Krestfield EzSign

.NET Client Integration Guide

Version 4.0

Introduction

The Krestfield EzSign Client is a lightweight .NET package which interfaces with the EzSign Server enabling applications to quickly generate and verify digital signatures or encrypt and decrypt data without the need for complex programming

The client also provides utilities to hash data, supporting both SHA-1 and the SHA-2 range of hash algorithms

This guide details the steps required to integrate the client into applications and make use of the API

For server side setup, please refer to the *EzSign Installation and Configuration Guide*

Library

The client consists of the following library files:

- `EzSignClient.dll`
- `EzSignClientUtils.dll`

These are located at `[installation folder]\DotNetClient\lib`

Where `[installation folder]` is the location the server was installed

To make use of the signing, verification and encryption functions, add a reference to the `EzSignClient.dll` file from your project

To make use of the hashing utilities, add a reference to the `EzSignClientUtils.dll` file from your project

Client API

The API to the EzSign client has intentionally been kept simple. Decisions on which keys to use, what certificates to include in signatures and what hashing algorithms to use are configured at the server level

All the client API calls are included in the EzSignClient class, which is included in the `com.krestfield.ezsign.client` namespace

Constructor

The constructor is defined as:

```
public EzSignClient(String host, int port)
```

Where:
`host` is the EzSign Server hostname (or IP Address)
`Port` is the EzSign Server listening port number

The default connection timeout to the server is 5 seconds (5000 milliseconds). But this can be overridden by using the following constructor:

```
public EzSignClient(String host, int port,
                    int timeoutInMs, int readTimeoutInMs)
```

Where:
`timeoutInMs` is the connection timeout expressed in milliseconds
`readTimeoutInMs` is the read timeout expressed in milliseconds

Example:

```
using com.krestfield.ezsign.client;

namespace App1
{
    class Program
    {
        static void Main(string[] args)
        {
            EzSignClient client = new EzSignClient("10.100.56.11", 5656);

        }
    }
}
```

If the communications between the client and server are to be secured, an Authentication Code may be used. The Authentication Code is provided by using one of the following constructors:

```
public EzSignClient(String host, int port, String authCode)

public EzSignClient(String host, int port,
                int timeoutInMs, int readTimeoutInMs, String authCode)
```

This will result in the encryption of the traffic between the client and server

There are no restrictions on what Authentication Code can be used, but a longer more complex code will increase security

The same Authentication Code must also be configured on the server. Refer to the Installation and Configuration Guide for details on how to configure this on the server

Generate Signature Methods

The following method is called to generate a signature:

```
public byte[] signData(String channelName, byte[] dataToSign,
                       bool isDigest)
```

The signature returned is dependent on the signature type specified at the server and will be either a PKCS#7 formatted signature or a raw PKCS#1 signature

Note: For large data sets, it is recommended to hash the data beforehand and provide this as the `dataToSign` together with `isDigest=true`. This prevents large amounts of data being passed between the client/server interface. Note: The client utils can be used to generate the required hash

This method throws the following exceptions:

`KSigningException`
: There was an error during the signing process

`KEzSignException`
: There was an internal error, incorrect parameters or other error

`KEzSignConnectException`
: There was an error connecting to the server

Example:

```
byte[] dataToSign = Encoding.ASCII.GetBytes("Data to sign");

byte[] signature = client.signData("SIGNCHANNEL", dataToSign, false);
```

Verify PKCS#7 Signature Methods

The following methods are used to verify a PKCS#7 signature and will perform the required path building and revocation checking as configured at the server

Verifying Signature 1

```
public void verifySignature(String channelName, byte[] signature,
                            byte[] contentBytes, bool dataIsDigest)
```

If `contentBytes` is a hash of the data then `dataIsDigest` must be true, otherwise false

This method throws the following exceptions:

`KVerificationException`
: There was an error during the verification process

`KPathException`
: There was a path building error

`KRevocationException`
: A certificate is revoked or there was an error during the revocation check process

`KEzSignConnectException`
: There was an error connecting to the server

`KEzSignException`
: There was another error

Example:

```
try
{
     client.verifySignature("SIGNCHANNEL", signature, dataToVerify, false);

     Console.Out.WriteLine("Signature verified!");
}
catch (KEzSignException generalEx)
{
     Console.Out.WriteLine("There was an error calling the verify function: " +
                           generalEx.Message);
}
catch (KVerificationException verifyEx)
{
     Console.Out.WriteLine("There was a signature verification error: " +
                           verifyEx.Message);
}
catch (KPathException pathEx)
{
     Console.Out.WriteLine("There was a path building error: " + pathEx.Message);
}
catch (KRevocationException revEx)
{
     Console.Out.WriteLine("There was a revocation check error: " + revEx.Message);
}
```

Verifying Signature 2

This method allows for the by-passing of revocation checking and the by-passing of path building i.e. only a simple verification check will be performed to confirm the data was signed by the certificate specified in the signature and has not been altered

```
public void verifySignature(String channelName, byte[] signature,
                byte[] contentBytes, bool dataIsDigest,
                bool bypassRevocationCheck, bool bypassPathBuild)
```

If the values for `bypassRevocationCheck` and `bypassPathBuild` are both set to `false` the signature will be verified in the same way as the previous method

This method throws the following exceptions:

`KVerificationException`
: There was an error during the verification process

`KPathException`
: There was a path building error

`KRevocationException`
: A certificate is revoked or there was an error during the revocation check process

`KEzSignConnectException`
: There was an error connecting to the server

`KEzSignException`
: There was another error

Verify Raw Signature Methods

The following methods are used to verify Raw (PKCS#1) signatures and will perform the required path building and revocation checking as configured at the server

Verifying Raw Signature 1

Note that as a PKCS#1 signature does not contain the signer certificate, this must be provided

If there are other certificates in the path that are not stored in the channel, use the Verify Raw Signature 2 method. Path building and revocation checking (if configured at the server) will be performed

```
public void verifySignature(String channelName, byte[] signature,
                byte[] contentBytes, bool dataIsDigest,
                X509Certificate signerCert)
```

This method throws the following exceptions:

`KVerificationException`
: There was an error during the verification process

`KPathException`
: There was a path building error

`KRevocationException`
: A certificate is revoked or there was an error during the revocation check process

`KEzSignConnectException`
: There was an error connecting to the server

`KEzSignException`
: There was another error

Verifying Raw Signature 2

This method accepts the signer certificate as well as other certificates in the path. Path building and revocation checking (if configured at the server) will be performed

```
public void verifySignature(String channelName, byte[] signature,
                byte[] contentBytes, bool dataIsDigest,
                X509Certificate signerCert,
                X509Certificate[] otherCerts)
```

This method throws the following exceptions:

`KVerificationException`
: There was an error during the verification process

`KPathException`
: There was a path building error

`KRevocationException`
: A certificate is revoked or there was an error during the revocation check process

`KEzSignConnectException`
: There was an error connecting to the server

`KEzSignException`
: There was another error

Verifying Raw Signature 3

Use this method if you wish to bypass revocation checking (whether configured at the server or not) and/or path building

```
public void verifySignature(String channelName, byte[] signature,
                byte[] contentBytes, boolean dataIsDigest,
                X509Certificate signerCert,
                X509Certificate[] otherCerts,
                bool bypassRevocationCheck, bool bypassPathBuild)
```

This method throws the following exceptions:

`KVerificationException`
: There was an error during the verification process

`KPathException`
: There was a path building error

`KRevocationException`
: A certificate is revoked or there was an error during the revocation check process

`KEzSignConnectException`
: There was an error connecting to the server

`KEzSignException`
: There was another error

Note:
`otherCerts` can be `null` if you do not wish to specify any other certificates in the path

If you wish to perform just the signature verification operation. Set `bypassRevocationCheck` and `bypassPathBuild` both to `true` and `otherCerts` to `null`

Generate Random Number Methods

The following method is called to generate random data:

```
public byte[] generateRandomBytes(String channelName, int numBytes)
```

The number of random bytes selected will be returned

This method throws the following exceptions:

`KEzSignConnectException`
: There was an error connecting to the server

`KEzSignException`
: There was another error

Encrypt/Decrypt Methods

These methods provide encryption and decryption using AES keys previously generated on the server. The algorithm used is AES with CBC (Cipher Block Chaining) and PKCS#5 padding. A random IV (Initialisation Vector) is created every time data is encrypted and this IV is placed in the first 16 bytes of the returned data, with the remaining bytes being the encrypted data itself

The following method is called to encrypt data:

```
public byte[] encryptData(String channelName, byte[] dataToEncrypt,
                          String keyLabel)
```

This will encrypt the clear data contained in `dataToEncrypt` using the key referenced by `keyLabel` and return the encrypted data

`keyLabel` must refer to a key which has previously been generated on the server using the management utility. If the key does not exist `KEncipherException` will be thrown

This method throws the following exceptions:

`KEncipherException`
: There was an error whilst encrypting the data

`KEzSignConnectException`
: There was an error connecting to the server

`KEzSignException`
: There was another error

The following method is called to decrypt previously encrypted data:

```
public byte[] decryptData(String channelName, byte[] encryptedData,
                          String keyLabel)
```

This will decrypt data previously encrypted with the `encryptData` method using the key referenced by `keyLabel` and return the clear data

keyLabel must refer to a key which has previously been generated on the server using the management utility. If the key does not exist `KEncipherException` will be thrown

This method throws the following exceptions:

`KEncipherException`
: There was an error whilst decrypting the data

`KEzSignConnectException`
: There was an error connecting to the server

`KEzSignException`
: There was another error

Example:

```
try
{
     byte[] originalClearData = Encoding.ASCII.GetBytes("Encrypt this");

     byte[] encryptedData = client.encryptData("ENCRYPTCHAN", originalClearData,
                                                "encryptkey1");
     byte[] clearData = client.decryptData("ENCRYPTCHAN", encryptedData,
                                           "encryptkey1");
}
catch (KEzSignException generalEx)
{
     Console.Out.WriteLine("There was an error calling the EzSign server: " +
                                generalEx.Message);
}
catch (KEncipherException encipherEx)
{
     Console.Out.WriteLine("There was an error encrypting/decrypting data: " +
                                encipherEx.Message);
}
```

Client Utils API

The Client Utilis API calls are included in the `KHash` class

This class is contained in the `KEzSignClientUtils.dll` file and held in the `com.krestfield.ezsign.client.utils` namespace

KHash

This class contains the following methods for the generation of hashes:

`public KHash(String hash)`

The constructor can be passed the hash algorithm as a string. Accepted values are `SHA-1`, `SHA-256`, `SHA-384` and `SHA-512`

`public byte[] digest(byte[] data)`

This method calculates the hash value over all the data provided and returns the hash value

`public void update(byte[] data)`

Sections or slices of data can be provided as received or processed and will be included in the overall hash generated

`public void doFinal(byte[] data)`

If the update method has been called, when this method is called together with the last piece of data to be hashed, the hash over all the data will be calculated and returned

`public void reset()`

This resets the internal buffer allowing the same object to be used to calculate a hash over fresh data

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 at

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