization Type

By default is client-certificate authentication required for a signature request to be processed. This can be changed with the ***AUTHTYPE*** property.

AUTHTYPE = NOAUTH, sets the server to not require any authentication.

AUTHTYPE = CLIENTCERT (default) requires a certificate of all the clients. The certificates must be in the signers access control list and be trusted by the Java distribution, i.e imported in JAVA_HOME/jre/lib/security/cacerts.

8 Disabling a Signer

A signer can be disabled, which means that it won't perform any signature requests or be included in the health check.

To disable a signer set the property “DISABLED” to “TRUE”

9 Archiving Responses

If there is a need to save all generated responses, then set the property “ARCHIVE” to “TRUE” and all generated responses for that signer will be saved to database.

The archived responses can later be extracted from data base using the CLI interface. See the CLI section for more information.

10 Building and Deploying the SignServer

To build and deploy the SignServer first make sure that ant, java and jboss is installed properly.

Then unzip the signserver package and go to it's home directory.

Before you start make, sure you have a web server key store in JKS format, (and make sure that the server certificate have the right host name in it's CN). Name it tomcat.jks at put
it in a p12 subdirectory. Also place the web server root certificate in DER encoding in the same directory, call it rootcert.cer

Edit the signserver_build.properties file. At least configure the httpsserver.password property.

If you are using another database that hsql built in Jboss (recommended for production use) then configure the database properties.

Then edit the signserver_server.properties file. This is the file where the available signers are configured and which SignToken they should use. Default is a time-stamp token with a PKCS12 sign token as signerId '1'.

Do ant deploy and then restart Jboss.

Then edit the signserver_cli.properties and set the hostname.* properties.

Now you are ready to configure the signer and signtoken. This is done by the bin\signserver.sh setProperty <signerId> <property> <value> command. See section 'Administrating the signserver' for more information about available cli commands.

11 Administrating the SignServer

The sign server is administrated using a cli interface.

Every signer is identified by a id. The id is the same as specified in the signserver_config.properties.

It is possible to do configuration while being in production. All configuration commands are cached until a reload command is issued and the configuration becomes active.

There is a special property file for the cli interface called signserver_cli.properties defining which nodes that exists in the cluster. The properties are:

> ***hostname.masternode*** = Should only contain one of the nodes, specified as the master node. Used by operations dealing with the database and where not all nodes in the cluster needs to be contacted.
>
> ***hostname.allnodes*** = Should contain all the nodes in the cluster, separated by a ';'. Used by the commands getStatus, activateSignToken and deactivateSignToken.

The cli lies in bin\signserver.sh/cmd

Get Status Command:
Returns the status of the given signer, it says if it's signtoken is active or not and the loaded 'active' configuration.

Get Config Command:
Returns the current configuration of signer. Observe that this configuration might not have been activated yet, not until a 'reload' command is issued.

Set Property Command:
Sets a custom property used by the signer or signer token, see reference for the Signer and SignToken for available properties.

Remove Property Command:
Removes a configured property

Upload Certificate Command:
Used when updating the certificate used for signing, sign requests

Upload Certificate Chain Command:
Used when updating the TSA signer when using a Hard Signer Token. Use this one instead of 'Upload Certificate Command' in this case.

Add Authorized Client Command:

Adds a client certificate to a signers list of acceptable clients using this signer. specify certificate serial number in hex and the IssuerDN of the client certificate.

Removes Authorized Certificate Command:
Removes added client certificates.

List Authorized Clients Commands:
Displays the current list of acceptable clients.

Activate Sign Token Command:
Used to activate hard sign tokens, authentication code is usually the PIN used to unlock the key store on the HSM. Not used if token is set to auto-activation.

Deactivate Sign Token Command:
Brings a Sign Token Off-line. Not used if token is set to auto-activation.

Archive Find from Archive Id Command:
Command used to extract archived data from database identified by the archive Id.

The Id depends on the signer, in case of the TSA is the TimeStampInfo serial number used.
The data is stored with the same file name as the archive id in the specified path.

Archive Find from Request IP Command:
Used to extract all archived data requested from a specified IP address.

All data is stored as separate files with the archive id as file name in the specified path.

Archive Find from Request Certificate Command:
Used to extract all archived data requested from a client by specified it's certificates serial number and issuer DN.

All data is stored as separate files with the archive id as file name in the specified path.

12 Making the SignServer highly-available

12.1 HTTP access requires a load balancer

HTTP based signers like the TSA can be clustered using a healthcheck servlet returning the state of the signserver. The healthcheck servlet can be configured in the file src/web/healthcheck/WEB-INF/web.xml. With the default settings will the servlet return the text 'ALLOK' when accessing the url http://localhost:8080/signserver/healthcheck/signserverhealth. If something is wrong with the sign server will and error message be sent back instead.

The healthcheck servlet can also be used to monitor the signs erver by creating a script that monitors the url periodically for error messages.

Tip, heartbeat with ldirectord is a good solution for a load balancer and works well with the sign server.

12.2 Setting up a MySQL Cluster

Here comes some notes about configuring a MySQL cluster and
perfoming the testscripts used to set it up.

Much of this HOWTO is taken from http://dev.mysql.com/tech-resources/articles/mysql-cluster-for-two-servers.html
written by Alex Davies.

A minimal cluster consists of three nodes, 2 datanodes and
one management station.

Tested with MySQL 4.1.11 and JDBC connector 3.1.10

First install mysql ('apt-get install mysql-server' on debian)

Start with the management station:
mkdir /var/lib/mysql-cluster
cd /var/lib/mysql-cluster
vi [or emacs or any other editor] config.ini

Insert the following (Without BEGIN and END):
-----BEGIN----
```
NDBD DEFAULT]
NoOfReplicas=2
[MYSQLD DEFAULT]
[NDB_MGMD DEFAULT]
[TCP DEFAULT]
# Managment Server
[NDB_MGMD]
HostName=192.168.0.3          # the IP of THIS SERVER
# Storage Engines
[NDBD]
HostName=192.168.0.1          # the IP of the FIRST SERVER
DataDir= /var/lib/mysql-cluster
[NDBD]
HostName=192.168.0.2          # the IP of the SECOND SERVER
DataDir=/var/lib/mysql-cluster
# 2 MySQL Clients
# I personally leave this blank to allow rapid changes of the mysql clients;
# you can enter the hostnames of the above two servers here. I suggest you dont.
[MYSQLD]
[MYSQLD]
```

-----END----

Now, start the managment server:

ndb_mgmd

Next is to setup the datanodes,

vi /etc/my.cnf (or /etc/mysql/my.cnf on debian)

Append the following:

```
-----BEGIN----
[mysqld]
ndbcluster
ndb-connectstring=192.168.0.3 # the IP of the MANAGMENT (THIRD) SERVER
default-storage-engine=NDBCLUSTER
[mysql_cluster]
ndb-connectstring=192.168.0.3 # the IP of the MANAGMENT (THIRD) SERVER
-----END-----
```

If you are going to use the mysql in a JBOSS cluster you should also
disable the bind-address variable so the database can be connected
from the network.
#bind-address = 127.0.0.1

The default storage variable should be set so JBOSS automatically can
create it's tables.

Now, we make the data directory and start the storage engine:

mkdir /var/lib/mysql-cluster
cd /var/lib/mysql-cluster
ndbd --initial
/etc/rc.d/init.d/mysql.server start

Note: you should ONLY use --initial if you are either starting from scratch or
have changed the config.ini file on the managment otherwise just use ndbd.

Do the exact same thing for the other node.

Next step is to check that everything is working. This is done on the
management station with the command ndbd_mgm.
In the console print 'show' and you will get something like:
----BEGIN-----
Cluster Configuration

```
[ndbd(NDB)]     2 node(s)
id=2    @192.168.115.4  (Version: 4.1.11, Nodegroup: 0)
id=3    @192.168.115.5  (Version: 4.1.11, Nodegroup: 0, Master)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @192.168.115.6  (Version: 4.1.11)

[mysqld(API)]   2 node(s)
id=4    @192.168.115.4  (Version: 4.1.11)
id=5    @192.168.115.5  (Version: 4.1.11)
```
----END-----

Which indicates that everything is ok.

TIP If you experience problems after a Mysql data node going down and complaining about not able to connect to a socket, issue this command on the managment concole:
PURGE STALE SESSIONS

To do a test of the setup using JDBC there is two small test script.

First create a test database on and a table 'ctest' with the following commands:

use test;
CREATE TABLE ctest (i INT)

Also add permissions in the database.

GRANT ALL ON test.* TO 'user'@'<yourtestscripthost>' IDENTIFIED BY 'foo123'

To test the cluster there is two small test scripts
ant test:db that does basic functionality tests
and test:dbContiously that adds an integer every second and checks that it really have been added. It outputs it's data to test_out.txt which you can tail to see what's happening when one of the servers is brought down.

13 Testing

The following tests have been done.

13.1 Automatic Junit Tests

Automatic Junit tests lies in the directory '`src/tests`'

To run the test suite do the following:

- Set the environment variable SIGNSERVER_HOME
- Make sure the sign server is deployed and JBoss is running
- do 'ant test:run'

13.2 The Test Client

There exists a test client built with the main distribution.

It only works without client authentication requirements and through HTTP.

To run the client do

```
ant
cd dist-client
java -jar timeStampClient.jar "http://<hostname>:8080/signserver/tsa?signerId=1"
```

It will continuously make one request per second.

13.3 Manual Tests

The time stamp signer have been tested with the OpenTSA client with both HTTP and HTTPS.

14 For Developers

To develop custom signers and sign tokens the following steps have to be done

14.1 The ISigner Interface

- Create a custom signer class implementing the se.primeKey.signserver.server.signers.ISigner interface. There exists a BaseSigner that can be inherited taking care of some of the functionality. If the BaseSigner is inherited only one method signData() needs to be implemented.

  You can define your own properties that the signer can use for configuration.
- Register the signer in the application by editing the signserver_server.properties file and assign it an id.
- Make sure the custom class is available to the application server
- Redeploy the signer server and it the signer is ready to be configured and then used.

14.2 The ISignToken Interface

- A custom sign token needs to implement the interface se.primeKey.signserver.server.signtokens.ISignToken. See P12SignToken for an example implementation.

  You can define own properties for a sign token in the same way as for signers. The properties are sent to the signtoken upon initialization.
- Register the signtoken to a signer by editing the signserver_server.properties file.

- Make sure the custom class is available to the application server
- Redeploy the signer server.

15 References

Java 1.5 (or 1.4) (http://java.sun.com)
Jboss-4.0.4.GA (http://www.jboss.org)
ant version 1.6.5 (http://ant.apache.org)
bouncycastle (http://www.bouncycastle.org)