eHealth Framework

How to configure an Assembly for Encryption

Imprint

InterComponentWare AG
Altrottstraße 31
69190 Walldorf
Tel.: +49 (0) 6227 385 0
Fax.: +49 (0) 6227 385 199

Document ID:
Document version: 0.3
Document Language: en (US)
Security Level: Public
Document Status: Approved
Product Name: eHealth Framework
Product Version: 2.9
Last Change: 23.07.2010

Document Version History

Version	Date	Name	Sections Changed	Change Description
0.01	14 .07.2010	YDI	All	Started content.
0.02	22.07.2010	KKL	All	Functional review.
0.03	23.07.2010	SGR	All	Editorial review.
0.04	12.08.2010	YDI	All	Functional review: BAS-2847, BAS-2581

Contents

1 Overview

The eHealth Framework (eHF) provides an infrastructure for pseudonymization and encryption of sensitive health data at application level. Sensitive data are encrypted in the application before going to the database. This document describes in detail the steps in your assembly to enable pseudonymization and encryption.

The eHF encryption infrastructure utilizes an application specific target mapping file which specifies how sensitive data are encrypted. Whenever the target mapping file changes, either due to changes of cryptographic parameters or as a consequence of changes in the domain model (i.e. new encryption-relevant data added, or classification of data changed), configurations in your assembly must be adapted accordingly. The required steps are described in chapter "Upgrade the assembly configuration" of the "How to Upgrade a Database for Encryption" document.

2 Tasks

The subsequent tasks guide you through the process for configuring your assembly to enable encryption and pseudonymization.

2.1 Specifying additional Dependencies

An assembly aggregates modules that are defined as dependencies of the assembly project. In order to enable encryption, additional dependencies have to be added.

1. Include the dependency to eHF Commons Encryption, eHF Encryption, and eHF Key Tools in `project.xml`.

```
<dependency>
    <groupId>ehf</groupId>
    <artifactId>ehf-commons-encryption-runtime</artifactId>
    <version>2.10</version>
    <properties>
        <war.bundle>true</war.bundle>
        <category>lib</category>
    </properties>
    <type>jar</type>
</dependency>

<dependency>
    <groupId>ehf</groupId>
    <artifactId>ehf-commons-encryption-doc</artifactId>
    <version>2.10</version>
    <properties>
        <war.bundle>false</war.bundle>
        <ehf-artifact>documentation</ehf-artifact>
        <ehf-documentation-path>
            modules/commons-encryption
        </ehf-documentation-path>
    </properties>
    <type>jar</type>
</dependency>

<dependency>
    <groupId>ehf</groupId>
    <artifactId>ehf-encryption-config</artifactId>
    <version>2.10</version>
    <properties>
        <ehf-module>encryption</ehf-module>
    </properties>
    <type>jar</type>
</dependency>

<dependency>
    <groupId>ehf</groupId>
    <artifactId>ehf-encryption-runtime</artifactId>
    <version>2.10</version>
    <properties>
        <war.bundle>true</war.bundle>
        <category>lib</category>
    </properties>
    <type>jar</type>
</dependency>

<dependency>
    <groupId>ehf</groupId>
    <artifactId>ehf-encryption-doc</artifactId>
    <version>2.10</version>
    <properties>
        <war.bundle>false</war.bundle>
        <ehf-artifact>documentation</ehf-artifact>
        <ehf-documentation-path>
            modules/encryption
        </ehf-documentation-path>
    </properties>
    <type>jar</type>
</dependency>

<dependency>
    <groupId>ehf</groupId>
    <artifactId>ehf-key-tools</artifactId>
    <version>2.10</version>
    <properties>
        <war.bundle>true</war.bundle>
        <category>lib</category>
         <release>ant</release>
         <releaseIncludePattern>**/*</releaseIncludePattern>
         <releaseExcludePattern/>
         <releaseTarget>install</releaseTarget>
    </properties>
    <type>jar</type>
</dependency>
```

2. Additionally, include the dependency to the third-party libraries `colt` and `bouncycastle`, which are required by the order preserving encryption algorithm of the encryption infrastructure.

```
<dependency>
    <groupId>colt</groupId>
    <artifactId>colt</artifactId>
    <version>1.2.0-icw-p1</version>
    <type>jar</type>
    <properties>
        <war.bundle>true</war.bundle>
    </properties>
</dependency>

<dependency>
  <groupId>bouncycastle</groupId>
  <artifactId>bcprov-jdk15</artifactId>
  <version>145</version>
  <type>jar</type>
  <properties>
        <war.bundle>true</war.bundle>
    </properties>
</dependency>
<dependency>
  <groupId>bouncycastle</groupId>
  <artifactId>bcmail-jdk15</artifactId>
  <version>145</version>
  <type>jar</type>
</dependency>
```

3. Mark dependencies for inclusion in the distribution lib folder.
A quite significant list of dependencies have to be marked with the property

```
<properties>
    <category>lib</category>
</properties>
```

as these are required during the initialization phase described in a later step. Please ensure that the following list of dependencies carries the category assignment (versions may differ):

```
ant-1.7.0.jar
aspectjrt-1.6.3.jar
aspectjtools-1.6.3.jar
```

```
camel-spring-2.0.0.jar
commons-codec-1.3.jar
commons-io-1.4.jar
commons-logging-1.1.1.jar
ehf-build-tools-2.10.jar
ehf-commons-encryption-runtime-2.10.jar
ehf-commons-runtime-2.10.jar
ehf-encryption-runtime-2.10.jar
ehf-key-tools-2.10.jar
ehf-migration-tools-2.10.jar
hsqldb-1.8.0.7.jar
jdom-1.1.jar
log4j-1.2.15.jar
servletapi-2.4.jar
spring-2.5.6.jar
xpp3_min-1.1.4c.jar
xstream-1.2.2.jar
```

If you experience missing classes during the initialization phase when installing the distributable artifact (release zip), you may require additional artifacts in the lib folder. This heavily depends on the demands of your application's initialization phase.

2.2 Configuring the Properties for Encryption

The encryption infrastructure can be configured using a set of properties which are defined in the property file `ehf-encryption.properties` in the directory `src/main/config` of the assembly project.

```
encryption.connection.driver=@@encryption.connection.driver@@
encryption.connection.url=@@encryption.connection.url@@
encryption.connection.username=@@encryption.connection.username@@
encryption.connection.password=@@encryption.connection.password@@
encryption.connection.validation.query=@@encryption.connection.validation.query@@
encryption.connection.dialect=@@encryption.connection.dialect@@
encryption.connection.schema.prefix=@@connection.schema.prefix@@
encryption.connection.schema.suffix=@@connection.schema.suffix@@
encryption.connection.jdbc.batch.size=@@encryption.connection.jdbc.batch.size@@

encryption.targetmapping.name=@@encryption.targetmapping.name@@
encryption.targetmapping.path=classpath:/META-INF/@@encryption.targetmapping.
name@@

encryption.keystore.path=keystore.keys
encryption.receipt.cache=@@encryption.receipt.cache@@
encryption.keyprovider.keyencryption.strategy=@@encryption.keyprovider.
keyencryption.strategy@@
encryption.keyprovider.keyencryption.keylength=@@encryption.keyprovider.
keyencryption.keylength@@
```

1. eHF Encryption can use a separate data source other than the system-wide data source. The data source is specified by the `encryption.connection.*` properties in `configuration.properties` and/or `configuration/configuration.deployment.properties` of your assembly.

2. The name of the target mapping file (relative to `classpath:/META-INF`) is specified by the `encryption.targetmapping.name` property. Specify it in `assembly.configuration.properties` of your assembly, for example:

   ```
   encryption.targetmapping.name=ehf-target-mapping.xml
   ```

   If you want to use a null-cipher (disabled encryption) configuration for the development and testing phase, specify the `encryption.targetmapping.name` property as follows:

```
encryption.targetmapping.name=ehf-target-mapping-nullcipher.xml
```

3. The name of the file storing the cryptographic keys is specified by the `encryption.keystore.path` property.
4. The `encryption.receipt.cache` property specifies whether a second-level database cache should be used for cryptographic parameters. It is recommended to enable caching for performance considerations. Specify it in the `configuration/configuration.product.instance.properties` file of your assembly, for example:

   ```
   encryption.receipt.cache=true
   ```
5. The `encryption.keyprovider.keyencryption.strategy` property specifies the strategy for protecting the master keys acquired from the Master Key Provider. The master keys are encrypted either with an `asymmetric` (for instance, the RSA algorithm) or with a `hybrid` cryptographic algorithm. Due to its flexibility regarding the allowed length of plaintexts (e.g. the length of master keys), a hybrid algorithm is preferred over an asymmetric algorithm. Specify the `encryption.keyprovider.keyencryption.strategy` property in the `configuration/configuration.product.instance.properties` file of your assembly, for example:

   ```
   encryption.keyprovider.keyencryption.strategy=hybrid
   ```

Note: The key encryption strategy property must be configured in the same way at the Master Key Provider and in your assembly.

---

6. To enhance transport security based on TLS/SSL, master keys are further encrypted with a temporary RSA public key ((which is created randomly at runtime) provided by your application. The `encryption.keyprovider.keyencryption.keylength` property specifies the length of the RSA public key. For compliance with the US NIST and BSI requirements, the length of the RSA key shall be at least 2048 bits. Specify the key length in the `configuration/configuration.product.instance.properties` file of your assembly, for example:

   ```
   encryption.keyprovider.keyencryption.keylength=4096
   ```
7. The context files `ehf-encryption-external-context.xml` and `ehf-encryption-internal-context.xml` declare the Master Key Provider for production-level deployments and test deployments respectively. The property `encryption.context` specifies whether the internal or the external context should be used. This property is specified in the `configuration.properties` and/or `configuration/configuration.product.instance.properties` files. For test deployments, add the following line to the property files:

   ```
   encryption.context=internal
   ```

   For production deployments, add the following line to the property files:

   ```
   encryption.context=external
   ```

2.3 Setting up the Initialization Process

During the initialization phase, an application instance specific key package consisting of cryptographic keys and encryption configurations is generated.

Attention: The key package is very essential for the product. Loosing the key package means also loosing all encrypted data!

The following steps describe how the eHF build and install process can be configured to produce the key package.

1. For the initialization step, specify the key package creation processor and the key package distribution processor by adding a `initialize-keypackage.xml` file with the following content in your assembly directory `src/main/resources/META-INF/assembly`, assuming the name of your assembly is `assembly`. You may use other names of your choice.

```
<?xml version="1.0" encoding="utf-8"?>
<beans [...]>
    <!-- ================================================================ -->
    <!--  The configuration is externalized to a property-file.           -->
    <!--  While this context definition stays within the module runtime,  -->
    <!--  the properties are loaded from the assembly. This enables       -->
    <!--  configuration of this module from the assembly.                 -->
    <!-- ================================================================ -->
    <bean class="org.springframework.beans.factory.config.
PropertyPlaceholderConfigurer">
        <property name="ignoreResourceNotFound" value="false" />
        <property name="ignoreUnresolvablePlaceholders" value="true" />
        <property name="locations">
            <list>
                <value>classpath:META-INF/ehf-encryption.properties</value>
            </list>
        </property>
    </bean>

    <bean id="keyEncryptionConfig" class="com.icw.ehf.commons.encryption.api.
KeyEncryptionConfig">
        <property name="keyEncryptionStrategy" value="${encryption.
keyprovider.keyencryption.strategy}" />
        <property name="keyEncryptionKeyLength" value="${encryption.
keyprovider.keyencryption.keylength}" />
    </bean>

    <bean id="keyPackageCreationProcessor" class="com.icw.ehf.keytools.
management.processor.KeyPackageCreationProcessor">
        <property name="keyProvider" ref="masterKeyProvider" />
        <property name="keyEncryptionConfig" ref="keyEncryptionConfig" />
    </bean>

    <ehf:processor id="keyPackageCreation" ref="keyPackageCreationProcessor">
        <ehf:resourceLocation value="classpath:/META-INF"/>
        <ehf:resourceLoader patterns="${encryption.targetmapping.name}"/>
        <ehf:parameter class="com.icw.ehf.keytools.management.processor.
KeyPackageCreationParameter">
            <property name="generatorParameter">
                <bean class="com.icw.ehf.keytools.management.crypto.
KeyPackageGeneratorParameter">
```

```
                    <property name="targetPath" value="." />
                </bean>
            </property>
        </ehf:parameter>
    </ehf:processor>

    <bean id="keyPackageDistributionProcessor" class="com.icw.ehf.keytools.
management.processor.KeyPackageDistributionProcessor"/>

    <ehf:processor id="keyPackageDistribution" ref=
"keyPackageDistributionProcessor">
        <ehf:resourceLocation value="classpath:/META-INF"/>
        <ehf:resourceLoader patterns="${encryption.targetmapping.name}"/>
        <ehf:parameter class="com.icw.ehf.keytools.management.processor.
KeyPackageDistributionParameter">
            <property name="keyPackagePath" value="keypackage"/>
        </ehf:parameter>
    </ehf:processor>
</beans>
```

Both processors take a target mapping file as input. The key package creation processor utilizes a Master Key Provider. Using the property `targetPath`, you can specify the location of the produced key package. In the above example, the package is located in the root directory of the assembly. For the distribution processor, the property `keyPackagePath` shall reference to the resulting key package.

2. A set of marker files are used to define the target locations for distributing the individual elements of the key package.
   - Add an empty `.keystore` file for the keystore to the directory `src/main/resources/META-INF`.
   - Add an empty `.targetmapping` file for the target mapping file to the directory `src/main/resources/META-INF`.
   - Add an empty `.mac` for the signed target mapping file and the signed keystore to the directory `src/main/resources/META-INF`.
   - Add an empty `.keyreferences` file for the key references to the directory `src/main/resources/META-INF/assembly/bootstrap`. Key references will be loaded into the encryption database during the bootstrap phase.

3. To add the initialization step to the initialization process of your assembly, specify the `src/main/resources/META-INF/assembly-initialize.xml` file with the following content:

```
<?xml version="1.0" encoding="utf-8"?>
<beans [...]>
    <import resource=
        "classpath:/META-INF/assembly/initialization/
            ehf-encryption-@@encryption.context@@-context.xml"/>
    <import resource=
        "classpath:/META-INF/assembly/initialize-keypackage.xml"/>

</beans>
```

4. Add the following two lines both to `configuration.properties` and `configuration.product.properites`:

```
configure.initialize.pattern=classpath:/META-INF/assembly-initialize.xml
configure.initialize.policy=assembly/deploy.policy
```

2.4 Setting up the Bootstrap Process

During the bootstrap phase, elements of the key package are imported into the encryption database. The following steps describe the required configuration.

1. Specify the import processor for key references by adding a `bootstrap-encryption-keyreferences.xml` file with the following content in your assembly directory, e.g. `src/main/resources/META-INF/assembly`. The import processor takes the `key-references.xml` file which is part of the key package as input.

```
<?xml version="1.0" encoding="utf-8"?>
<beans [...]>
 <ehf:processor id="encryptionKeyReferenceImport" order="0" ref=
"encryptionKeyReferenceImportProcessor">
    <ehf:resourceLocation value="classpath:/META-INF/assembly/bootstrap" />
    <ehf:resourceLoader patterns="**/key-references.xml" />
 </ehf:processor>

 <bean id="keyStore" class="com.icw.ehf.commons.encryption.impl.
KeyStoreFile">
    <constructor-arg index="0" value="JCEKS"/>
    <constructor-arg index="1" value="META-INF/${encryption.keystore.path}" /
>
  </bean>
</beans>
```

2. Add the import processor for key references to the bootstrap process by adding the following line to the `src/main/resources/META-INF/assembly-bootstrap.xml` file.

```
<import resource=
  "classpath:/META-INF/[...]/bootstrap-encryption-keyreference.xml"/>
```

2.5 Extending the Assembly Spring Context

The following steps are necessary to configure the spring context to enable encryption.

1. Ensure that you import the Master Key Provider configuration by including the following line into the list of imports:

```
<import resource=
    "classpath:/META-INF/ehf-encryption-@@encryption.context@@-context.xml"/>
```

2. Ensure an alias, wiring the encryption module `CryptoService` is defined:

```
<alias alias="cryptoService" name="encryptionModuleService" />
```

3. Due to the current way audit events are processed, the following configuration is necessary:

```
<bean id="auditModuleAttributeModulationConfig"
    class="com.icw.ehf.audit.domain.ModuleAttributeModulation" factory-
method="getAttributeModulation">
    <property name="cryptoService" ref="cryptoService"/>
</bean>
```

2.6 Configuring the Master Key Provider

The eHF Encryption utilizes master keys to protect the encryption keys. In particular, a primary master key is a password protecting the keystore, while a secondary master key is used to encrypt the keys inside the keystore. Different providers of master keys are available for test and production-level deployments.

For testing purposes, a local `DummyKeyProvider` which stores the master keys as plaintexts in the configuration file `ehf-encryption-internal-context.xml` can be used.

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

```
<beans [...]>
  <!-- Master Key Provider (for internal use only; development) -->
  <bean id="masterKeyProvider" class="com.icw.ehf.commons.encryption.impl.
DummyKeyProvider">
   <constructor-arg index="0">
     <map>
       <entry key="keystore:primary-key" value="Y2hhbmdlaXQ="/>
       <entry key="keystore:secondary-key" value="rxu+tu4C6l3WB81YEl6wVQ=="/>
     </map>
   </constructor-arg>
  </bean>
</beans>
```

For production-level deployments, a remote Master Key Provider should be used. The Master Key Provider is specified in the configuration file `ehf-encryption-external-context.xml` as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans [...]>
<bean class="org.springframework.beans.factory.config.
PropertyPlaceholderConfigurer">
     <property name="ignoreResourceNotFound" value="false" />
     <property name="ignoreUnresolvablePlaceholders" value="true" />
     <property name="locations">
         <list>
             <value>classpath:META-INF/ehf-keyprovider.properties</value>
         </list>
     </property>
 </bean>

 <!-- master key provider HTTP-based proxy -->
 <bean id="masterKeyProvider" class="com.icw.ehf.commons.encryption.impl.
HttpBasedKeyProviderClient">
   <constructor-arg value ="${keyprovider.requestkey.url}"/>
   <property name="clientStorePass" value="${keyprovider.clientstore.password}" />
   <property name="trustStorePass" value="${keyprovider.truststore.password}" />

   <property name="clientStoreName" value="${keyprovider.clientstore.path}" />
   <property name="trustStoreName" value="${keyprovider.truststore.path}" />
 </bean>
</beans>
```

An `HttpBasedKeyProviderClient` is used to access the remote Master Key Provider over SSL with mutual authentication. To facilitate client authentication (i.e. proving the identity of your application to the Master Key Provider), a client certificate with the associated private key stored in a client keystore are used. To facilitate server authentication (i.e. making sure that your application is communicating with the trusted Master Key Provider), a server certificate stored in a truststore is used.

Perform the following steps to use a remote Master Key Provider:

1. Specify the location of the Master Key Provider using the `keyprovider.requestkey.url` property in the `configuration.deployment.properties` file of your assembly. For example,

   ```
   keyprovider.requestkey.url=https://mkp-trunk.de.icw.int/mkp/requestkey
   ```

2. Specify the properties of your client keystore and truststore in the `configuration.deployment.properties` file of your assembly. For example,

   ```
   keyprovider.clientstore.path=mkp-client-001.keystore
   keyprovider.clientstore.password=changeit
   keyprovider.truststore.path=mkp-client.truststore
   keyprovider.truststore.password=changeit
   ```

   The `keyprovider.clientstore.path` and `keyprovider.truststore.path` properties specify the name of the client keystore and the name of the truststore respectively. The `keyprovider.clientstore.password` and `keyprovider.truststore.password` properties specify the passwords protecting the client keystore and the truststore respectively. For each keystore or truststore, the same password serves as both key password and store password. You should choose a secure password instead of a default password like `changeit`.

3. Acquire a client keystore which stores the client certificate and the associated private key.

   Either you can acquire the client certificate and the associated private key from the Master Key Provider serving as a Certificate Authority (CA), or you can acquire them from an independent CA. In the latter case, you need to pass your client certificate (without the private key) to the Master Key Provider.

4. Acquire a truststore which stores the certificate of the Master Key Provider.
5. Place the client keystore and truststore to the `src/main/resources` directory of your assembly.

2.7 Compliance with ATNA Connection Authentication

The Audit Trail and Node Authentication (ATNA) Integration Profile of IHE (Integrating the Healthcare Enterprise) ↗ mandates the use of the Transport Layer Security (TLS) security negotiation mechanism for all communications between secure nodes. The concrete security requirements for the communication are defined by the ANTA Connection Authentication transaction. Please refer to the "How to Comply with ATNA Connection Authentication" document for detailed information.

In the context of communications between your assembly and a remote Master Key Provider, the ATNA compliance requires a mutual, certificate-based authentication and HTTPS-based connections with the encryption option. The compliance is mainly enforced at the server side by the Master Key Provider. Please refer to the installation manual of the Master Key Provider for detailed configuration information.

At the client side, as described in Configuring the Master Key Provideron page 8, your assembly is configured to utilize client and server certificates, and the HTTPS protocol to invoke services of the Master Key Provider.