icw
eHEALTH FRAMEWORK

## eHealth Framework

# How to Upgrade a Database 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: 0.5
Document Language: en (US)
Security Level: Public
Document Status: draft
Product Name: eHealth Framework
Product Version: 2.9
Last Change: 22.07.2010

# Document Version History

| Version | Date | Name | Sections Changed | Change Description |
|---|---|---|---|---|
| 0.01 | 14.06.2010 | YUD | All | Started content. |
| 0.02 | 02.07.2010 | SGR | all | Reworked document structure; image added |
| 0.03 | 06.07.2010 | YUD | Specify the upgrade tasks | Restrcutured and content added. |
| 0.04 | 08.07.2010 | SGR | Overview | Image updated. |
| 0.05 | 22.07.2010 | SGR | all | Editorial review. |

# Contents

# 1 Overview

The eHealth Framework (eHF) supports both application-level encryption (ALE) and Transparent Data Encryption (TDE) provided by the Oracle database in various upgrade paths as illustrated in Figure 1.

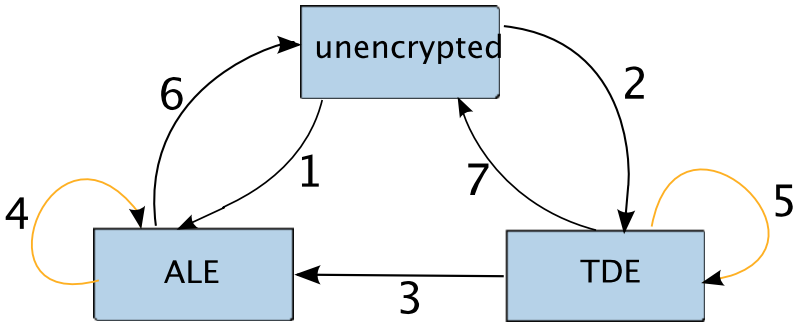


**Figure 1:** Possible upgrade paths for encryption

1. **Upgrade path: Initial ALE enabling**
   Initially enable ALE of an unencrypted database (see ALE Encryption Task on page 7).
2. **Upgrade path: Initial TDE enabling**
   Initially enable TDE of an unencrypted database (see TDE Encryption Task on page 8).
3. **Upgrade path: Switch from TDE to ALE**
   Disable TDE of a TDE-encrypted database (i.e. decrypt with TDE), and subsequently encrypt the database with ALE (see TDE Decryption Task on page 9 and ALE Encryption Task on page 7).
4. **Upgrade path: ALE re-encryption**
   Upgrade an already ALE-encrypted database due to changes in the domain model (e.g. new encryption-relevant attributes added) and/or cryptographic parameters (see ALE Re-encryption Task on page 11, ALE Decryption Task on page 10 and ALE Re-encryption Task for Null Cipher on page 11).
5. **Upgrade path: TDE re-encryption**
   Upgrade an already TDE-encrypted database due to changes in the domain model (e.g. new encryption-relevant attributes added) and/or cryptographic parameters (see TDE Decryption Task on page 9 and TDE Encryption Task on page 8).
6. **Upgrade path: ALE decryption**
   Decrypt one or multiple ALE-encrypted columns due to changes in the domain model (see ALE Decryption Task on page 10).
7. **Upgrade path: TDE decryption**
   Decrypt one or multiple TDE-encrypted columns due to changes in the domain model (see TDE Decryption Task on page 9).

**Note:** In order to switch from an ALE-encrypted to an TDE-encrypted database, you have to compose the upgrade path 6 and 1.

This document describes in detail the required steps for the different upgrade paths in your module (see Upgrade the module configuration on page 3) and in your assembly (see Upgrade the assembly configuration on page 13).

# 2 Upgrade the module configuration

This section describes the required steps in the involved module to perform a upgrade for encryption.

## 2.1 Prepare for the upgrade

This section describes the preparing steps before upgrading the database for encryption.

1. Drop the triggers for history tables, if there are any.

   In eHF, history tables are used to keep the complete history of data changes. Before data in live tables are to be updated or deleted, the current status of data is populated by triggers to the associated history tables. Encryption does not change the semantic of data, thus the action of encryption needs not to be tracked in a history table. After encrypting data in live tables, the upgrade process automatically encrypts the history data in history tables.

---

**Note:** Keep in mind to activate the triggers after the completion of the upgrade process.

---

2. Set up the folder structure.
   Each eHF module consists of a `configuration/database` folder storing database-related configurations. The `database` folder further consists of an `encryption` and `upgrade` sub-folder.
   - The `encryption` folder consists of configurations for an initial TDE enabling. Each sub-folder of `encryption` stores TDE configurations for one eHF release. For instance, the `ehf-2.10` folder is associated to configurations for the eHF 2.10 release.
   - The `upgrade` folder consists of configurations for an initial ALE enabling, switching from TDE to ALE, and modifications for an already encrypted (either ALE or TDE) database. Each sub-folder of `upgrade` stores configurations to upgrade eHF to a higher release. For instance, the `ehf-2.9-2.10` folder is used to upgrade eHF from 2.9 to 2.10.

## 2.2 Specify the upgrade configuration

An upgrade configuration specifies upgrade-relevant attributes of domain objects in your model. Please use the following convention to name your configuration files.

- Use the pattern `encryption-*-tde.xml` for TDE related configurations under the `configuration/database/encryption` folder.
- Use the pattern `encryption-upgrade-*-tde.xml` for TDE related configurations, and `encryption-upgrade-*.xml` for ALE related configurations under the `configuration/database/upgrade` folder.

The wild card stands for your module name. To initially enable TDE for eHF Record Medical in eHF 2.10, for example, place the `encryption-record-medical-tde.xml` file into the sub-folder `ehf-2.10/ehf-record-medical` under `encryption`. To modify TDE configuration in eHF Record Medical from eHF 2.9 to eHF 2.10, place the `encryption-`

`upgrade-record-medical-tde.xml` file into the sub-folder `ehf-2.9-2.10/ehf-record-medical` under `upgrade`.

The following code fragment shows an excerpt of the `encryption-upgrade-record-medical.xml` configuration file for eHF Record Medical.

```
<?xml version="1.0" encoding="UTF-8"?>
    <upgrade-configuration
        xmlns="http://www.intercomponentware.com/schema/ehf-keytools"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.intercomponentware.com/schema/ehf-keytools
            http://www.intercomponentware.com/schema/ehf-keytools ">

        <xModelPath>@@upgrade.path@@/record-medical-xmodel.xml</xModelPath>

        <package name="com.icw.ehf.record.medical.domain">
            <attribute name="scope"/>
            <attribute name="changeDate"/>
            <attribute name="createDate"/>
            <attribute name="creatorId"/>
            <attribute name="changerId"/>
        </package>

        <abstractType name="com.icw.ehf.record.medical.domain.Act">
            <attribute name="text"/>
            <attribute name="dataEntererInstanceIdentifierRoot"/>
            <attribute name="dataEntererInstanceIdentifierExtension"/>
            <attribute name="dataEntererOrganization"/>
            <attribute name="performerInstanceIdentifierRoot"/>
            <attribute name="performerInstanceIdentifierExtension"/>
            <attribute name="performerOrganization"/>
            <attribute name="authorInstanceIdentifierRoot"/>
            <attribute name="authorInstanceIdentifierExtension"/>
            <attribute name="authorOrganization"/>
        </abstractType>

        <type name="com.icw.ehf.record.medical.domain.Allergy">
            <attribute name="valueKey"/>
            <attribute name="valueValue"/>
            <attribute name="adverseReaction"/>
        </type>

        <type name="com.icw.ehf.record.medical.domain.AdverseReaction"
              idColumnName="T_Vaccination_C_Id">
            <holderInfo parentClassAttribute="adverseReactions"
              holderClass="com.icw.ehf.record.medical.domain.helper.
                AdverseReactionHolder"
              parentClass="com.icw.ehf.record.medical.domain.Vaccination"/>
            <attribute name="description"/>
        </type>
        ...
        <tde-configuration algorithm="AES256" salt="false"/>
    </upgrade-configuration>
```

The structure of an upgraded configuration file is specified by the schema `encryption-upgrade-configuration.xsd`, defined in the eHF Key Tools module. Configuration elements are explained in the following.

## 2.2.1 Upgrade configuration elements

This section lists and explains the key elements and their attributes. Especially the attributes

- scopeColumnName
- idColumnName

which can be used in a <package>, <type>, or <abstractType> element will be illustrated after the element descriptions.

**The <xModelPath> element**

For a generator-based module, the associated XModel during the module build phase is generated. An XModel consists of information specified in the domain model (for instance, classification annotation) as well as OR mapping information generated by the generator in an XML format.

Encryption-relevant attributes are defined in terms of Java classes and attributes in an upgrade configuration. However, the upgrade process directly works on the database (e.g. through Spring JDBC support) instead of support by the application. Therefore, the upgrade process requires OR mapping information and classification information to bridge between Java objects and database tables. The upgrade process utilizes either an XModel or annotations in Java classes to retrieve OR mapping information. If an XModel is available, the XModel-based option is to be preferred over the reflection-based option.

The `xModelPath` element specifies the location of a module-specific xModel (e.g., `record-medical-xmodel.xml` for the eHF Record Medical module as shown in the above example). If no `xModelPath` element is specified, the upgrade process has to rely on Java reflection on Java classes.

**The <package> element**

The `package` element defines encryption-relevant attributes that apply to all classes in the given package. Classes and interfaces that are not associated with database tables are automatically ignored by the upgrade process.

**Note:**

The eHF Key Tools module utilizes Java reflection to determine the involved classes in the package. Use `package` elements only if the Jar file of your module is available for the upgrade process.

**The <abstractType> element**

Using the `abstractType` element, you can specify encryption-relevant attributes, which apply to all subclasses (the given class inclusive) of the given (abstract) class or interface. In the example above, the `text` attribute of all subclasses of the abstract class `com.icw.ehf.record.medical.domain.Act` are encryption-relevant.

**Note:**

The eHF Key Tools module utilizes Java reflection to determine the involved subclasses of a given class or interface. Use `abstractType` elements only if the Jar file of your module is available for the upgrade process.

**The <type> element**

Using the `type` element, you can specify encryption-relevant attributes for a particular class.

**The <holderInfo> element of a <type> element**

The `holderInfo` element of a `type` element specifies one-to-many relationship between domain objects. In the example of eHF Record Medical, a vaccination can cause one or multiple adverse reactions. `com.icw.ehf.record.medical.domain.Vaccination` is the parent class of `com.icw.ehf.record.medical.domain.AdverseReaction`. The `adverseReactions` attribute of a vaccination object points to the collection of adverse reactions. The collection of adverse reactions are hold by the holder class `com.icw.ehf.record.medical.domain.helper.AdverseReactioneHolder`. When you specify a `holderInfo` element, the attributes `parentClass`, `parentClassAttribute` and `holderClass` are mandatory.

### The *scopeColumnName* and *idColumnName* attributes

The eHF application-level encryption supports scope-based and instance-based encryption. By default, the upgrade process assumes that the scope of a domain object is stored in the `C_SCOPE` column of the associated database table, while the instance identifier is stored in the `C_ID` column.

Using the `scopeColumnName` or `idColumnName` attribute of a `package`, `abstractType` or a `type` element, you specify custom names for the scope or identifier column respectively. In the following example of eHF Audit, The domain class `AuditEntry` does not have a scope attribute. Hence, it consists of the specification `scopeColumnName="NULL"`. The domain class `AuditEntryAttribute` uses the name `T_Vaccination_C_Id` column for its identifier column.

```
<?xml version="1.0" encoding="UTF-8"?>
<upgrade-configuration     xmlns=...>

    <xModelPath>@@upgrade.path@@/audit-xmodel.xml</xModelPath>

    <excludesFilterName>
        @@upgrade.path@@/ehf-audit/upgrade-audit-encryption-excludes-filter.xml
    </excludesFilterName>

    <type name="com.icw.ehf.audit.domain.AuditEntry"
        scopeColumnName="NULL">
        <attribute name="userId"  />
        <attribute name="objectId" />
        <attribute name="objectScope"/>
    </type>

    <type name="com.icw.ehf.audit.domain.AuditEntryAttribute"
          idColumnName="T_AUDITENTRY_C_ID">

        <holderInfo parentClassAttribute="attributes"
            holderClass="com.icw.ehf.audit.domain.helper.
                AuditEntryAttributeHolder"
            parentClass="com.icw.ehf.audit.domain.AuditEntry" />

        <attribute name="value"/>
    </type>

    <tde-configuration algorithm="AES256" salt="false"
          useHistoryTable="false"/>
</upgrade-configuration>
```

### The <excludesFilterName> element

The `excludesFilterName` element specifies a configuration file consisting of attributes which should be excluded from the parent configuration file. The file that consists of excluded attributes also follows the configuration schema `encryption-upgrade-configuration.xsd`. An `excludesFilterName` element has only effect in the top-level parent configuration file. The above configuration listing shows an example.

**TDE specific configurations**

Cryptographic parameters used for ALE are specified in a target mapping file provided by each eHF-based module. In case of TDE, the `tde-configuration` element is used to specify the `algorithm` and the `salt` property. The property values specified in a `tde-configuration` element apply to all attributes in the configuration file. If no properties are defined, then AES256 is the default algorithm and no salt is used.

The `useHistoryTable` property of a `tde-configuration` element specifies whether the associated history table is encryption-relevant. Its default value is false.

Using the `tde-algorithm`, `tde-salt` and `tde-useHistoryTable` properties of an `attribute` element, you can specify the TDE configuration for a particular attribute as shown in the following example.

```
<type name="com.icw.ehf.record.medical.domain.Allergy">
   <attribute name="valueKey" tde-algorithm="AES256" tde-salt="true"
        tde-useHistoryTable="true"/>

   <attribute name="valueValue" tde-algorithm="AES256" tde-salt="false"
        tde-useHistoryTable="false"/>
   ...
</type>
```

An ALE-related task ignores the `tde-configuration` element as well as `tde-algorithm`, `tde-salt` and `tde-useHistoryTable` attributes.

## 2.3 Specify the upgrade tasks

The eHF Key Tools provides a number of upgrade tasks for the various upgrade scenarios.

For ALE, the following upgrade tasks are available:

- `ALEEncryptionTask`
- `ALEDecryptionTask`
- `ALEReencryptionTask`
- `ALEReencryptionForNullCipherTask`

For TDE, the following tasks are provided:

- `TDEEncryptionTask and`
- `TDEDecryptionTask`

Place the task configuration files under either `configuration/database/encryption/ehf-2.*` or `configuration/database/upgrade/ehf-2.*-2.*` depending on your upgrade path. The following sections describe the configuration of upgrade tasks in detail.

### 2.3.1 ALE Encryption Task

The following example shows the configuration file `encrypt-record-medical.ale` defining an `ALEEncryptionTask`.

```
<migration-configuration>
    <mode>@@upgrade.mode@@</mode>
    <sql-output-filename>
        @@upgrade.path@@/logs/log-sql-output.txt
        </sql-output-filename>
    <log-filename>
        @@upgrade.path@@/logs/log-sql-scripts.txt
    </log-filename>
    <db-connection>
        <driver>@@connection.driver@@</driver>
        <url>@@connection.url@@</url>
        <database>@@database.type@@</database>
```

```
    </db-connection>
    <config-location>
        classpath:/META-INF/ehf-upgrade-context.xml
    </config-location>

    <migration-steps>
        <com.icw.ehf.keytools.upgrade.tasks.ALEEncryptionTask>
            <skipVerificationStep>
                @@upgrade.skip.verification@@
            </skipVerificationStep>
            <batchSize>@@upgrade.batch.size@@</batchSize>
            <concurrency>@@upgrade.concurrency@@</concurrency>
            <configurationFilePath>
                @@upgrade.path@@/ehf-record-medical/encryption-upgrade-
                    record-medical.xml
            </configurationFilePath>
            <xModelPath>
                @@upgrade.path@@/ehf-record-medical/record-medical-xmodel.xml
            </xModelPath>
        </com.icw.ehf.keytools.upgrade.tasks.ALEEncryptionTask>
    </migration-steps>
</migration-configuration>
```

The `configurationFilePath` element refers to the configuration file `encryption-upgrade-record-medical.xml` defined in Specify the upgrade configuration on page 3.

The `xModelPath` element specifies the location of the module-specific XModel.

The `skipVerificationStep`, `batchSize` and `concurrency` elements are used to tune the performance of the upgrade process.

- Internally, an `ALEEncryptionTask` consists of several steps. For instance, after the encryption step, a verification step decrypts the ciphertexts and compares the resulting plaintexts with the original plaintexts. If you want to skip the verification step to minimize database migration time, set the property `@@upgrade.skip.verification@@` to be true in the `configuration.product.instance.properties` file of your assembly. By default, this property is set to false, and a verification step will be executed.
- The upgrade process retrieves multiple rows of the database into memory, encrypts them and batch updates the database with the encrypted values. You can specify the batch size as number of database rows with the `@@upgrade.batch.size@@` property in the `configuration.product.instance.properties` file of your assembly.
- The upgrade process is able to utilize multiple processors running in parallel to boost upgrade performance. You can specify the concurrency as number of concurrent threads with the `@@upgrade.concurrency@@` property in the `configuration.product.instance.properties` file of your assembly.

### 2.3.2 TDE Encryption Task

The following example shows the configuration file `encrypt-record-medical.tde` defining a `TDEEncryptionTask`.

```
<migration-configuration>
    <mode>@@upgrade.mode@@</mode>
    <sql-output-filename>
        @@upgrade.path@@/logs/log-sql-output.txt
    </sql-output-filename>
        <log-filename>@@upgrade.path@@/logs/log-sql-scripts.txt</log-filename>
    <db-connection>
       <driver>@@connection.driver@@</driver>
       <url>@@connection.url@@</url>
       <database>@@database.type@@</database>
    </db-connection>
```

```
    <config-location>
        classpath:/META-INF/ehf-upgrade-context.xml
    </config-location>

    <migration-steps>
        <com.icw.ehf.keytools.upgrade.tasks.TDEEncryptionTask>
            <username>EHF_RECORD_MEDICAL</username>
            <password>@@connection.record-medical.password@@</password>
            <configurationFilePath>
                @@upgrade.path@@/ehf-record-medical/encryption-upgrade-
                    record-medical.xml
            </configurationFilePath>
            <xModelPath>
                @@upgrade.path@@/ehf-record-medical/record-medical-xmodel.xml
            </xModelPath>
            <verificationOutputFileName>
                @@upgrade.path@@/logs/tde-verification-ehf-record-medical.log
            </verificationOutputFileName>
        </com.icw.ehf.keytools.upgrade.tasks.TDEEncryptionTask>
    </migration-steps>
</migration-configuration>
```

The `configurationFilePath` and `xModelPath` elements are defined as in ALE Encryption Task on page 7.

The `username` and `password` elements specify the user name and password to connect to the database.

If specified, the `verificationOutputFileName` element indicates that the upgrade process shall verify the encrypted results. The output file is used internally by the upgrade process to verify the results.

### 2.3.3 TDE Decryption Task

The following example shows the configuration file `decrypt-record-medical.tde` defining a `TDEDecryptionTask`.

```
<migration-configuration>
    <mode>@@upgrade.mode@@</mode>
    <sql-output-filename>
        @@upgrade.path@@/logs/log-sql-output.txt
    </sql-output-filename>
    <log-filename>@@upgrade.path@@/logs/log-sql-scripts.txt</log-filename>
    <db-connection>
        <driver>@@connection.driver@@</driver>
        <url>@@connection.url@@</url>
        <database>@@database.type@@</database>
    </db-connection>
    <config-location>classpath:/META-INF/ehf-upgrade-context.xml</config-location>

    <migration-steps>
        <com.icw.ehf.keytools.upgrade.tasks.TDEDecryptionTask>
            <username>EHF_RECORD_MEDICAL</username>
            <password>@@connection.record-medical.password@@</password>
            <configurationFilePath>
                @@upgrade.path@@/ehf-record-medical/encryption-upgrade-
                    record-medical.xml
            </configurationFilePath>
            <xModelForDecryptionPath>
                @@upgrade.path@@/ehf-record-medical/record-medical-xmodel.xml
            </xModelForDecryptionPath>
        </com.icw.ehf.keytools.upgrade.tasks.TDEDecryptionTask>
    </migration-steps>
</migration-configuration>
```

The `xModelForDecryptionPath` property specifies the location of XModel for decryption.

The `configurationFilePath` property is defined as in ALE Encryption Task on page 7.

In order to decrypt all TDE encrypted columns, neither upgrade configuration file nor XModel for decryption has to be provided. The upgrade process extracts information about TDE encrypted columns from the database itself.

### 2.3.4 ALE Decryption Task

The following example shows the definition of an `ALEDecryptionTask`.

```
<migration-configuration>
    <mode>@@upgrade.mode@@</mode>
    <sql-output-filename>
        @@upgrade.path@@/logs/log-sql-output.txt
    </sql-output-filename>
    <log-filename>@@upgrade.path@@/logs/log-sql-scripts.txt</log-filename>
    <db-connection>
        <driver>@@connection.driver@@</driver>
        <url>@@connection.url@@</url>
        <database>@@database.type@@</database>
    </db-connection>
    <config-location>
        classpath:/META-INF/ehf-encryption-upgrade-context.xml
    </config-location>
    <previous-config-location>
        classpath:/META-INF/ehf-encryption-upgrade-previous-context.xml
    </previous-config-location>

    <migration-steps>
        <com.icw.ehf.keytools.upgrade.tasks.ALEDecryptionTask>
           <concurrency>@@upgrade.concurrency@@</concurrency>
           <batchSize>@@upgrade.batch.size@@</batchSize>
           <configurationFilePath>
                @@upgrade.path@@/ehf-medicine-cabinet/encryption-upgrade-decrypt-
                    record-medical.xml
            </configurationFilePath>
            <xModelForDecryptionPath>
                @@upgrade.previous.path@@/ehf-record-medical/record-medical-
xmodel.xml
            </xModelForDecryptionPath>
         </com.icw.ehf.keytools.upgrade.tasks.ALEDecryptionTask>
    </migration-steps>
</migration-configuration>
```

If the target mapping file has been changed since the last upgrade, the `previous-config-location` element specifies the previous upgrade context which uses the previous target mapping file. If no `previous-config-location` element is specified, the upgrade process assumes no changes in the upgrade context and uses the context specified in the `config-location` element for decryption.

The `xModelForDecryptionPath` property specifies the location to find the XModel for decryption.

**Note:** For decryption, upgrade-relevant attributes can only be specified using `type` elements in the upgrade configuration file specified by the `configurationFilePath` property. Neither `package` nor `abstractType` elements can be used to specify upgrade-relevant attributes for decryption. Since the Jar file of a module may have changed due to changes in the domain model, an `ALEDecryptionTask` cannot rely

on Java reflection to find information about the relevant attributes. It solely relies on the XModel specified by the above `xModelForDecryptionPath` element.

### 2.3.5 ALE Re-encryption Task

The following example shows the definition of an `ALEReencryptionTask`. A re-encryption task at first decrypts the relevant attributes according to the previous upgrade context and XModel, and then encrypts them using the current upgrade context and XModel.

```
<migration-configuration>
    <mode>@@upgrade.mode@@</mode>
    <sql-output-filename>
        @@upgrade.path@@/logs/log-sql-output.txt
    </sql-output-filename>
    <log-filename>@@upgrade.path@@/logs/log-sql-scripts.txt</log-filename>
    <db-connection>
        <driver>@@connection.driver@@</driver>
        <url>@@connection.url@@</url>
        <database>@@database.type@@</database>
    </db-connection>
    <config-location>
        classpath:/META-INF/ehf-encryption-upgrade-context.xml
    </config-location>
    <previous-config-location>
        classpath:/META-INF/ehf-encryption-upgrade-previous-context.xml
    </previous-config-location>

    <migration-steps>
        <com.icw.ehf.keytools.upgrade.tasks.ALEReencryptionTask>
            <concurrency>@@upgrade.concurrency@@</concurrency>
            <batchSize>@@upgrade.batch.size@@</batchSize>
            <configurationFilePath>
                @@upgrade.path@@/ehf-medicine-cabinet/encryption-upgrade-
                reencrypt-record-medical.xml
            </configurationFilePath>
            <xModelPath>
                @@upgrade.path@@/ehf-record-medical/record-medical-
                    xmodel.xml
            </xModelPath>
            <xModelForDecryptionPath>
                @@upgrade.previous.path@@/ehf-record-medical/record-medical-
                    xmodel.xml
            </xModelForDecryptionPath>
        </com.icw.ehf.keytools.upgrade.tasks.ALEReencryptionTask>
    </migration-steps>
</migration-configuration>
```

**Note:** The scope attribute must be included in the upgrade configuration file specified in `configurationFilePath`, whenever a scope-based encryption (i.e. the scope values in plaintext are used to determine the cryptographic keys or initialization vectors) is involved.

### 2.3.6 ALE Re-encryption Task for Null Cipher

A special ALE upgrade task is the `ALEReencryptForNullCipherTask`. Use this task if the following conditions are fulfilled:

1. The involved database table is already ALE encrypted, especially the scope column is encrypted.
2. A previously unencrypted column in this table shall now be encrypted, especially a scope-based encryption (which is configured in the target mapping file) shall be used.

Because of the scope-based encryption, the `ALEReencryptForNullCipherTask` internally at first copies the encrypted scopes to a temporary column, decrypts the scopes, encrypts the column in question with the scopes in plaintext, and then copies back the encrypted scopes.

The following example shows the definition of an `ALEReencryptForNullCipherTask`. It uses the same properties as an `ALEEncryptionTask`.

```
<migration-configuration>
    <mode>@@upgrade.mode@@</mode>
    <sql-output-filename>
        @@upgrade.path@@/logs/log-sql-output.txt
        </sql-output-filename>
    <log-filename>
        @@upgrade.path@@/logs/log-sql-scripts.txt
    </log-filename>
    <db-connection>
        <driver>@@connection.driver@@</driver>
        <url>@@connection.url@@</url>
        <database>@@database.type@@</database>
    </db-connection>
    <config-location>
        classpath:/META-INF/ehf-upgrade-context.xml
    </config-location>

    <migration-steps>
        <com.icw.ehf.keytools.upgrade.tasks.ALEReencryptForNullCipherTask>
            <skipVerificationStep>
                @@upgrade.skip.verification@@
            </skipVerificationStep>
            <batchSize>@@upgrade.batch.size@@</batchSize>
            <concurrency>@@upgrade.concurrency@@</concurrency>
            <configurationFilePath>
                @@upgrade.path@@/ehf-record-medical/encryption-upgrade-
                    record-medical.xml
            </configurationFilePath>
            <xModelPath>
                @@upgrade.path@@/ehf-record-medical/record-medical-xmodel.xml
            </xModelPath>
        </com.icw.ehf.keytools.upgrade.tasks.ALEReencryptForNullCipherTask>
    </migration-steps>
</migration-configuration>
```

**Note:** The scope attribute must be included in the upgrade configuration file specified in `configurationFilePath`, whenever a scope-based encryption is involved. If no scope attribute is included, an `ALEReencryptForNullCipherTask` has the same functionality as an `ALEEncryptionTask`.

# 3 Upgrade the assembly configuration

The section describes the required steps in the assembly to perform an upgrade for encryption.

## 3.1 General assembly configuration steps

1. Make sure the `project.xml` file of your assembly consists of the complete properties of the eHF Key Tools Module.

   The eHF Key Tools Module provides a set of ant tasks which are responsible for processing the upgrade configurations defined in the modules (see Specify the upgrade configuration on page 3). As shown in the following example, with the `release*` properties definitions of these tasks get included into the release.

```
<dependency>
    <groupId>ehf</groupId>
    <artifactId>ehf-key-tools</artifactId>
    <version>SNAPSHOT</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. Make sure the `ehf-encryption-upgrade-context.xml` file defining the application context for the upgrade process is available in the folder `src/main/config` of your assembly.

3. Define the `@@encryption.path@@` property in the `configuration/configuration.product.instance.properties` file of your assembly, if you want to initially enable TDE.
   Define the `@@upgrade.path@@` property in the `configuration/configuration.product.instance.properties` file of your assembly, if you want to upgrade TDE of your database or upgrade with ALE.

4. Define the upgrade tasks in the `configuration/configuration.product.instance.properties` file of your assembly.

   Add the tasks defined in Specify the upgrade tasks on page 7. to `encryption.tasks`, if you want to initially enable TDE. Add the tasks to `upgrade.tasks`, if you want to upgrade TDE of your database or upgrade with ALE.

   The following listing shows an example to at first de-activate TDE encryption using the `decrypt-record-medical.tde` task, and then activate ALE for eHF Record Medical using the `encrypt-record-medical.ale` task. The `upgrade-modules.xml` task performs structural upgrade.

```
upgrade.tasks=\
            local:upgrade-modules.xml,\
            ...

            local:decrypt-record-medical.tde,\
            local:encrypt-record-medical.ale,\
```

```
            ...
```

**Note:** Keep in mind to add the drop triggers task before the encryption upgrade tasks, and add the activate triggers after the encryption upgrade tasks (see Prepare for the upgrade on page 3).

## 3.2 Configurations due to changes in target mapping

The following sections describe the steps which are only required when the target mapping file has been changed.

### 3.2.1 Specify target mapping changes

1. Specify changes of target mapping in a target mapping upgrade file in the folder `src/main/resources/META-INF` of your assembly.

   The following shows an excerpt of the `ehf-target-mapping-upgrade.xml` file for the eHF Medicine Cabinet module.

```
<?xml version="1.0" encoding="UTF-8"?>
<target-mapping-upgrade xmlns="http://www.intercomponentware.com/schema/ehf-
targetmapping-upgrade" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.intercomponentware.com/schema/ehf-
targetmapping-upgrade ../../main/resources/META-INF/target-mapping-upgrade.
xsd ">
  <newTargetMappingPath>classpath:/META-INF/ehf-target-mapping.xml
  </newTargetMappingPath>
  <module module-id="medicineCabinet">
      <add>
            <!-- Introduce new classification and new key pool -->
            <config-element classification="confidential">
               <parameters key-pool="pool_confidential" key-type="id"/>
            </config-element>

            <key-pool id="pool_confidential" minKeys="10"/>
      </add>

      <modify>
          <!-- Increase the number of keys in pool_medical -->
          <key-pool id="pool_medical" minKeys="10"/>
      </modify>
  </module>
</target-mapping-upgrade>
```

   The structure of a target mapping upgrade file is defined by the schema `target-mapping-upgrade.xsd` in the eHF Key Tools module. The `newTargetMappingPath` element specifies the name of the new, current target mapping file in use. The `add` and `modify` elements are explained in the following.

   **Add**

   An `add` element introduces new key pools and configuration elements referring to these key pools.

   **Modify**

   A `modify` element specifies changes to an existing key pool as well as an existing configuration element.

   The target mapping upgrade file is required to update content of the existing key package.

- According to the specification in `add` and `modify` elements, new keys are created and inserted to the existing keystore, and the associated key references are created and persisted to the `key-references-upgrade.xml` file in the key package. This file is used to bootstrap the encryption database with the new keys. Deletion of existing keys is currently not supported.
- A MAC file (for example, `ehf-target-mapping.mac`) consisting of the message authentication code of the new target mapping file (e.g. `ehf-target-mapping.xml`) is created. The previous target mapping file will be renamed.

2. Add the `encryption.targetmapping.upgrade.name` and `encryption.targetmapping.previous.name` properties in the `assembly.configuration.properties` file in the root directory of your assembly.

   An example is shown in the following:

```
# This file defines the difference between the previous and the new target
 mapping file.
encryption.targetmapping.upgrade.name=ehf-target-mapping-upgrade.xml

# This property specifies the name of the previous target mapping file.
encryption.targetmapping.previous.name=ehf-target-mapping-previous.xml
```

3. Define the `encryption.targetmapping.upgrade.name`, `encryption.targetmapping.previous.name` and `encryption.targetmapping.previous.path` property values in the `ehf-encryption.properties` file in the `src/main/config` folder of your assembly. They specify the name of the target mapping upgrade file, the name of the previous, renamed target mapping file and its location respectively.

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

4. Make sure the `ehf-encryption-upgrade-previous-context.xml` file defining the application context using the previous target mapping file is available in the `src/main/config` folder of your assembly.

5. Add the `upgrade.previous.path` property to the `configuration.product.instance.properties` file in the `configuration` folder of your assembly. This property specifies the folder which keeps previous upgrade configurations.

   The following shows an example:

```
upgrade.previous.path=configuration/database/upgrade/ehf-2.9-2.10
```

### 3.2.2 Update the key package

1. Make sure the existing `keyPackage` folder from your previous upgrade is available in the binary folder of your current release (for example, `target/ehf-2.10-bin`).

   Each key package consists of a target mapping file and its MAC file. Consequently, the existing key package must be updated, whenever the target mapping file changes.

   Content of the existing key package will be updated by executing the command `ant -f install/install.xml configure:all`.

2. Define the `keyPackageUpgradeProcessor` and `keyPackageDistributionProcessor` processors which are responsible for updating the content of the existing key package and distributing these modifications to the appropriate locations of your release.

The processors are defined in the following `ehf-upgrade-keypackage.xml` file located in the `src/main/resources/META-INF/ehf-assembly` folder of your assembly.

```
<?xml version="1.0" encoding="utf-8"?>
<beans xmlns=...
    <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="keyPackageUpgradeProcessor"
         class="com.icw.ehf.keytools.management.processor.
KeyPackageUpgradeProcessor">
        <property name="keyProvider" ref="masterKeyProvider" />
        <property name="keyEncryptionConfig" ref="keyEncryptionConfig" />
    </bean>

    <ehf:processor id="keyPackageUpgrade" ref="keyPackageUpgradeProcessor">
        <ehf:resourceLocation value="classpath:/META-INF"/>
        <ehf:resourceLoader patterns="${encryption.targetmapping.upgrade.
name}"/>
        <ehf:parameter class="com.icw.ehf.keytools.management.processor.
            KeyPackageCreationParameter">
            <property name="generatorParameter">
                <bean class="com.icw.ehf.keytools.management.processor.
                    KeyPackageUpgradeParameter">
                    <property name="previousTargetMappingName"
                              value="${encryption.targetmapping.previous.
                                name}" />
                    <property name="upgradedTargetMappingName"
                              value="${encryption.targetmapping.name}" />
                    <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">
    <!-- Note, neither value of resourceLocation nor patterns of
 resourceLoader is used here.-->
        <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>
```

3. Integrate key package update to the initialization phase.

   The `ehf-assembly-initialize.xml` file in the `src/main/resources` folder of your assembly is processed during the initialization phase as part of the `configure:all` ant task. Import the `ehf-upgrade-keypackage.xml` defined above to trigger upgrading the existing key package.

```
<?xml version="1.0" encoding="utf-8"?>
<beans xmlns=[...]>
    <!-- include processors dedicated to initialization step -->
    <import resource="classpath:/META-INF/ehf-encryption-@@encryption.
context@@-context.xml"/>

    <!-- Initialize the keypackage  -->
<!--    <import resource="classpath:/META-INF/ehf-assembly/ehf-initialize-
keypackage.xml"/>-->

    <!-- Upgrade an existing keypackage -->
    <import resource="classpath:/META-INF/ehf-assembly/ehf-upgrade-
keypackage.xml"/>
</beans>
```

   An updated key package consists of the following files: `keystore.keys`, `keystore.mac`, `key-references.xml`, `key-references-upgrade.xml`, `ehf-target-mapping-previous.xml`, `ehf-target-mapping-previous.mac`, `ehf-target-mapping.xml` and `ehf-target-mapping.mac`.

### 3.2.3 Bootstrap new keys

If changes in the target mapping file introduce new keys, the `key-references-upgrade.xml` file contains the manifest of these new keys, and the following steps are required to bootstrap the new keys. If changes in the target mapping file do not introduce new keys, you can skip the following steps.

1. Define the `encryptionKeyReferenceImport` processor which is responsible for bootstrapping the encryption database with new keys specified in the `key-references-upgrade.xml` file.

   The processor is defined in the `ehf-bootstrap-encryption-keyreference-upgrade.xml` file in the `src/main/resources/ehf-assembly` folder of your assembly. Note, the resource pattern refers to the `key-references-upgrade.xml` file.

```
<?xml version="1.0" encoding="utf-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:ehf="http://www.intercomponentware.com/schema/ehf-commons"

    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.intercomponentware.com/schema/ehf-commons
        http://www.intercomponentware.com/schema/ehf-commons/
            ehf-commons.xsd">

    <ehf:processor id="encryptionKeyReferenceImport"
                   order="0"
                   ref="encryptionKeyReferenceImportProcessor">

        <ehf:resourceLocation value="classpath:/META-INF/ehf-assembly/
            bootstrap" />
        <ehf:resourceLoader patterns="**/key-references-upgrade.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/keystore.keys" />
    </bean>
</beans>
```

2. Define the following `ehf-upgrade-bootstrap-phase-upgrade-keyreferences.xml` file in the folder `src/main/resources/ehf-assembly` which refers to the above key references import processor.

```
<?xml version="1.0" encoding="utf-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:p="http://www.springframework.org/schema/p"
        xmlns:ehf="http://www.intercomponentware.com/schema/ehf-commons"

        xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans.xsd
            http://www.intercomponentware.com/schema/ehf-commons
            http://www.intercomponentware.com/schema/ehf-commons/
                ehf-commons.xsd">

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

        <import resource="classpath:/META-INF/ehf-assembly/ehf-bootstrap-
            encryption-keyreference-upgrade.xml"/>
</beans>
```

3. Define the following `upgrade-bootstrap-phase-02a.xml` file in your upgrade path (for example, the `configuration\database\upgrade\ehf-2.9-2.10` folder) to integrate bootstrapping of new keys as a migration step.

```
<migration-configuration>
    <mode>@@upgrade.mode@@</mode>
    <sql-output-filename>
        @@upgrade.path@@/logs/log-sql-output.txt
    </sql-output-filename>
    <log-filename>@@upgrade.path@@/logs/log-sql-scripts.txt</log-filename>
    <db-connection>
        <driver>@@connection.driver@@</driver>
        <url>@@connection.url@@</url>
        <database>@@database.type@@</database>
    </db-connection>

    <config-location>
        classpath:/META-INF/ehf-assembly/ehf-upgrade-bootstrap-phase-
            upgrade-keyreferences.xml
    </config-location>

    <migration-steps></migration-steps>
</migration-configuration>
```

4. Add the upgrade tasks to the list of upgrade tasks in the `configuration/configuration.product.instance.properties` file of your assembly as shown in the following example.

```
upgrade.tasks=\
    local:upgrade-keypackage.xml,\
    local:upgrade-modules.xml,\
    local:upgrade-bootstrap-phase-02a.xml,\
    ...
```

# 4 Multi-Schema Support

The eHF multi-schema support feature allows using a single database instance for multiple eHF-based applications. In order to separate data of different applications, the same module has different schema names in different applications.

In eHF, each persistent module has a default module schema name. To enable multi-schema support, unique, application-specific prefixes and suffixes are added to the default module schema name. For instance, the default schema name of the eHF Medicine Cabinet module becomes `PRE_EHF_MEDICINE_CABINET_SUF`. For details about naming conventions for schema prefixes and suffixes, please refer to the document "How to enable Multi-Schema Support".

The following describes the steps to enable multi-schema support for the upgrade process.

1. Specify the `connection.schema.prefix` and `connection.schema.suffix` properties for schema prefix and suffix respectively in the `configuration/configuration.deployment.properties` file in the root directory of your assembly. An example is shown in the following.

```
connection.schema.prefix=PRE
connection.schema.suffix=SUF
```

**Note:** The build process automatically appends _ after the specified prefix and prepends _ before the specified suffix. Hence, the prefix and suffix properties shall not end or start with _.

# 5 Troubleshooting

The following sections cover some frequently asked questions and pitfalls.

## 5.1 Property Tokens are not replaced

*The property tokens in my ALE and TDE task configuration files were not replaced after invoking `ant -f install\install.xml upgrade:configure` in your release folder.*

The eHF Build Tools are responsible for replacing the property tokens. However, it supports only files with known suffix. We use the suffix `.ale` for an ALE task configuration file and the suffix `.tde` for a TDE task configuration file. Please make sure that your task configuration files have the suffix `.ale`, `.tde` or `.xml`.