Krestfield EzSign

Installation and Configuration Guide

version 4.1

Introduction

The Krestfield EzSign suite enables applications to quickly generate and verify digital signatures or encrypt and decrypt data without the need for complex programming

It provides the following key features:

Compliant Signature Generation and Verification

The server produces PKCS#7 compliant signatures (RSA or Elliptic Curve), which include signed attributes and the certificate chain. The SHA-1, SHA-2 and SHA-3 suite of digest algorithms are supported

The server performs full signature validation including path building and revocation checking, supporting both CRL and OCSP revocation checking

OCSP validation also supports the signing of OCSP requests and the inclusion of the correct Service Locator extension for use with the IdenTrust OCSP four corner model

Support for proxies to access CRL and OCSP servers is also supported, including proxies requiring authentication

AES Encryption and Decryption

AES keys of 128, 196 or 256 bits can be generated for encryption/decryption purposes. Data is encrypted using CBC (Cipher Block Chaining) and a random IV (Initialisation Vector) is generated for each and every encryption operation, ensuring the data is secured to the maximum level

Multi Token Support

The server supports several mechanisms for secure key storage, including:

- Cloud Based HSMs (including AWS Cloud HSM, Google KMS and the Thales DPoD Cloud HSM)
- PKCS#11 based HSMs (such as the nCipher and Thales/Gemalto Luna range)
- Thales PayShield HSMs

- Software
  For testing or applications that do not require hardware key protection, a software key store may be used. Keys and certificates are AES encrypted

Java based

The server is completely java based, and supports Java versions 8 onwards

Simple Client API

A thin java or .NET client is available with a simple interface to the server enabling rapid integration. You can start to generate signatures by writing only two lines of code!

A REST API is also available via the PKCloud product

Multi-Channel

The server provides key separation and the ability to support different configuration options per channel e.g. one channel can use a software key store whilst another makes use of an HSM, all from the same server

The number of channels is not limited (technically or by license)

Installation

The software is delivered as either a zip or gzip (tar.gz) file. Unzip and unpack the installation file to a location of your choice

The installation files are organised as follows

```
[Installation Folder]/EzSignVX.Y.Z/EzSignClient
[Installation Folder]/EzSignVX.Y.Z/EzSignServer
```

(Where X.Y.Z is the version number e.g. `EzSignV4.1.0`)
The EzSignClient folder (or just the `kezsign-client-X.Y.Z.jar` file) should be copied to all clients that will be accessing the server

Within the EzSignClient folder there are the following directories

- `/doc` – contains EzSignClient documentation
- `/lib` – contains the EzSignClient jar files
- `/samples` – contains client samples

Within the EzSignServer folder there are the following directories

- `/bin` – scripts to start, stop and manage the server
- `/doc` – contains this document
- `/lib` – the EzSignServer libraries
- `/logconf` – contains the logging (log4j2.xml) configuration file
- `/logs` – the default location for log files
- `/samples` – contains sample configurations

The product is bundled with a Java runtime but if you prefer to use your own, edit the `envs.sh` file in the `EzSignServer/bin` directory and edit the `JAVA_HOME` parameter so that it points to your JDK/JRE installation folder e.g.

```
JAVA_HOME=/fs01/app/jdk1.8.0_251
```

If you have specified a java version earlier than 8u161 then you must ensure that the Java Cryptography Extension Unlimited Strength Jurisdiction Policy Files for your Java version have been downloaded and installed. The download will be an archive containing two files

```
local_policy.jar
US_export_policy.jar
```

These files must be copied to the `jre/lib/security` folder of your JDK/JRE installation e.g.

```
/fs01/app/jdk1.8.0_101/jre/lib
```

The account used to run the server must have permissions to access the `EzSignServer` folder and be able to execute the scripts (located in the `/bin` directory) and jar files (located in the `/lib` directory). The account must also have read/write access to the keystore location (see the Key Store Files section later)

Client applications must be able to access the jar files located in the `EzSignClient/lib` folder

For nCipher HSMs the account running the server must have access to the nCipher `kmdata/local` folder. This is usually achieved by adding the user to the `nfast` group

Components

The EzSign product consists of the following individual components:

- The EzSign Server
  - The processing engine which manages the keys, HSMs and performs the signature generation and verification

- The EzSign Client
  - The client component which is integrated with applications and makes the calls to the server

- The EzSign Management Utility
  - The utility which allows for certificate management, the configuration of passwords and the generation of CSRs (Certificate Signing Requests)

- The EzSign Control Utility
  - A utility which permits the remote pausing/resuming of the server and the ability to alter the logging level on the fly

These are discussed in more detail below

EzSign Server

The EzSign Server is a multi-channel, digital signature and encryption processing application capable of interfacing with Hardware Security Modules for secure signature generation and validation or encryption/decryption operations

The Server supports several different key stores including a software key store (useful for non-production or systems that do not require a high level of security) and hardware key stores

Hardware key stores support any HSM (Hardware Security Module) which exposes the industry standard PKCS#11 interface (including the nCipher range of HSMs). It also supports several cloud based HSMs and the Thales PayShield HSMs

The Server makes use of a properties file for its configuration. Certain operations such as the setting of passwords, generation of CSRs (Certificate Signing Requests) and importing of issued certificates are carried out via the Management utility

Channels

A single Server can be configured with multiple channels. A channel contains only the keys and certificates which were generated or imported for that particular channel. Therefore a channel provides key separation such that different applications can access only the keys and certificates of interest to them A channel also supports its own token

Each channel can utilise a different key store and different configuration options. For example, one channel can make use of a PayShield and be configured to perform OCSP revocation checking whilst another channel can be configured for an nCipher HSM and not perform any revocation checking

There are two types of channel

- PKI: Performs digital signature generation and verification
- Symmetric: Provides encryption an decryption operations

Key Store Files

Whichever key store is used the server stores information locally about each key and certificate held by, or protected by the key store

The server will store these files in a folder which has the same name as the channel. This folder will be located beneath the key store directory (specified by the `keyStoreDir` property)

For example, if `keyStoreDir` is set to `/opt/ezsign/keystore` and a channel is configured called `ChannelOne`, then the keys and certificates associated with ChannelOne will be stored here:

```
/opt/ezsign/keystore/ChannelOne
```

These files are encrypted by the server and may contain encrypted key information (in the case of a software key store). If an HSM is used, either references to keys will be stored (identifiers or labels) or key data that is encrypted by the HSM's itself (as in the case of the PayShield HSMs)

Access to these files should be controlled such that only the server is able to access them as although they are encrypted, accidental deletion or corruption may prevent the server from operating correctly

The account used to run the server must have read/write permissions to the keystore location

Logging

Logging is performed via Apache Log4J2. See http://logging.apache.org/log4j/2.x/

By default, logs are written to:

```
EzSignServer/logs/ezsign.log
```

Logs are rolled over once they reach 100mb. At which point they are zipped and stored in a folder (a separate folder is created per month) e.g.

```
EzSignServer/logs/2019-05/ezsign-2019-05-21-1.log.gz
```

Thus saving log files automatically whilst preserving space

Log4J is highly configurable and these options can easily be altered by editing the configuration file located here:

```
EzSignServer/logconf/log4j2.xml
```

If an alternative location for the configuration file is required, update the `LOGCONFIG` property in the `ezsign-daemon-start` script to point to the new location e.g.

```
LOGCONFIG=/opt/ezsign/server1/log4j2.xml
```

Refer to the Apache documentation for more information on the configuration options

The Thread Pool

When the server starts it creates a pool of channels. Each channel loads its keys and certificates and creates a connection to the HSM (if required)

The size of the pool is configured via the `threadPoolSize` property and defaults to 1 if omitted. Larger sizes will allow for greater parallelism but may result in slower start up times. Some degree of experimentation with a particular setup may be required to find the optimum value. Usually, a good starting point is to set this value to the number of clients (or client threads) expected to connect to the server at any one time

As each request arrives the next available free channel instance processes the request. If no free instance is available the server will wait a short time and then check again whether any instances have become free. It will do this until the `waitTimeoutIfAllBusy` value has been reached at which point an error will be returned to the client. This prevents hanging if, for example external OCSP servers were running particularly slowly. The client may choose to try again if this does occur

Properties Configuration

The server's configuration is contained within a properties file. The properties file must be passed to the server (as a parameter) at start up. The server will then load the properties and then wait for client requests

Essentially the properties file contains information about the IP Addresses and Ports to listen on, the key store location, logging settings and channel configurations

The properties file has the following available items

Property	Description	Example
server.bindIpAddress	The interface/IP Address the server will bind to If omitted the server will listen on all available interfaces	**10.100.15.32**
server.port	The port the server will listen on	**5656**
server.allowedSourceIpAddresses	A comma separated list of IP Addresses which can connect to the server If omitted any client can connect	**10.100.15.40, 10.100.15.41**
server.threadPoolSize	The pool size that will be created at start up and used to process requests. If omitted, defaults to 1	**5**
server.waitTimeoutIfAllBusy	The maximum time to wait for a thread to become free. If all instances in the thread pool are busy the server will wait this number of milliseconds before returning an error If omitted, defaults to 1000 (i.e. 1 second)	**1000**
server.waitForMessageTimeout	The maximum time to wait for a message to be sent following connection. This should be increased if there are connection issues under load If omitted, defaults to 1000 (i.e. 1 second)	**1000**
server.logMessages	If false, the received and sent messages will not be logged If encrypting sensitive data, set this to false to prevent cleartext being sent to the log file If omitted defaults to true	**true**
server.ctrl.bindIpAddress	The interface/IP Address the control server will bind to If omitted the server will listen on all available interfaces	**127.0.0.1**
server.ctrl.port	The port the control server will listen on	**5657**
server.ctrl.allowedSourceIpAddresses	A comma separated list of IP Addresses which can connect to the control server If omitted no client can connect	**10.100.15.40,10.100.15.41,10.100.15.42**
server.ctrl.logStatusMessages	By default the server will log every status request message sent to the control interface. If regular monitoring is in place this can fill logs. Set this value to false to switch off this logging	**true**
server.authCode	The passphrase used to encrypt messages between the client and server. If set the client must pass the same string to the constructor. If not set, messages will be sent in the clear	**x55tHH#ih65W**
keyStoreDir	The folder beneath which all channel key stores will be held	**/opt/ezsign/STORE**
log.level	Set the logging level. The range is from 0 to 4 as follows: 0 : Logging is off 1 : Only error messages will be logged 2 : Errors and Warning messages will be logged 3 : Errors, Warnings and Events will be logged 4 : This is the debug level - all messages as well as low level events will be logged	**4**
channel.N.name	The name of the channel. This will also be used as the folder name where the keys and certificates for this channel will be stored	**CHAN1**

channel.N.type	The channel type:
	`PKI`
	`SYM`
		PKI

	PKI channels can perform signature generation and verification SYM (symmetric) channels can perform encryption or decryption	
channel.N.enabled	If false, the channel is disabled and will not be loaded. If missing, defaults to true	**true**

channel.N.tokenType	The token type:
	`SOFTWARE`
	`PKCS11`
	`HSM9000`
	`GOOGLEKMS`
	`AZUREKEYVAULT`
		SOFTWARE

channel.N.token.useModuleProtection	If an HSM normally requires a password to logon (such as some PKCS#11 tokens). This can be overridden by setting this value to true. This means the HSM will not be logged on to and only the module's keys will be used (rather than an operator card set etc)	**true**
channel.N.token.pkcs11.library	The full path to the PKCS11 library Required if `tokenType=PKCS11`	**/opt/nfast/toolkits/p11/ibcknfast.so**
channel.N.token.pkcs11.slot	The PKCS11 slot number Required if `tokenType=PKCS11`	**1**
channel.N.token.googleKms.project	The ID of the project as created in the Google Cloud Console For full details on the setup for Google KMS refer to the specific tech note	**ezsign**
channel.N.token.googleKms.location	The location of the key ring e.g. europe-west2	**global**
channel.N.token.googleKms.keyRing	The name of the key ring	**ezsignkeys**
channel.N.token.googleKms.credentialFile	The JSON file containing the private key associated with the service account with permissions to the KMS	**/opt/google/googlekms.json**
channel.N.token.googleKms.keyImportVersion	If using imported keys that are not at version 1 set this to the version of the keys you use to be imported	**1**
channel.N.token.azureKeyVault.clientId	The client ID as configured on the Azure platform	**eb1739f5-6dd2-43b1-9bf7-266717a24cf4**
channel.N.token.azureKeyVault.tenantId	The Tenant ID as configured on the Azure platform	**d12c3e45-350c-4413-2bb3-34514a35407c**
channel.N.token.azureKeyVault.keyVault	The full DNS name of the Key Vault e.g. https://yourvault.vault.azure.net/	https://yourvault.vault.azure.net/
channel.N.token.hsm9000.ipAddress	The IP Address of the HSM9000 Required if `tokenType=HSM9000`	**10.100.15.101**
channel.N.token.hsm9000.port	The port the HSM9000 listens on Required if `tokenType=HSM9000`	**1500**
channel.N.token.hsm9000.timeoutMs	The time to wait for a response from the HSM before failing Required if `tokenType=HSM9000`	**3000**
channel.N.token.hsm9000.headerLen	The HSM command header length Required if `tokenType=HSM9000`	**4**
channel.N.token.hsm9000.useVariantLmk	If the HSM has a variant LMK installed, set this to true If not specified, defaults to false (meaning a KeyBlock LMK will be used)	**false**
channel.N.token.hsm9000.lmkId	If the HSM has multiple LMKs loaded, set this to the LMK ID that you wish EzSign to use. Range 0-99 If not specified, the default LMK (as configured on the HSM) will be used	**0**
channel.N.token.password	The token password required for all token types and will be used to encrypt key store objects This must be set by running the `ezsign-manange` script If `tokenType=PKCS11` this is the PIN or Passphrase. For nCipher devices this will be the operator smartcard passphrase	**zijFhJ+BMAO8B3bYw9XD0AZlOYt4eYACY4zW9UXtZk2EC7hl+dgevA==**
channel.N.defaultKeyLabel	Relates to symmetric channels only (`type=SYM`) Specifies the default AES key label to use if none is passed to the client	**key1**
Note: All of the remaining settings relate to PKI channels only (`type=PKI`)		

channel.N.signature.algorithm	The signature key algorithm. Possible values:
	`RSA`
	`ECDSA`
	If RSA is chosen the keysize must also be specified
	If elliptic curve (ECDSA) is chosen the curve must also be specified
		RSA

channel.N.signature.keySize	`The RSA key (modulus) size in bits`
	`1024`
	`2048`
	`4096`
	`8192`
	`16384`
	`32768`
		2048

channel.N.signature.ecc.curve	The named curve e.g.
	`secp192r1`
	`secp224r1`
	`secp256r1`
	`secp384r1`
	`secp521r1`
	`secp256k1`
	`secp256k1`
	`Or any name supported by the version of java or HSM being used`
		secp256r1

channel.N.signature.hash	The hash that will be generated over the data to be signed before signing
	Possible values:
	`SHA1`
	`SHA256`
	`SHA512`
	`SHA3-224`
	`SHA3-256`
	`SHA3-384`
	`SHA3-512`
		SHA256

channel.N.signature.type	The signature type to generate. Options are:
	`PKCS7`
	`RAW`
	If RAW is chosen the signature will be PKCS#1 formatted if RSA is chosen
		PKCS7

channel.N.signature.includeCerts	What certificates to include in the signature. Possible values: `ALL` – all certificates including the root will be included `ALLEXCEPTROOT` – all certificates except the root will be included `SIGNERONLY` – only the signer certificate will be included	**ALLEXCEPTROOT**

channel.N.signature.includeContent	Whether to include the content with the signature or not. Possible values:
	`true`
	`false`
		false

channel.N.signature.includeSignedAttribs	If true signed attributes will be included in the signature including signing time, message digest and content type	**true**
channel.N.signature.keyId	The key which will be used to generate the signature This can be set by running the `ezsign-manange` script	**15723e16cd89401**
channel.N.verify.pathCheckClass	To override the default path checker with a custom version, specify the path checker class here. This must be in the classpath If omitted the default path check class is used	**org.pathcheck.PathChecker**

channel.N.verify.signedAttribsRequired	If true and signed attributes are not included in a signature being verified the signature will be rejected. Possible values:
	`true`
	`false`
		true

channel.N.verify.denyWeakSignatureHash	If true, signatures will be rejected if they have not been generated with a SHA-2 or SHA-3 hash. Possible values:
	`true`
	`false`
		false

channel.N.verify.denyWeakCertificateHash	If true, certificates will be rejected if they have not been generated with a SHA-2 or SHA-3 hash. Possible values: `true`	**false**

	`false`	
channel.N.verify.relaxRootCertExtensionChecks	If true, none of the usual certificate extension checks will be performed on the root certificate (including, key usage, basic constraints or key size) Defaults to false	**false**
channel.N.verify.relaxAllCertExtensionChecks	If true, none of the usual certificate extension checks will be performed including key usage criticality, basic constraints or key size Defaults to false	**false**
channel.N.verify.nonRepudiationRequired	If true, a signer certificate must have the non-repudiation key usage set Defaults to true	**true**
channel.N.verify.caBasicConstraintsRequired	If true, a CA certificate must have basic constraints CA extension Defaults to true	**true**
channel.N.verify.minKeySize	Sets the minimum permitted key size for all certificates in the chain Defaults to 1024	**2048**
channel.N.verify.maxKeySize	Sets the maximum permitted key size for all certificates in the chain Defaults to 8192	**8192**

channel.N.allowExpiredCerts	Whether to permit expired certificates. This MUST only be set to true in extreme circumstances (such as to maintain a live service) where other checks can be performed that ensure the certificate would otherwise still be valid. Possible values:
	`true`
	`false`
		false

channel.N.allowExpiredCertsForDays	If allow `ExpiredCerts=true` then the number of days permitted to all an expired certificate for e.g. if set to 5 a certificate will be permitted for 5 days after it has expired	**5**
channel.N.revocationChecker.type	The revocation checker type. Possible values: `NONE` - No revocation checking will be performed `CRL` - CRLs will be used to check revocation `OCSP` - OCSP will be used to check revocation `ANY` - OCSP will be tried first. If this fails (e.g. if there is no OCSP URL or if the server or status is unavailable) then CRL will be attempted	**OCSP**
channel.N.revocationChecker.ocsp. connectTimeoutSecs	If `revocationChecker.type=OCSP` then this determines the OCSP Server connection timeout. Defaults to 5 seconds	**5**
channel.N.revocationChecker.ocsp. readTimeoutSecs	If `revocationChecker.type=OCSP` then this determines the OCSP Server read timeout. Defaults to 5 seconds	**5**

channel.N.revocationChecker.ocsp.useDefaultUrl	If `revocationChecker.type=OCSP` then this determines whether the default URL (which must be specified) will always be used or, if false, the certificate's AIA extensions will be used to extract the OCSP URL. Possible values:
	`true`
	`false`
		true

channel.N.revocationChecker.ocsp.defaultUrl	If `useDefaultUrl` is `true` the default URL must be specified in this property	**https://ocsp.server .com**

channel.N.revocationChecker.ocsp.signRequest	If true, the OCSP request will also be signed. Possible values:
	`true`
	`false`
		true

channel.N.revocationChecker.ocsp.signingKeyId	if `signRequest=true`. The signing key used to sign the OCSP requests. This is set to the same key as the signature signing key	**15723e16cd89401**

channel.N.revocationChecker.ocsp.signatureHash	The hash used in the OCSP request signature generation. Used when `signRequest=true`. Possible values:
	`SHA1`
	`SHA224`
	`SHA256`
	`SHA384`
	`SHA512`
		SHA256

channel.N.revocationChecker.ocsp.checkOcspSigningCertUsage	Whether to check the key usage on the certificate used to sign the OCSP response. If true, the certificate's extended key usage will be checked for the ocsp-signing attribute. Possible values:
	`true`
	`false`
		false

channel.N.revocationChecker.ocsp.maxProducedAtAgeMins	Revocation checking will fail if the time the OCSP was generated (indicated in the producedAt field) is more than this number of minutes in the past i.e. this is the maximum lifetime allowed for an OCSP response If ommitted this will not be checked	**4**
channel.N.revocationChecker.ocsp.onlyCheckEndEntity	If this is true, only the end-entity certificate will be checked for revocation i.e. none of the other certificates in the path (nor any OCSP signing certificates) will be checked	**false**
channel.N.revocationChecker.ocsp.relaxSigningPath	RFC 6960 requires that the issuer of the certificate being checked sign the OCSP request or this is performed by a certificate that it has issued. If the response is signed by another trusted certificate (e.g. a root CA issued certificate – as is often the case for IdenTrust responders), set this option to true	**false**

channel.N.revocationChecker.ocsp.ignoreSSLErrors	If true, when sending an OCSP request, if SSL/TLS is used and the SSL certificate is not specifically trusted, setting this to true will still permit the connection. Possible values:
	`true`
	`false`
		false

channel.N.revocationChecker.ocsp.enableCache	Enable or disable OCSP response caching Default is false	**true**
channel.N.revocationChecker.ocsp.cacheSeconds	If enableCache is true, this is the maximum time to cache an OCSP response in seconds Default is 120	**60**
channel.N.revocationChecker.ocsp.cacheCaCertsOnly	If true, only OCSP responses for CA certs will be cached i.e. end entities will always be checked Default is true	**true**

channel.N.revocationChecker.ocsp.useProxy	Whether to send OCSP requests via a proxy. If true the other proxy settings are required. Possible values:
	`true`
	`false`
		true

channel.N.revocationChecker.ocsp.proxyServer	If `useProxy=true`, this specifies the proxy server address	**proxyserver**
channel.N.revocationChecker.ocsp.proxyPort	If `useProxy=true`, this specifies the proxy server port	**8080**

channel.N.revocationChecker.ocsp.proxyAuthRequired	If `useProxy=true` and the server requires authentication set this to true and specify the username and password below. Possible values:
	`true`
	`false`
		false

channel.N.revocationChecker.ocsp.proxyUsername	If `proxyAuthRequired=true` set the username here	**user1**
channel.N.revocationChecker.ocsp.proxyPassword	If `proxyAuthRequired=true` set the password here	**password**
channel.N.revocationChecker.crl.downloadFolder	The location to which CRL files will be downloaded and stored	**/opt/ezsign/crl**

channel.N.revocationChecker.crl.forceDownload	Whether to force the download of the CRL for each request. Possible values:
	`true`
	`false`
		false

channel.N.revocationChecker.crl.allowExpiredCrl	Whether to allow expired CRLs or not Note: This MUST only be used in extreme circumstances e.g. a live service outage as a revoked certificate may be accepted. Possible values:
	`true`
	`false`
		false

channel.N.revocationChecker.crl.allowExpiredCrlForDays	If `allowExpiredCrl=true` then the number of days expiry that will be permitted	**1**
channel.N.revocationChecker.crl.onlyCheckEndEntity	If this is true, only the end-entity certificate will be checked for revocation i.e. none of the other certificates in the path will be checked	**false**

channel.N.revocationChecker.crl.ignoreSSLErrors	If true, when sending an OCSP request, if SSL/TLS is used and the SSL certificate is not specifically trusted, setting this to true will still permit the connection. Possible values:
	`true`
	`false`
		true

channel.N.revocationChecker.crl.useProxy	Whether to download CRLs via a proxy. If true the other proxy settings are required. Possible values:
	`true`
	`false`
		false

channel.N.revocationChecker.crl.proxyServer	If `useProxy=true`, this specifies the proxy server address	**proxyserver**
channel.N.revocationChecker.crl.proxyPort	If `useProxy=true`, this specifies the proxy server port	**8080**

channel.N.revocationChecker.crl. proxyAuthRequired	If `useProxy=true` and the server requires authentication set this to true and specify the username and password below. Possible values:
	`true`
	`false`
		true

channel.N.revocationChecker.crl. proxyUsername	If `proxyAuthRequired=true` set the username here	**user2**
channel.N.revocationChecker.crl.proxyPassword	If `proxyAuthRequired=true` set the password here	**password1**

See Appendix A for an example properties file

Certificate Path Checking

During signature verification, EzSign will perform the following operations:

1. Verify the signature data against the signer certificates' public key
2. Build a certificate path
3. Perform path checks
4. Check certificate revocation

Step 1 performs the mathematical calculations over the signature data I.e. digesting the data, decryption and digest comparisons

Step 2 builds a path, using the certificates from the signature and certificates that may have been uploaded into the channel. A trusted root must have been imported into the channel for this step to succeed as the path must terminate on a trusted root

Each certificate is checked for time validity (the Valid From date is before the current time and the Valid To date after) and its signature is verified against the issuing certificates public key

Step 3 then performs the following steps:

1. If the setting `channel.N.verify.denyWeakCertificateHash` is true, if any of the certificates in the path have a weak hash (anything weaker than SHA-2) they will be rejected
2. If the setting `channel.N.verify.relaxAllCertExtensionChecks` is true, no further checks will be performed on the path, if this setting is false (or not set at all), then the additional checks will be performed:
    1) If the settings for key size (`channel.N.verify.minKeySize` and `channel.N.verify.maxKeySize`) are set, each certificate's key size must be within these limits
    2) The certificates must have the keyUsage extension and this must be marked as critical
    3) Signer certificates must have the Digital Signature key usage set
    4) If the setting `channel.N.verify.nonRepudiationRequired` is true, signer certificates must also have the Non Repudiation key usage set
    5) For CA and Root CA certificates they must have the Key Cert Sign key usage set and if the setting `channel.N.verify.caBasicConstraintsRequired` is true, they must also have the Basic Constraints extension.
    When Basic Constraints are checked the path length permitted will also be checked
    If the setting `channel.N.verify.relaxRootCertExtensionChecks` is true, these additional checks will not be carried out on root certificates. This may be required if legacy root certificates are being used

All the checks performed in Step 3 may be overridden by developing a custom path check class

Custom Certificate Path Checking

Specific checks may be performed on certificate paths by developing a custom java class. You may develop the custom class yourself following the details below, or Krestfield can develop one to your specific requirements. Custom path checking may be required, if for example you wish to check a certificate has been registered, check custom extensions or any other specific certificate checks your system may require

To create a custom path checker perform the following operations:

1. Create a Java project and add a reference to the `KEzSign.jar` (located in the `EzSignServer/lib` directory of the installation)
2. Create a new class (e.g. `MyCustomPathChecker`) which implements the `KPathCheckBase` interface e.g.

```
package com.myorg.ezsign.pathcheck;

import com.krestfield.ezsign.KEzSignException;
import com.krestfield.ezsign.KPathException;
import com.krestfield.ezsign.log.KSigLog;
import com.krestfield.ezsign.path.KPathCheckBase;

public class MyCustomPathChecker implements KPathCheckBase
{
    public void loadProperties(int channelNum, Properties props) throws KEzSignException
    {
        KSigLog.LogEvent("Loading properties from CustomPathCheck.loadProperties");

        // Load any specific properties required
    }

    public void check(ArrayList<X509Certificate> certPath) throws KPathException
    {
        // Perform custom checks on the path
        throw new KPathException("MyCustomPathChecker - Not yet implemented");
    }
}
```

3. Implement the `loadProperties` and `check` methods (see below), add the compiled class to the server classpath and reference this class in the server properties as follows:

   ```
   channel.N.verify.pathCheckClass=com.myorg.ezsign.pathcheck.MyCustomPathChecker
   ```

Note: If a custom path checker is implemented, the default checks will not be performed. If you require any of the custom checks to be implemented, Krestfield can supply source code snippets to assist with this

Implement the loadProperties Method

This method is passed the channel number and the properties object – which is the loaded server properties

Therefore, you may add any specific properties into an existing server properties file. These can then be read in this method

E.g. You could add in the following specific properties for channel 1:

```
channel.1.mypathchecker.allowedExtensions=2.5.29.19,2.5.29.31,1.2.840.114021.1.4.2
```

```
channel.1.mypathchecker.validityTimeDays=30
```

Or you could have server wide settings such as:

```
mypathchecker.allowedExtensions=2.5.29.19,2.5.29.31,1.2.840.114021.1.4.2
mypathchecker.validityTimeDays=30
```

If there is a failure to load any properties, throw an `KEzSignException`

Implement the check Method

All certificates in the path are included in the ArrayList, index 0 is always the end-entity (signer) certificate, other certificates are CA certificates and the last certificate in the list will be the root

Perform the required checks on these certificates and throw a `KPathException` if there are any failures or rejections

Note, that you may use the EzSign logging functions which will result in the messages being sent to the standard EzSign log files. Use the `KSigLog` class to perform this logging

Start Scripts

The server is started by calling the `ezsign-daemon-start` script located in the `EzSignServer/bin` directory. This starts the server managing process (the daemon) and also starts the server listening on the interface and port configured

The script must be passed the properties file and the master password. The master password may be passed as an additional parameter e.g.

```
ezsign-daemon-start.sh server.properties masterpassword
```

Or if the master password is held within a file, the filename may be passed after the -f switch e.g.

```
ezsign-daemon-start.sh server.properties -f masterpassfile.txt
```

Finally, the master password may be set within the environment variable `EZMASTERPASS`, in which case no password parameter is required e.g.

```
ezsign-daemon-start.sh server.properties
```

Once the daemon has started the status can be monitored via the EzSign Control utility (see below). The listening server can be stopped and started by running the following scripts

```
ezsign-server-stop
ezsign-server-start
```

These scripts do not halt the daemon process but stop and start the server listening process which will either prevent or permit further requests from the client being processed

To stop the main daemon process run the following script

```
ezsign-daemon-stop
```

Overriding Properties

The following properties can be overridden by system properties

keyStoreDir
server.port
server.bindIpAddress
server.threadPoolSize
server.allowedSourceIpAddresses
server.waitTimeoutIfAllBusy
log.level
server.ctrl.port
server.ctrl.bindIpAddress
server.ctrl.allowedSourceIpAddresses

If included as `-D` parameters at server start up, they will override any settings configured within the properties file

For example. The `ezsign-daemon-start.sh` script could be updated as follows:

```
java -Dserver.port=1234 -Dserver.bindIpAddress=127.0.0.1
     -cp $CLASSPATH com.krestfield.ezsign.server.EzSignServer $1 $2
```

(where the highlighted items were added)

This would result in the `server.port` being set to `1234` and the `bindIpAddress` to `127.0.0.1` whatever values the properties file held

These fields could be configured as parameters which would allow for the dynamic setting of these values. For example:

```
java -Dserver.port=$PORT -cp $CLASSPATH
                com.krestfield.ezsign.server.EzSignServer $1 $2
```

Or if you intended to pass these as parameters to the start script, such as:

```
ezsign-daemon-start.sh [port] [properties file] [password]
```

e.g.

```
ezsign-daemon-start.sh 5006 server.properties mypassword
```

You could update the start script as follows:

```
java -Dserver.port=$1 -cp $CLASSPATH
                com.krestfield.ezsign.server.EzSignServer $2 $3
```

This functionality allows for the dynamic control of the server which may be of use to automate multi-instance deployments

Passwords

The EzSign server requires two passwords: A Token Password and a Master Password. The Token Password is used to either encrypt software keys or to authenticate to HSMs and is stored in the properties file, encrypted. The Master Password is not stored but is used to derive a key under which the Token Passwords are encrypted

Token Passwords are per channel and each channel can be configured with a different Token Password

The Master password is per server. The same Master Password must be provided each time a Token Password is configured or the server started

When the server is started the Master Password must be provided. This password is then used to decrypt the Token Passwords stored in the properties file. If the Master Password provided is incorrect, the server cannot start

If using the Start Script (e.g. `ezsign-daemon-start.sh`) the master password can be provided in one of three ways:

As clear text e.g. the server could be started as follows:

```
ezsign-daemon-start.sh server.properties masterpassword
```

Where `masterpassword` is the master password

Or the master password can be stored within a file, and the filename them passed to the start script. If this option is chosen then the -f switch must be specified before the filename e.g.

```
ezsign-daemon-start.sh server.properties -f masterpassfile.txt
```

Where `masterpassfile.txt` is the file containing the master password

Finally, the master password may be set within the environment variable `EZMASTERPASS`, in which case no password parameter is required when calling the start script e.g.

```
ezsign-daemon-start.sh server.properties
```

On UNIX/Linux the master password could be set as follows:

```
export EZMASTERPASS=masterpassword
```

Note: If no password is passed in this way and the `EZMASTERPASS` variable is not available you will be prompted to enter a password before the server will start

EzSign Client

The EzSign client is a lightweight java (or .NET) component which exposes a simple interface enabling applications to quickly utilise the PKI and encryption functions offered by the EzSign Server

The communication between the client and server is socket based. The server name (or IP Address) and the port the server is listening on must be specified at the client

The communications between the client and server can be protected by using an Authentication Code. This is a passphrase which is provided at the client and also configured at the server (see Set Authentication Code below)

Refer to the *EzSign Client Integration Guide* for details on how to integrate the client with your applications

EzSign Management

The Management utility enables the following operations:

- Setting of Token Passwords
- Generation of CSRs (Certificate Signing Requests)
- Specifying Signing Keys
- Importing Certificates
- Importing existing HSM keys
- Deleting Certificates
- Displaying Certificates
- Generating AES Keys
- Displaying AES Keys
- Deleting AES Keys
- Translating Keystore Objects from an old password to new
- Setting the Authentication Code

The server will not start until the token passwords have been set and you will be unable to sign or verify signatures until the required keys and certificates have been generated and imported. Encrypting and decrypting requires an AES key to be generated. Therefore, the Management tool must usually be run before the server can be started (if you are duplicating configurations there is no need to re-run the tool)

Starting

To start the management utility, run the following script which requires that the server properties file is passed as a parameter e.g.

```
ezsign-manage.sh server.properties
```

(On Windows run `ezsign-manage.bat`)

Once the script starts you will be asked to enter the Master Password before being able to proceed

```
Krestfield EzSign Management
----------------------------

The master password is required to manage the server

Enter password:
```

Type the Master Password and press enter. This will then display the menu

```
Krestfield EzSign Management
----------------------------

1.  Set Passwords

2.  Generate CSR
3.  Import Certificate
4.  Import Existing HSM Keys

5.  Set Signing Key

6.  Display Certificates
7.  Display Certificates Details
8.  Delete Certificate

9.  Generate AES Key
10. Display AES Keys
11. Delete AES Key

12. Translate Keystore Objects
13. Display all Channel Objects
14. Set Authentication Code

15. Exit

Command:
```

Setting Master and Token Passwords

To set the Master and Token passwords for a channel choose option 1. You will then be prompted to choose the channel, enter the Master Password and Token Passwords and then whether you want to write the new data to the properties file. This option must be run before the server is started in order to encrypt the Token Passwords

```
Channels
--------

  1. TEST
  2. TechTest1
  3. TechTest2
```

```
Select Channel Number: 1

  Master Password
  ---------------

  This is the password which is used to start the server
  It is used to encrypt other passwords (Token Passwords etc). It is not stored

 Enter Password:
Retype Password:

  Token Password
  --------------

  For software tokens, this is the password used to encrypt keys
  For PKCS#11 tokens, this is the token PIN/Password used to authenticate,
      also referred to as the operator password
  For HSM9000 tokens, this is password is used to re-encrypt the local keys
     (although they are already encrypted under the HSM Master Key)
  This password is stored in the configuration, encrypted under the master password

 Enter Password:
Retype Password:

  A backup of the original has been saved to ..\test.properties.2016.11.12.09.03
  Ready to update properties file ..\test.properties with the new passwords

Proceed? (y/n): y

  The properties file ..\test.properties has been updated successfully.
```

Note: For sensitive operations including:

- Generating CSRs
- Importing certificates
- Deleting certificates
- Setting the signing key
- Generating AES keys
- Deleting AES keys

The channels token password will be required. When required you will receive a prompt as follows:

```
The token password is required for this operation.  Please enter the token password below

Token Password:
```

The token password must then be entered to proceed

Generating a CSR

To generate a CSR (Certificate Signing Request) choose option 2. You will be prompted to enter the selected channel again, then requested DN (Distinguished Name), key size and the filename to store the CSR

```
Generating CSR for channel TEST

Distinguished Name: CN=Test,O=Krestfield Ltd,C=GB

CSR Filename: krestfield.p10

  Generated CSR Data:
-----BEGIN CERTIFICATE REQUEST-----
MIICYTCCAUkCAQAwHjEPMA0GA1UEAwwGVGVzdC5PMQswCQYDVQQGEwJHQjCCASIwDQYJKoZIhvcN
AQEBBQADggEPADCCAQoCggEBAIMvY7RaJtHTz2jB7Nfqb2OLANmjOqACybd5FSlwFxvCSjTzoRoG
Y97aFWldDHueyVLmKKJtYMScTIs1gZSvm9guUpOhVsvQ0KaX58ZFMmRvlmeBlP2rbQIe0F1Fp724
XggI/5dXr9OKVbdIWrJkZTsFYn8bXU7nY1MAuRl5NK6CSkl6XZTvODRezL9ioFmkE09EWP4wIKyQ
zQW0Z/mn7a51eiJA+utBf3MgtUkmEzTc8Z73xrGflwt0fCnghs1gXnHNlZMkiR3l1iOPV/ppQ9l2
FsVv85JFk4eMJxP1cp+niEtoTFIc49JF/nB4u+B1aslnBBimn2oMzD/gItOdCc0CAwEAATANBgkq
hkiG9w0BAQsFAAOCAQEAYh4erwAkHL+ZuHMRcOmufK9ZJCxLXgbF2DjCgCC53xMuNwiQ7wIL6at6
N1jK8v4WlIaRE1WRSOO7k7OG3MwXBtDy3oVmr5NWEKu0WSVbwM/7mVjkczuBxLXdr/pEeyH9Nm02
h3kH0sO25xt5BjoCzcoHIYmmHxa5tyzjchqTE5Fw68S/7rusodvAbEwNKYQQhHTCwAzuusGDxb0D
+JHNTk63zenDHw56pcURsvmA16BSXF30MatVcoTd7elqmAho+yUm0c0CVOfGoH2zFv7jcyl8jDtm
w3Kg45sND7L2A9hII0QGy3L+seE/GpMaoxhhDBPxbIfAkqecxyapA+aX3w==
-----END CERTIFICATE REQUEST-----

  CSR written to krestfield.p10
```

The CSR should then be processed by the CA who will issue the certificate. Once the certificate has been issued the Import Certificate option can be chosen to import the certificates

Importing Certificates

To import root certificates or certificates issued from a CSR choose option 3. You will then be prompted for the channel and a path to the certificate.

```
  Channels
  --------

    1. TEST

Select Channel Number: 1

Path to certificate: /opt/ezsign/certs/signing.cer

  Certificate imported successfully
```

If the certificate is associated with a private key on the token, you will also be prompted if you wish to set this as the new signing certificate:

```
Do you wish to set this as the default signing certificate? (y/n): y
  A backup of the original has been saved to ..\samples\test.properties.2016.10.12.11.00
```

```
  Ready to update properties file ..\samples\test.properties with the new passwords
Proceed? (y/n): y

  The properties file ..\samples\test.properties has been updated successfully.
```

Choosing **y** will result in the properties file being updated. If you do not wish to set this as the signing certificate now, you can do this later by running option 5. Set Signing Key

Note that for signing certificates, all certificates in the path must be imported

Importing Existing Keys and Certificates

If you are using a PKCS#11 HSM which has been used by another system you may import these objects into EzSign. To do this choose option `4.Import Existing HSM Keys`

Choose the channel to import the objects to. Note: the channel selected must be configured to use the HSM from which the objects are to be imported

If not previously entered, you will be prompted for the token password:

```
  The token password is required for this operation.  Please enter the token password
  below

Token Password:

  Password verified OK

  Importing objects from the HSM...

  Objects imported from the HSM OK
```

The objects imported can be examined by running the `Display Certificates` or `Display all Channel Objects` options. The signing key can be selected by running the `Set Signing Key` option

Setting the Signing Key

To set the default signing key (that is the key that will be used to sign data), choose option 5. You will then be presented with a list of available certificates which have associated private keys and can therefore be used to sign data. Note: a certificate will not be available for signing if there is not a complete path for that certificate

```
  Current Signing Certificate:
  ----------------------------

       Subject: CN=Test Cert 1, O=Krestfield Ltd, C=GB
       Issuer : CN=Krestfield Test CA, OU=Engineering, O=Krestfield Ltd, C=GB
       Valid From: Wed Oct 12 08:12:48 BST 2016
       Valid To  : Thu Oct 12 08:12:48 BST 2017
       Serial Number: 5b000000348a17f73059f07217000000000034


  Available Signing Certificates:
```

```
  ------------------------------
    1. Subject: CN=Test Cert 2, O=Krestfield Ltd, C=GB
       Issuer : CN=Krestfield Test CA, OU=Engineering, O=Krestfield Ltd, C=GB
       Valid From: Wed Nov 12 18:12:48 BST 2016
       Valid To  : Thu Nov 12 18:12:48 BST 2017
       Serial Number: 5b000000348a17f73059f07217000002010F13


Select Certificate Number: 1

  A backup of the original has been saved to ..\test.properties.2016.10.12.09.28
  Ready to update properties file ..\test.properties with the new passwords

Proceed? (y/n): y

  The properties file ..\test.properties has been updated successfully.
```

Displaying and Deleting Certificates

Options 6, 7 and 8 are used to delete and display the available certificates. When certificates are displayed they are shown in the following format

```
  Current Signing Certificates:
  -----------------------------
    1. Subject: CN=Test 1, O=Krestfield Ltd, C=GB
       Issuer : CN=Krestfield Test CA2, OU=Engineering, O=Krestfield Ltd, C=GB
       Valid From: Wed Oct 12 08:12:48 BST 2016
       Valid To  : Thu Oct 12 08:12:48 BST 2017
       Serial Number: 5b000000348a17f73059f07217000000000034

    2. Subject: CN=Krestfield Test CA2, OU=Engineering, O=Krestfield Ltd, C=GB
       Issuer : CN=Krestfield Test Root CA2, OU=Engineering, O=Krestfield Ltd, C=GB
       Valid From: Fri Dec 04 08:22:54 GMT 2015
       Valid To  : Thu Nov 29 08:22:54 GMT 2035
       Serial Number: 3e000000022542122l0f37d14f000000000002

    3. Subject: CN=Krestfield Test Root CA2, OU=Engineering, O= Krestfield Ltd, C=GB
       Issuer : CN=Krestfield Test Root CA2, OU=Engineering, O= Krestfield Ltd, C=GB
       Valid From: Thu Dec 03 09:13:47 GMT 2015
       Valid To  : Wed Dec 03 09:23:47 GMT 2036
       Serial Number: 5de549fbaf4b14b141d63d3c631b27c0


  Other Certificates:
  -------------------

    4. Subject: CN=Krestfield Test Root CA1, OU=Engineering, O=Krestfield Ltd, C=GB
       Issuer : CN=Krestfield Test Root CA1, OU=Engineering, O=Krestfield Ltd, C=GB
       Valid From: Thu Nov 03 09:13:47 GMT 2015
       Valid To  : Wed Nov 03 09:23:47 GMT 2036
       Serial Number: 5de549abaf4b14b141d63d3c631bde54
```

The first section displays the Current Signing Certificates. This is based on the selected signing key and displays the complete path. The second sections displays all other certificates which are stored but not included in the current signing path

Generating AES Keys

Choose option 9 to generate an AES key. Enter the key size and the key label as follows:

```
Enter AES Key Size (128, 192 or 256): 256

Enter a unique label for this key: key10

AES key generated OK
```

Once the key has been created it can be used to encrypt and decrypt data via the client, where the label set above must be specified to select this key

Displaying and Deleting AES Keys

AES keys can be displayed and deleted by choosing options 10 and 11

```
Current AES Keys:
-----------------------------
     #    Key Size   Date Created         Label
     ---  --------   -------------------  --------
       1  128bits    11-3-2017 17:16:45   key2
       2  256bits    13-3-2017 21:33:00   testkey5
       3  192bits    11-3-2017 17:14:52   key1
       4  256bits    19-3-2017 09:20:26   key10
       5  256bits    11-3-2017 17:17:05   key3
```

Key details will be shown including the key size, date created and associated label

Translate KeyStore Objects

If you wish to translate keystore objects from one token password to another. For example, when refreshing passwords for software tokens or if an HSM's objects have been translated to another operator cardset, choose option 12

You will be prompted to choose the channel and then enter the current token password

```
Please enter the current token password

Enter Password:
Retype Password:
```

Then enter the new token password. This is the new password or new operator cardset passphrase:

```
Please enter the NEW token password

Enter Password:
Retype Password:
```

```
Objects have been translated to the new password successfully
A backup was made of the original objects and stored in the SIGN keystore folder
```

The objects will be translated and re-encrypted under the new password. The pre-translated objects will be backed up to a timestamped folder within the keystore directory e.g. `/20181101_1015_BACKUP`

Display KeyStore Objects

To display all the objects stored within a channel, select option 13, then enter the channel

All objects details will be displayed indicating what type of object they are (i.e. private key, certificate etc), the ID and filename e.g.

```
Current Objects:
----------------
Object ID: 15fbc93ade39910
Created  : Tue Nov 14 22:08:35 GMT 2017
Type     : CERTIFICATE
  Subject: CN=Krestfield CA, OU=PKI Services, O=Krestfield Ltd, C=GB
  Issuer : CN=Krestfield Root, OU=PKI Services, O=Krestfield Ltd, C=GB
  Serial Number: 380000000e3308b4434ca3142100000000000e
Filename : 15fbc93ade39910.cer

Object ID: 15fbc93956f0472
Created  : Tue Nov 14 22:08:29 GMT 2017
Type     : CERTIFICATE
  Subject: CN=Krestfield Root, OU=PKI Services, O=Krestfield Ltd, C=GB
  Issuer : CN=Krestfield Root, OU=PKI Services, O=Krestfield Ltd, C=GB
  Serial Number: 5f0609d62d60709e45c1051774a13021
Filename : 15fbc93956f0472.cer

Object ID: PRVK:CC035985F170B51460A3B659523A8D757AD0CBCD
Created  : Thu Apr 05 22:03:12 BST 2018
Type     : PRIVATE KEY
Filename : PRVK_CC035985F170B51460A3B659523A8D757AD0CBCD.priv

Object ID: PUBK:13996A889E52A844660D083D631EE0F30405C576
Created  : Tue Nov 14 22:08:35 GMT 2017
Type     : CERTIFICATE
  Subject: CN=SSAS Cert, O=Krestfield, C=GB
  Issuer : CN=Krestfield CA, OU=PKI Services, O=Krestfield Ltd, C=GB
  Serial Number: 450000000716edae60376f2200000000000007
Filename : PUBK_13996A889E52A844660D083D631EE0F30405C576.cer
```

Set Authentication Code

To set the authentication code which is used to encrypt traffic between the client and server, choose option 14

Enter the master password, followed by the authentication code:

```
You will now be asked to enter the Master Password
Followed by the Authentication Code Password
```

```
  Master Password
  ---------------

  This is the password which is used to start the server
  It is used to encrypt other passwords (Token Passwords etc). It is not stored

 Enter Password:
Retype Password:

  Enter the Authentication Code
  -----------------------------

  This is a password used to secure traffic from the client to the server
  Once this has been set, the client must provide this same password to the EzSign client

 Enter Password:
Retype Password:
```

You will now be prompted whether to set this as the server code only (i.e. securing comms between the client and the server when sending messages), as the server control code only (i.e. securing comms between the client utils scripts and the control server) or both:

```
  Do you want to set this password as the:

  1. Server Authentication Code
  2. Server Control Authentication Code
  3. Both
  4. Cancel

Enter Choice: 1
  A backup of the original has been saved to ..\config\config.properties.2018.11.05.12.23
```

The server configuration will be updated and a backup made

EzSign Control

The Control utility connects to the running server on the specified control port and enables the live monitoring of the server status (running or stopped), the pausing and resuming of the server and the ability to alter the logging level whilst the server is running

Run the `ezsign-server-ctrl` script to start the EzSign Control utility

The script requires two parameters: the IP Address and Control Port of the server e.g:

```
ezsign-server-ctrl 127.0.0.1 5657
```

These are the values set by the following properties in the server properties file:

```
server.ctrl.bindIpAddress

server.ctrl.port
```

If an Authentication Code is being used on the server's control interface, this should also be provided at startup via the authCode parameter e.g.

```
ezsign-server-ctrl 127.0.0.1 5657 authCode=1f84-66c2-29f5-a60b
```

In this case the Authentication Code is `1f84-66c2-29f5-a60b`

Starting the utility in this way will display the menu, enabling the user to select the option required

This script may also be called with an action supplied to carry out the operation with no further interaction required e.g.

To obtain the running status of the server:

```
ezsign-server-ctrl 127.0.0.1 5657 status
```

To pause the server:

```
ezsign-server-ctrl 127.0.0.1 5657 pause
```

To resume the server

```
ezsign-server-ctrl 127.0.0.1 5657 resume
```

To shut down the server process

```
ezsign-server-ctrl 127.0.0.1 5657 stopdaemon
```

To change the log level

```
ezsign-server-ctrl 127.0.0.1 5657 loglevel=3
```

Support

All questions, queries around the API described within this document should be directed to Krestfield Support at the following email address:

```
support@krestfield.com
```

Or visit our site here:

```
https://www.krestfield.com
```

Appendix A – Server Properties File

The following describes the possible options that are available to be used within the server properties file
Note: all items marked in red must be configured for the specific installation

```
################################################################################
# Server Control
# These settings define how the server will listen to control messages
# such as pause server, restart server, get server status etc.
################################################################################

# The IP Address that the control listener will bind to
server.ctrl.bindIpAddress=127.0.0.1

# A comma seperated list of IP addresses that the control server
# will permit to send control messages
# Set this to local host (127.0.0.1) to ensure that only the same server can
# send control messages
server.ctrl.allowedSourceIpAddresses=127.0.0.1,192.168.1.78

# The port the control server will listen on
server.ctrl.port=5657

################################################################################
# Server
# These settings define how the server listens to API messages sent from
# the client
# The client must send messages to the port and IP address defined here
################################################################################

# The port the server will listen on
# Ensure this is not the same as the control server above
server.port=5656

# The IP Address that the server listener will bind to
# If this is commented out/missing the server will bind to all interfaces
#server.bindIpAddress=127.0.0.1

# A comma seperated list of IP addresses that the server
# will permit to send messages
# Set this to local host (127.0.0.1) to ensure that only the same server can
# send messages
# If omitted any client can send messages
#server.allowedSourceIpAddresses=127.0.0.1,192.168.1.78

# The pool size that will be created at start up and used to process requests
# If omitted, defaults to 1
server.threadPoolSize=5

# The maximum time (milliseconds) to wait for a thread to become free
# If all instances in the thread pool are busy the server will wait this
# number of milliseconds before returning an error
server.waitTimeoutIfAllBusy=1000

# The maximum time (milliseconds) to wait for a message to be received following
# connection
# Increase if there are issues connecting under load
server.waitForMessageTimeout=1000

# Whether to log all received and returned messages
# If encrypting sensitive data, set this to false to prevent cleartext being
# sent to the log file
server.logMessages=false
```

```
################################################################################
# Key Store Location
# This is where all of the keys and certificates (either encrypted files
# or references to objects residing on HSMs) will be stored
# A folder per channel is created beneath this location e.g.
# if keyStoreDir=/var/keystore and you configure channel1 and channel2
# folders /var/keystore/channel1 and /var/keystore/channel2 will
# be created
################################################################################
keyStoreDir=/var/KEYSTORE

################################################################################
# Logging settings
# Note: Changes to log filenames, rollover frequency and size etc. can be
# made by editing the log4J2.xml file located in the EzSignServer/logconf
# directory
################################################################################
# Set the logging level.  The range is from 0 to 4 as follows:
# 0 : Logging is off
# 1 : Only error messages will be logged
# 2 : Errors and Warning messages will be logged
# 3 : Errors, Warnings and Events will be logged
# 4 : This is the debug level - all messages as well as low level events will be logged
log.level=3

################################################################################
# Channel settings
# Channels are configured here
# The format shall be:
#   channel.1.property1
#   channel.1.property2
#   ...
#   channel.2.property1
#   channel.2.property2
#   etc.
################################################################################

# The channel name.  Do not include spaces in the name
# A folder will be created beneath keyStoreDir with this name
channel.1.name=CHANNEL1

# Whether the channel is enabled or disabled
# If disabled it will not be loaded and cannot be used
channel.1.enabled=true

# The channel type - can be PKI or SYM
# PKI channels can sign and verify, SYM channels can encrypt and decrypt
channel.1.type=PKI

# If the channel is type=SYM the default key label can be specified here
# If no channel name is passed
channel.1.defaultKeyLabel=test1

# The token type - can be SOFTWARE, PKCS11, HSM9000 or GOOGLEKMS
# If PKCS11 the pkcs11.library and pkcs11.slot settings must also be specified
# If HSM9000 the hsm9000.port, hsm9000.ipAddress, hsm9000.timeoutMs
# and hsm9000.headerLen settings must also be specified
channel.1.tokenType=SOFTWARE

# If tokenType=PKCS11 the following properties must be set
# The path to the PKCS#11 library:
channel.1.token.pkcs11.library=/opt/nfast/toolkits/cknfast-64.dll
# The slot number:
channel.1.token.pkcs11.slot=1

# If tokenType=GOOGLEKMS the following properties must be set
# The ID of the project as created in Google Cloud
channel.1.token.googleKms.project=ezsign
```

```
# The location of the key ring
channel.1.token.googleKms.location=europe-west2
# The name of the key ring
channel.1.token.googleKms.keyRing=ezsign
# if importing keys under a version other than 1, set the version to import
channel.1.token.googleKms.keyImportVersion=2

# If tokenType=HSM9000 the following properties must be set
# The HSM IP Address:
channel.1.token.hsm9000.ipAddress=10.0.1.101
# The HSM listening port:
channel.1.token.hsm9000.port=1500
# The timeout to use when connecting to the HSM:
channel.1.token.hsm9000.timeoutMs=10000
# The command header length:
channel.1.token.hsm9000.headerLen=4
# Whether to use a Variant or KeyBlock LMK:
channel.1.token.hsm9000.useVariantLmk=true
# The ID of the LMK to use:
channel.1.token.hsm9000.lmkId=1

# The token password required for all token types
# This must be set by running the ezsign-manange script
# If tokenType=SOFTWARE this is used to encrypt the keys and certificates only
# If tokenType=PKCS11 this is the PIN or Passphrase, for nCipher devices this will be
# the operator smartcard passphrase
# If tokenType=HSM9000 this is used to re-encrypt the keys and certificates
channel.1.token.password=zijFhJ+BMAO8B3bYw9XD0AZlOYt4eYACY4zW9UXtZk2EC7hl+dgevA==

# The signature key algorithm
# RSA - RSA algorithm
# ECDSA - Elliptic Curve Digital Signature Algorithm
channel.1.signature.algorithm=RSA

# The signature key size (if algorithm ia RSA)
channel.1.signature.keySize=2048

# The signature key named curve (if algorithm ia ECDSA)
channel.1.signature.curve=secp256r1

# The hash that will be used to sign the data
# SHA1, SHA256, SHA512, SHA3-256 etc
channel.1.signature.hash=SHA256

# What certificates to include in the produced signature
# ALL - All certificates including the root
# SIGNERONLY - Only the signer certificate
# ALLEXCEPTROOT - All certificates in the path except the root
channel.1.signature.includeCerts=ALL

# If true, signed attributes containing signing time, content type
# and message digest will be included in the signature
channel.1.signature.includeSignedAttributes=true

# Whether to include the content within the signature or not
channel.1.signature.includeContent=false

# The key which will be used to generate the signature
# This can be set by running the ezsign-manange script
channel.1.signature.keyId=15723e16cd89401

# To override the default path checker with a custom version, specify the
# path checker class here.  This must be in the classpath
channel.1.verify.pathCheckClass=com.myorg.ezsign.pathcheck.MyCustomPathChecker

# If true, the signature being verified must include signed attributes
# otherwise it will be rejected
channel.1.verify.signedAttribsRequired=true
```

```
# If true, the signature being verified must have been generated with
# a strong hash or be rejected
# i.e. SHA-2 range and above i.e. MD5 and SHA-1 will be rejected
channel.1.verify.denyWeakSignatureHash=false

# If true, the certificates being verified must have been generated with
# a strong hash or be rejected
# i.e. SHA-2 range and above i.e. MD5 and SHA-1 will be rejected
channel.1.verify.denyWeakCertificateHash=false

# If true, none of the usual certificate extension checks will be
# performed on the root certificate (including, key usage, basic
# constraints or key size)
# Defaults to false
channel.1.verify.relaxRootCertExtensionChecks=false

# If true, none of the usual certificate extension checks will be
# performed including, key usage, basic constraints or key size
# Defaults to false
channel.1.verify.relaxAllCertExtensionChecks=false

# If true, a signer certificate must have the non-repudiation
# key usage set
# Defaults to true
channel.1.verify.nonRepudiationRequired=true

# If true, a CA certificate must have basic constraints CA extension
# Defaults to true
channel.1.verify.caBasicConstraintsRequired=true

# Sets the minimum permitted key size for all certificates in the chain
# Defaults to 1024
channel.1.verify.minKeySize=2048

# Sets the maximum permitted key size for all certificates in the chain
# Defaults to 8192
channel.1.verify.maxKeySize=8192

# Whether to permit expired certificates
# This MUST only be set to true in extreme circumstances (such as to
# maintain a live service) where other checks can be performed that
# ensure the certificate would otherwise still be valid
channel.1.allowExpiredCerts=true

# If allow ExpiredCerts=true then the number of days permitted
# to all an expired certificate for e.g. if set to 5 a certificate
# will be permitted for 5 days after it has expired
channel.1.allowExpiredCertsForDays=5

# The revocation checker type
# This can be:
# NONE - No revocation checking will be performed
# CRL - CRLs will be used to check revocation
# OCSP - OCSP will be used to check revocation
channel.1.revocationChecker.type=OCSP

# If revocationChecker.type=OCSP then this determines whether the
# default URL (which must be specified) will always be used or,
# if false, the certificate's AIA extensions will be used to extract
# the OCSP URL
channel.1.revocationChecker.ocsp.useDefaultUrl=true

# If useDefaultUrl is true the default URL must be specified here
channel.1.revocationChecker.ocsp.defaultUrl=https://ocsp.server.co.uk

# If true, the OCSP request will also be signed with the signing
# key specified
```

```
channel.1.revocationChecker.ocsp.signRequest=true

# The signing key used to sign the OCSP requests if signReques=true
# This can be set by running the ezsign-manange script
channel.1.revocationChecker.ocsp.signingKeyId=1557ca2b4ba3283

# The hash used in the OCSP request signature generation
# Used when signRequest=true.  Can be SHA1, SHA256, SHA512 etc
channel.1.revocationChecker.ocsp.signatureHash=SHA1

# Whether to check the key usage on the certificate used to sign the
# OCSP response.
# If true, the certificate's extended key usage will be checked for the
# ocsp-signing attribute
channel.1.revocationChecker.ocsp.checkOcspSigningCertUsage=true

# Revocation checking will fail if the time the OCSP was generated
# (indicated in the producedAt field) is more than this number of
# minutes in the past i.e. this is the maximum lifetime allowed
# for an OCSP response
# If ommitted this will not be checked
channel.1.revocationChecker.ocsp.maxProducedAtAgeMins=4

# If true, when sending an OCSP request, if SSL/TLS
# is used and the SSL certificate is not specifically trusted, setting
# this to true will still permit the connection
channel.1.revocationChecker.ocsp.ignoreSSLErrors=true

# Enable or disable OCSP response caching
# Default is false
channel.1.revocationChecker.ocsp.enableCache=true

# If enableCache is true, this is the maximum time to cache an OCSP response
# in seconds
# Default is 120
channel.1.revocationChecker.ocsp.cacheSeconds=120

# If true, only OCSP responses for CA certs will be cached
# i.e. end entities will always be checked
# Default is true
channel.1.revocationChecker.ocsp.cacheCaCertsOnly=true

# Whether to send OCSP requests via a proxy
# If true the other proxy settings are required
channel.1.revocationChecker.ocsp.useProxy=true

# If useProxy=true, this specifies the proxy server address
channel.1.revocationChecker.ocsp.proxyServer=10.0.0.12

# If useProxy=true, this specifies the proxy server port
channel.1.revocationChecker.ocsp.proxyPort=8080

# If useProxy=true and the server requires authentication set this to true
# and specify the username and password below
channel.1.revocationChecker.ocsp.proxyAuthRequired=true

# If proxyAuthRequired=true set the username here
channel.1.revocationChecker.ocsp.proxyUsername=user1

# If proxyAuthRequired=true set the password here
channel.1.revocationChecker.ocsp.proxyPassword=password

# If revocationChecker.type=CRL then the following properties are required
# The location to which CRL files will be downloaded and stored:
channel.1.revocationChecker.crl.downloadFolder=/opt/crl

# Whether to force the download of the CRL for each request:
channel.1.revocationChecker.crl.forceDownload=false
```

```
# Whether to allow expired CRLs or not
# Note: This MUST only be used in extreme circumstances e.g. a live service outage
# as a revoked certificate may be accepted
channel.1.revocationChecker.crl.allowExpiredCrl=false

# If allowExpiredCrl=true then the number of days expiry that will be permitted
channel.1.revocationChecker.crl.allowExpiredCrlForDays=5

# If true, when downloading a CRL, if SSL/TLS
# is used and the SSL certificate is not specifically trusted, setting
# this to true will still permit the connection
channel.1.revocationChecker.crl.ignoreSSLErrors=true

# Whether to send CRL download requests via a proxy
# If true the other proxy settings are required
channel.1.revocationChecker.crl.useProxy=true

# If useProxy=true, this specifies the proxy server address
channel.1.revocationChecker.crl.proxyServer=10.0.0.12

# If useProxy=true, this specifies the proxy server port
channel.1.revocationChecker.crl.proxyPort=8080

# If useProxy=true and the server requires authentication set this to true
# and specify the username and password below
channel.1.revocationChecker.crl.proxyAuthRequired=true

# If proxyAuthRequired=true set the username here
channel.1.revocationChecker.ocsp.proxyUsername=user1

# If proxyAuthRequired=true set the password here
channel.1.revocationChecker.crl.proxyPassword=password
```

Appendix B – PayShield Error Codes

The following lists the possible error codes that may be returned by the Thales PayShield (HSM 9000/10k)

Code	Description
00	No error
01	Verification failure or warning of imported key parity error
02	Key inappropriate length for algorithm
04	Invalid key type code
05	Invalid key length flag
10	Source key parity error
11	Destination key parity error or key all zeros
12	Contents of user storage not available. Reset, power-down or overwrite
13	Invalid LMK Identifier
14	PIN encrypted under LMK pair 02-03 is invalid
15	Invalid input data (invalid format, invalid characters, or not enough data provided)
16	Console or printer not ready or not connected
17	HSM not authorized, or operation prohibited by security settings
18	Document format definition not loaded
19	Specified Diebold Table is invalid
20	PIN block does not contain valid values
21	Invalid index value, or index/block count would cause an overflow condition
22	Invalid account number
23	Invalid PIN block format code. (Use includes where the security setting to implement PCI HSM limitations on PIN Block format usage is applied, and a Host command attempts to convert a PIN Block to a disallowed format.)
24	PIN is fewer than 4 or more than 12 digits in length
25	Decimalization Table error
26	Invalid key scheme
27	Incompatible key length
28	Invalid key type
29	Key function not permitted
30	Invalid reference number
31	Insufficient solicitation entries for batch
32	LIC007 (AES) not installed
33	LMK key change storage is corrupted
39	Fraud detection
40	Invalid checksum
41	Internal hardware/software error: bad RAM, invalid error codes, etc.
42	DES failure
43	RSA Key Generation Failure
47	Algorithm not licensed
49	Private key error, report to supervisor
51	Invalid message header
65	Transaction Key Scheme set to None
67	Command not licensed
68	Command has been disabled
69	PIN block format has been disabled
74	Invalid digest info syntax (no hash mode only)
75	Single length key masquerading as double or triple length key
76	Public key length error
77	Clear data block error
78	Private key length error
79	Hash algorithm object identifier error
80	Data length error. The amount of MAC data (or other data) is greater than or less than the expected amount.
81	Invalid certificate header
82	Invalid check value length
83	Key block format error
84	Key block check value error
85	Invalid OAEP Mask Generation Function
86	Invalid OAEP MGF Hash Function
87	OAEP Parameter Error

90	Data parity error in the request message received by the HSM
91	Longitudinal Redundancy Check (LRC) character does not match the value computed over the input data (when the HSM has received a transparent async packet)
92	The Count value (for the Command/Data field) is not between limits, or is not correct (when the HSM has received a transparent async packet)
A1	Incompatible LMK schemes
A2	Incompatible LMK identifiers
A3	Incompatible keyblock LMK identifiers
A4	Key block authentication failure
A5	Incompatible key length
A6	Invalid key usage
A7	Invalid algorithm
A8	Invalid mode of use
A9	Invalid key version number
AA	Invalid export field
AB	Invalid number of optional blocks
AC	Optional header block error
AD	Key status optional block error
AE	Invalid start date/time
AF	Invalid end date/time
B0	Invalid encryption mode
B1	Invalid authentication mode
B2	Miscellaneous keyblock error
B3	Invalid number of optional blocks
B4	Optional block data error
B5	Incompatible components
B6	Incompatible key status optional blocks
B7	Invalid change field
B8	Invalid old value
B9	Invalid new value
BA	No key status block in the keyblock
BB	Invalid wrapping key
BC	Repeated optional block
BD	Incompatible key types
BE	Invalid keyblock header ID