eHealth Framework

# How to enable 3rd level Support for an encrypted Database

# Imprint

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

Document version: preliminary
Document Language: en (US)
Product Name: eHealth Framework
Product Version: 2.10.3
Last Change: 23.03.2010
Editorial Staff: BAS Technology

# Notice

**Note:**

The current version of this document has a draft status and various chapters are still in review.

The document is collaboratively built with the use of the Darwin-Information-Typing-Architecture (DITA) and has therefore a draft status concerning styles and layout. The necessary adaptations are currently also in a developmental stage.

# Contents

# 1 Overview

A 3rd-level support agent usually executes database scripts to analyze and solve problems in the database. Once the database is encrypted, database scripts with plaintext parameters will no more succeed to find the desired results. This document describes the eHF Support Process, which intercepts invocations of database scripts, and encrypts relevant parameters in the scripts before passing them to the database. Upon receiving results from the database, it decrypts relevant results, if required. The support proocess utilizes the eHF Encryption Infrastructure to perform encryption and decryption.

# 2 User's Guide

The following describes the required steps for the support agent to prepare database scripts to run with the support process against an encrypted database, and to retrieve the returned results.

## 2.1 Specify Support Configurations

1. Provide SQL scripts to execute against the database. An SQL script contains a series of SQL commands that are executed, one by one, when the script is invoked. In order to allow the Support Process to decide whether a parameter or a result is relevant for encryption or decryption, SQL commands must be enriched with certain metadata.

   The following shows a number of SQL scripts to explain their usage.

```
<?xml version="1.0" encoding="UTF-8"?>
<sql-commands xmlns="http://www.intercomponentware.com/schema/
ehf-supporttools" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.intercomponentware.com/schema/ehf-
supporttools ../../main/resources/META-INF/encryption-sql-support.xsd ">
<sql-commands>
  <sql-command>
    <sql-query  name="query_intakeEntry">
         <parameters>
            <parameter name="c_scope"
                column="EHF_MEDICINE_CABINET.T_INTAKEENTRY.C_SCOPE"/>
            <parameter name="c_amount"
                column="EHF_MEDICINE_CABINET.T_INTAKEENTRY.C_AMOUNT"/>
         </parameters>
         <parameterValues>
           <value name="c_scope" value="test_scope_1"/>
           <value name="c_amount" value="1"/>
         </parameterValues>
         <returns>
             <return name="C_ID"
                table="EHF_MEDICINE_CABINET.T_INTAKEENTRY"/>
             <return name="C_SCOPE"
                table="EHF_MEDICINE_CABINET.T_INTAKEENTRY"/>
             <return name="C_Amount"
                table="EHF_MEDICINE_CABINET.T_INTAKEENTRY"/>
         </returns>
         <sql-statement>
            select C_ID, C_SCOPE, C_Amount
                from EHF_MEDICINE_CABINET.T_INTAKEENTRY
                where C_SCOPE = :c_scope and C_AMOUNT = :c_amount
         </sql-statement>
    </sql-query>
  </sql-command>

  <sql-command>
    <sql-query  name="query_medicine_for_scope_list">
        <parameters>
             <parameter name="c_scope"
                column="EHF_MEDICINE_CABINET.T_MEDICINE.C_SCOPE"/>
        </parameters>
        <parameterValues>
         <value name="c_scope">
                <valueList>
                    <value>test_scope_1</value>
                    <value>test_scope_2</value>
                </valueList>
            </value>
        </parameterValues>
        <returns>
             <return name="*" table="EHF_MEDICINE_CABINET.T_MEDICINE"/>
        </returns>
        <sql-statement>
           select * from EHF_MEDICINE_CABINET.T_MEDICINE where
            C_SCOPE in (:c_scope)
        </sql-statement>
    </sql-query>
  </sql-command>

  <sql-command>
        <sql-query>
             <returns>
                 <return returnInt="true" />
             </returns>
             <sql-statement>
               select count(*) from EHF_MEDICINE_CABINET.T_MEDICINE
             </sql-statement>
        </sql-query>
  </sql-command>

  <sql-command>
    <sql-procedure  name="read_medicine"
        schemaName="EHF_MEDICINE_CABINET">
        <parameters>
             <parameter name="in_c_scope"
                column="EHF_MEDICINE_CABINET.T_MEDICINE.C_SCOPE"/>
             <parameter name="in_c_id"
                column="EHF_MEDICINE_CABINET.T_MEDICINE.C_ID"/>
        </parameters>
        <parameterValues>
            <value name="in_c_scope" value="test_scope_1">
            </value>
            <value name="in_c_id"
                value="86ed8f27-40c2-4aba-9636-92ace90a9fe7"></value>
        </parameterValues>
        <returns>
             <return name="c_name" alias="out_c_name"
                table="EHF_MEDICINE_CABINET.T_MEDICINE"/>
             <return name="c_id" alias="out_c_id"
                table="EHF_MEDICINE_CABINET.T_MEDICINE"/>
        </returns>
        <sql-statement>
        CREATE or replace PROCEDURE EHF_MEDICINE_CABINET.read_medicine
            (in_c_scope VARCHAR2,
             in_c_id VARCHAR2,
             out_c_name out VARCHAR2,
             out_c_id out VARCHAR2
            )
        IS
           BEGIN
               select C_NAME, C_ID into out_c_name, out_c_id from
                EHF_MEDICINE_CABINET.T_MEDICINE where
                C_SCOPE = in_c_scope and C_ID = in_c_id;
           END;
        </sql-statement>
    </sql-procedure>
  </sql-command>
</sql-commands>
```

The structure of SQL scripts for the support process is specified by the schema `encryption-sql-support.xsd` defined in the eHF Key Tools module. An SQL command can be an SQL query, SQL update (including insert, update and delete operations) or a PL/SQL stored procedure, which are represented by the `sql-query`, `sql-update`, and `sql-procedure` element in an SQL script respectively.

**SQL statements**

Each SQL command element has an `sql-statement` element. An `sql-statement` element consists of either an SQL statement or a PL/SQL block. Note, a PL/SQL procedure must end with a semicolon, while an SQL query or update never ends with a semicolon.

SQL queries and SQL updates use named parameters (for example `:c_scope` and `:c_amount`) as placeholders for actual parameter values. PL/SQL procedures use input parameters of the procedure to specify parameter values (for example, `in_c_scope` and `in_c_id` in the `read_medicine` procedure).

In order to decide whether a parameter in an SQL command is encryption relevant, the support process needs to know the database column referenced by this parameter. For example, the parameter `c_scope` refers to the `EHF_MEDICINE_CABINET.T_MEDICINE.C_SCOPE` column. Such information is available when parsing the SQL statement. However, developing a sophisticated SQL or PL/SQL parser is not a trivial task. Hence, in addition to an `sql-statement` element, an SQL command must consist of metadata which explicitly specify the connection between parameters and referenced database columns. Metadata is provided in the elements `parameters` and `returns` of an SQL command.

**Parameters**

The `parameters` element consists of a set of `parameter` elements, each of which specifies exactly one parameter. The `name` attribute defines the name of the parameter (e.g., it is the named parameter without the colon sign). The `column` attributes defines the fully qualified name of the associated database column. The `update` attribute shall be set to true, if the associated database column will be updated by the SQL statement. If not set, the `update` attribute is set to be false.

**Return values**

The `returns` element consists of a set of `return` elements. The `name` attribute of a `return` element specifies the column name (without schema and table prefix). To refer to all columns of a table, a wildcard can be specified in the `name` attribute. The fully qualified table name is specified in the `table` attribute. In a PL/SQL procedure, the `alias` attribute of a `return` element refers to the output parameter of the procedure. In our example, the output parameter `out_c_name` of the procedure `read_medicine` refers to values of the `c_name` column from table `EHF_MEDICINE_CABINET.T_MEDICINE`.

If an SQL query does not return one or multiple rows but an Integer (for example, the number of selected rows), the `returnInt` attribute of a `return` element shall set to be true. By default, `returnInt` is set to be false.

Note, the current implementation of the support process cannot handle CLOBs and BLOBs as return values.

**Parameter values**

Parameter values are specified in the `parameterValues` element of an SQL command. The element consists of a set of `value` elements, each of which specifies the value for a particular parameter. The `name` attribute of a `value` element specifies the parameter name as defined in the corresponding `parameter` element. The `value` attribute specifies the actual value. If the value is a list, a `valueList` element can be used instead of the `value` attribute. The `query_medicine_for_scope_list` command in our example shows such a case.

**SQL command name**

Each `sql-query`, `sql-update` or `sql-procedure` element has a `name` attribute which specifies the name of the SQL command.

For an `sql-procedure`, the `name` attribute is mandatory. Additionally, the mandatory `schemaName` attribute specifies the schema which holds the procedure. The name of a procedure defined in PL/SQL in an `sql-statement` element is a concatenation of schema name and command name.

For `sql-query` and `sql-update`, the `name` attribute is optional. However, if specified, it allows cross referencing among SQL commands as shown in the following example.

```
<sql-commands>
  <sql-command>
      <sql-query name="retrieve_medicine_identifiers">
           <parameters>
              <parameter name="c_scope"
                column="EHF_MEDICINE_CABINET.T_MEDICINE.C_SCOPE"/>
           </parameters>
          <parameterValues>
              <value name="c_scope" value="test_scope_1">
              </value>
          </parameterValues>
          <returns>
               <return name="C_ID"
                table="EHF_MEDICINE_CABINET.T_MEDICINE"/>
           </returns>
          <sql-statement>
              select C_ID from EHF_MEDICINE_CABINET.T_MEDICINE
                where C_SCOPE = :c_scope
          </sql-statement>
      </sql-query>
  </sql-command>

  <sql-command>
      <sql-update name="update_medicine_name" batchUpdate="true">
           <parameters>
              <parameter name="c_name"
                    column="EHF_MEDICINE_CABINET.T_MEDICINE.C_NAME"
                    update="true"/>
              <parameter name="c_id"
                    refCommand="retrieve_medicine_identifiers"
                    refReturn="C_ID" />
           </parameters>
          <parameterValues>
              <value name="c_name" value="medicine_1"></value>
          </parameterValues>
          <sql-statement>
              update EHF_MEDICINE_CABINET.T_MEDICINE
                set C_NAME=:c_name where C_ID in (:c_id)
          </sql-statement>
      </sql-update>
  </sql-command>
</sql-commands>
```

The `update_medicine_name` command references the preceding `retrieve_medicine_identifiers` command in the `parameter` element for `c_id` with the `refCommand` attribute. More precisely, it references the returned value from `C_ID` column. This allows passing results of a preceding command to parameters of a successor command at runtime. The `update` attribute for the parameter `c_name` is set to be true, since the column will be updated.

## Batch update

In the above example, the `retrieve_medicine_identifiers` command returns a list of identifiers. In a batch operation, the `update_medicine_name` command updates all medicine entries having identifiers specified in this list. The optional `batchUpdate` attribute in an `sql-update` element specifies whether the update is a batch update. By default, the attribute is set to be false.

### Cursors and loops in PL/SQL

With its control structures such as cursors and loops PL/SQL provides a powerful tool for conditional and iterative processing. Using a cursor, each row returned by an SQL statement can be processed one at a time in a loop. The support process supports PL/SQL procedures. Currently, PL/SQL functions are not supported. However, if a row referenced by the cursor consists of encryption relevant results, the support process must trigger the required decryptions to allow subsequent processing. Hence, the procedure originally defined for an un-encrypted database must be split into several procedures to give the support process access to intermediate results.

The following shows an example, which splits a procedure using a cursor and a loop into two procedures.

```
<sql-commands>
  <sql-command>
    <sql-procedure  name="read_intake_journal"
        schemaName="EHF_MEDICINE_CABINET">
        <parameters>
             <parameter name="in_c_scope"
                column="EHF_MEDICINE_CABINET.T_INTAKEJOURNAL.C_SCOPE"/>
        </parameters>
        <parameterValues>
            <value name="in_c_scope" value="test_scope_1"></value>
        </parameterValues>
        <returns>
             <return cursorName="result_cursor"
                table="EHF_MEDICINE_CABINET.T_INTAKEJOURNAL"/>
        </returns>
        <sql-statement>
        CREATE or REPLACE PROCEDURE
            EHF_MEDICINE_CABINET.read_intake_journal (
                in_c_scope VARCHAR2,
                result_cursor OUT SYS_REFCURSOR
            )
        IS
           BEGIN
            OPEN result_cursor FOR
               select * from EHF_MEDICINE_CABINET.T_INTAKEJOURNAL
                where c_scope = in_c_scope;
           END read_intake_journal;
        </sql-statement>
    </sql-procedure>
  </sql-command>

  <sql-command>
        <sql-procedure  name="update_entry_changer"
            schemaName="EHF_MEDICINE_CABINET"
            executeInLoop="true">
        <parameters>
             <parameter name="in_changerId"
                refCommand="read_intake_journal"
                refReturn="C_CHANGERID"
                update="true"/>
            <parameter name="in_scope" refCommand="read_intake_journal"
                refReturn="C_SCOPE"
                update="true"/>
             <parameter name="in_id" refCommand="read_intake_journal"
                refReturn="C_ID"/>
        </parameters>
        <sql-statement>
        CREATE or REPLACE PROCEDURE
```

```
            EHF_MEDICINE_CABINET.update_entry_changer (
                in_changerId VARCHAR2,
                in_id VARCHAR2,
                in_scope VARCHAR2
                )
        IS
           BEGIN
               UPDATE EHF_MEDICINE_CABINET.T_INTAKEENTRY
               SET C_CHANGERID = in_changerId, C_SCOPE = in_scope
               WHERE C_ENTRIES_ID = in_id;
               COMMIT;
           END update_entry_changer;
        </sql-statement>
    </sql-procedure>
  </sql-command>

</sql-commands>
```

The first procedure `read_intake_journal` returns a cursor. The `cursorName` attribute of a `return` element specifies the name of the cursor. After executing the procedure `read_intake_journal`, the support process loops over the result set to perform necessary decryptions of relevant returned column values.

The second procedure `update_entry_changer` references the results of `read_intake_journal`. In order to process the results in a loop, the `executeInLoop` attribute of the `sql-procedure` element is set to be true. By default, the `executeInLoop` attribute is set to be false.

**2.** Specify the support steps to be performed in a support configuration.

The support process provides two support steps, the `SimpleSQLSupportStep` and the `EncryptionSQLSupportStep`. The `SimpleSQLSupportStep` is used to execute SQL scripts against unencrypted data, while the `EncryptionSQLSupportStep` is used to execute SQL scripts against encrypted data. A support configuration file consists of one or multiple support steps.

The following shows examples of a `SimpleSQLSupportStep` and a `EncryptionSQLSupportStep`. For both steps, the `sqlCommandsFile` element specifies the name of the file consisting of SQL scripts to be executed, and the `sqlOutputFile` element specifies the name of the resulting SQL output file. If the `sqlOutputFile` element is not specified, no output will be written. Note, each support step must use a separate output file.

The `configLocation` element specifies the Spring application context to run the support process. Make sure, this context file is on the classpath of your assembly.

```
<support-configuration>
 <support-steps>
     <configLocation>classpath:/META-INF/ehf-support-context.xml</
configLocation>

     <SimpleSQLSupportStep>
       <sqlCommandsFile>encryption-sql-commands-medicine-cabinet.xml.xml</
sqlCommandsFile>
       <sqlOutputFile>simple-support-medicine-cabinet.result</sqlOutputFile>
     </SimpleSQLSupportStep>
  </support-steps>
</support-configuration>
```

```
<support-configuration>
 <configLocation>classpath:/META-INF/ehf-support-context.xml</configLocation>
 <upgradePath>configuration/database/upgrade/ehf-2.8-2.9</upgradePath>

 <support-steps>
   <EncryptionSQLSupportStep>
```

```
     <sqlCommandsFile>encryption-sql-commands-medicine-cabinet.xml</
sqlCommandsFile>
     <sqlOutputFile>encryption-support-medicine-cabinet.result</
sqlOutputFile>
    </EncryptionSQLSupportStep>
  </support-steps>
</support-configuration>
```

For an `EncryptionSQLSupportStep`, the `upgradePath` element specifies the location of table configurations (files named as `encryption-table-configs-module.xml`) for encryption relevant tables. Table configurations are automatically generated and part of the application release. Alternatively, you can specify the table configurations for each step using one or multiple `tableConfigurations` elements of `EncryptionSQLSupportStep`. Specifications in a `EncryptionSQLSupportStep` element override the configurations specified with the global `upgradePath` element. The following shows an example.

```
<support-configuration>


 <support-steps>
   <configLocation>classpath:/META-INF/ehf-support-context.xml</
configLocation>
   <EncryptionSQLSupportStep>
     <sqlCommandsFile>encryption-sql-commands-medicine-cabinet.xml</
sqlCommandsFile>
     <tableConfigurations>@upgrade.path@/encryption-table-configs-medicine-
cabinet.xml</tableConfigurations>
     <sqlOutputFile>encryption-support-medicine-cabinet.result</
sqlOutputFile>
    </EncryptionSQLSupportStep>
  </support-steps>
</support-configuration>
```

## 2.2 Sign and Pack Support Configurations

Since the support process utilizes the eHF Encryption Infrastructure to perform necessary encryption and decryption, it can potentially be misused to compromise data protected by encryption. For example, a malicious application administrator could submit arbitrary SQL commands to the database via the support process, and view results returned by the database in plaintext. Hence, both organizational and technical mechanisms must be in place to prevent misuses. Organizational mechanisms are beyond the scope of this document. Technical mechanisms make sure that the support process only accepts SQL commands which have not been tampered with and which originate from authorized, authenticated support agents. Moreover, technical mechanisms prevent application administrators from reading the results in plaintext.

1. Get a pair of private key and public key and the associated certificate.

   Each authorized support agent is equipped with a pair of private key and public key. The private key is used to sign SQL commands for their integrity and authenticity. The public key is used by the support process to verify the signature and to encrypt the SQL results. The support agent subsequently decrypts the SQL results using his/her private key.

   The private key must be stored in a JKS keystore file `keystore.keys` which is protected by a password. The same password must be used as store and key password. The certificate must be available to the support process.

2. Download the eHF Key Tool which consists of the support process from the code repository.

3. Create a folder (for example `support-cases/sql-commands-0001`) to keep the SQL scripts and the associated support configuration. The support configuration file must be named as `support-configuration.xml`. The support process uses the support configuration as a starting point to find the referenced SQL scripts.
4. Navigate to the root directory of eHF Key Tool, and invoke the `support:signcommands` maven goal to create an signed archive. For example, the following command creates the archive `ovsp0001.p7` for the content in the folder `sql-commands-0001`. The `keystore` argument specifies the keystore to retrieve the signing private key. The `passwd` argument specifies the password protecting the keystore.

```
maven support:signcommands -Dfolder=support-cases/sql-commands-0001
-Doutput=ovsp0001.p7 -Dkeystore=keystore.keys -Dpasswd=changeit
```

5. Send the resulting archive to the hoster to execute the support process.

## 2.3 Decrypt and Unpack Support Results

Result files of the support process are encrypted with the public key of the support agent and stored in an archive.

1. Create a folder (for example `support-cases/sql-results`) to keep the SQL results. Copy the result archive received from the hoster (for example `sql_outputs_2010315_1268667784723.p7`), to this folder.
2. Navigate to the root directory of eHF Key Tool, and invoke the `support:unpackresults` maven goal to decrypt and unpack the results in the archive. For example, the following command unpacks and decrypts the result files in the archive `sql_outputs_2010315_1268667784723.p7` to the folder `sql_outputs_2010315_1268667784723`. The same keystore and password used for signing are used for the `keystore` argument and `passwd` argument respectively.

```
maven support:unpackresults
-Darchive=support-cases/sql-results/sql_outputs_2010315_1268667784723.p7
-Dkeystore=keystore.keys -Dpasswd=changeit
```

## 2.4 Private Key Policy

### 2.4.1 Why keep your private key private?

Your private key is a very powerful in order to get access to sensitive data. There are three aspects of the use of your private key:

- It proofs to the support process that you are authorized to execute SQL-commands – anyone who uses your key can do the same.
- It links audit entries to you – if someone else executes SQL-commands with your key (especially in a compromising way), you will be blamed for it.
- It decrypts the result files of the support process – anyone holding your key can read all result files (current ones, but all past results too).

Those are the reasons why you are obliged to keep your private key safely to yourself.

### 2.4.2 How to keep your private key private

You are responsible for storing your private key in a way that only you can access and use it. Your private key is contained in the file `keystore.keys` and protected by a pass-word. As a guideline, handle it like a credit card with PIN:

- don’t share the password with others - not even your colleagues, if they need access, they should have their own private key.
- don’t write your password down (if you have problems to remember it, use tricks like inventing a sentence where the first letters of each word sum up to your password)
- keep your private key (i.e. the file `keystore.keys`) on a separate medium, that is under your strict control. Lock it up where only you can access it, take it out only when needed, don’t leave it unattended. Be sure that no copies are available to others.
- In case you suspect that your private key got known to others (original or copy), please contact your manager. The key will be exchanged to prevent further use.

**Important:**

As nobody else has a copy and can restore the data, it is recommended that you keep a backup copy in a safe location accessible only to yourself.

## 2.5 Multi-Schema Support

The eHF multi-schema support feature allows for using a single database instance for multiple eHF-based applications. In order to separate data of 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 support process.

1. Change the schema names in SQL scripts.

   Both the schema names in SQL statements as well as in metadata (i.e. in `parameter` and `return` elements) must be modified. The following shows an example.

```
<sql-command>
  <sql-query  name="query_intakeEntry">
     <parameters>
        <parameter name="c_scope"
            column="PRE_EHF_MEDICINE_CABINET_SUF.T_INTAKEENTRY.C_SCOPE"/>
        <parameter name="c_amount"
            column="PRE_EHF_MEDICINE_CABINET_SUF.T_INTAKEENTRY.C_AMOUNT"/>
     </parameters>
     <parameterValues>
       <value name="c_scope" value="test_scope_1"/>
       <value name="c_amount" value="1"/>
     </parameterValues>
     <returns>
         <return name="C_ID"
            table="PRE_EHF_MEDICINE_CABINET_SUF.T_INTAKEENTRY"/>
         <return name="C_SCOPE"
            table="PRE_EHF_MEDICINE_CABINET_SUF.T_INTAKEENTRY"/>
         <return name="C_Amount"
            table="PRE_EHF_MEDICINE_CABINET_SUF.T_INTAKEENTRY"/>
     </returns>
     <sql-statement>
```

```
        select C_ID, C_SCOPE, C_Amount
            from PRE_EHF_MEDICINE_CABINET_SUF.T_INTAKEENTRY
            where C_SCOPE = :c_scope and C_AMOUNT = :c_amount
     </sql-statement>
  </sql-query>
  </sql-command>
```

2. Specify the schema prefix and suffix in the support configuration.

   Add a `schema-configuration` element which specifies the schema prefix and suffix to the support configuration. An example is shown in the following.

```
<support-configuration>
 <configLocation>classpath:/META-INF/ehf-support-context.xml</configLocation>
  <schema-configuration prefix="PRE" suffix="SUF" />
  <upgradePath>configuration/database/upgrade/ehf-2.8-2.9</upgradePath>

  <support-steps>
   <EncryptionSQLSupportStep>
     <sqlCommandsFile>encryption-sql-commands-medicine-cabinet.xml</
sqlCommandsFile>
     <sqlOutputFile>encryption-support-medicine-cabinet.result</
sqlOutputFile>
    </EncryptionSQLSupportStep>
  </support-steps>
</support-configuration>
```

# 3 Administrator's Guide

The following describes the required steps for the Support Administrator to run database scripts with the support process against an encrypted database.

## 3.1 Run the Support Process

1. Navigate to the binary folder of your application installation folder: `%> cd [application-installation-folder]/application-version-bin`
2. Copy the archive of signed SQL commands (for example, `ovsp0001.p7`) from a support agent to the `support` folder under `[application-installation-folder]/application-version-bin`.
3. Execute the following command to run the support process. You will find the results as an archive (for example `sql_outputs_2010315_1268667784723.p7`) under the folder `[application-installation-folder]/application-version-bin`.

   ```
   ant -f install/support/support.xml support:execute
   -Dsupport.config=support/ovsp0001.p7
   ```
4. Send the result archive to the requesting support agent.

# 4 Troubleshooting

**When executing some SQL commands I got an `IncompleteCryptoTargetException`.**

If the execution of a support process fails with `IncompleteCryptoTargetException` and the message `Missing id for encryption with an id-based IV in crypto target` or `Missing scope for encryption with an scope-based IV in crypto target` then this is an indicator that the configured encryption for this field classification needs this values in addition. To know that in advance we must lookup the `target-mapping` for all fields classifications .

**Tip:**

If it is not clear if and how the encryption is based a quick solution could be to always add the two possible C_SCOPE and C_ID in the parameters and parameterValues of the update.

Please keep in mind that not all valid SQL commands executed against an unencrypted database are valid commands against an encrypted database. The following SQL command updates the `C_NAME` column of table `T_MEDICINE` for rows where `C_SCOPE` equals to `'test_scope_1'`. However, values of the `C_NAME` column are instance-based encrypted, i.e. encryption parameters are derived dependent from the identifier of each instance (row). Values of the `C_ID` column must be available for encrypting `C_NAME` before they are inserted to the database for update.

```
update EHF_MEDICINE_CABINET.T_MEDICINE set C_NAME='medicine_1'
where C_SCOPE='test_scope_1'
```

The above update command must be split into the following two SQL commands to be executed against an encrypted database. The first one retrieves the `C_ID` column, while the second command refers to the retrieved values.

```
<sql-command>
  <sql-query name="query_medicine_retrieve_C_ID">
    <parameters>
       <parameter name="c_scope"
                  column="EHF_MEDICINE_CABINET.T_MEDICINE.C_SCOPE"/>
    </parameters>
    <parameterValues>
       <value name="c_scope" value="test_scope_1">
       </value>
    </parameterValues>
    <returns>
        <return name="C_ID" table="EHF_MEDICINE_CABINET.T_MEDICINE"/>
    </returns>
    <sql-statement>
       select C_ID from EHF_MEDICINE_CABINET.T_MEDICINE
       where C_SCOPE = :c_scope
    </sql-statement>
  </sql-query>
</sql-command>

<sql-command>
  <sql-update name="update_medicine_for_C_ID" batchUpdate="true">
    <parameters>
```

```
       <parameter name="c_name" column="EHF_MEDICINE_CABINET.T_MEDICINE.C_NAME"
                  update="true"/>
       <parameter name="c_id" refCommand="query_medicine_retrieve_C_ID"
           refReturn="C_ID" />
    </parameters>
   <parameterValues>
       <value name="c_name" value="medicine_1"></value>
   </parameterValues>
   <sql-statement>
       update EHF_MEDICINE_CABINET.T_MEDICINE set C_NAME=:c_name
       where C_ID in (:c_id)
   </sql-statement>
  </sql-update>
</sql-command>
```