**AES Decryption:**

**Source code:**







**Steps to run:**

Create JAR of project and run it.

**OR**

Run **RunDecrption.java** class from eclipse.

**About Project:**

Project has two important classes

**SecuredRSAUsage**: It has decrypt() and verifySignature() methods. This class is used for Decrypting session key and verify signature.

**SecuredGCMUsage:** It is used for decrypting payload.

**VAULT POC:**

**Wildfly vault setup**

**Below steps are tested using version:**

Wildfly verion: 12.0.0.Final

Java Version: 1.8.0\_181

**Run the following command from command line:**

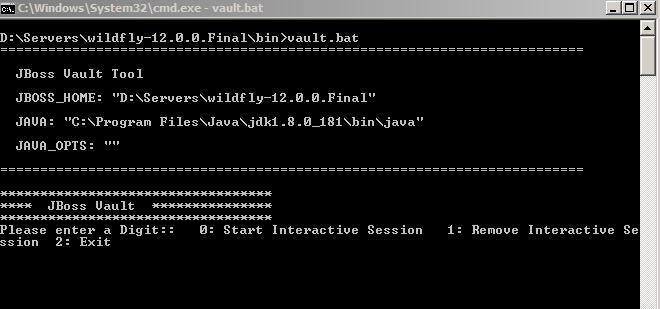
keytool -genseckey -alias gomzee -storetype jceks -keyalg AES -keysize 128 -storepass gomzee1! -keypass gomzee1! -keystore D:\Servers\vault\vaultkey.store

**NOTE:** storepass and keypass should be same otherwise it will give error while running vault.bat/vault.sh of WildFly. There are few tickets like this already opened: [**https://issues.jboss.org/browse/SECURITY-710**](https://issues.jboss.org/browse/SECURITY-710)

As per the command **vaultkey.store** gets created on the specified location.

Go to **<WILDFLY\_HOME>/bin** and run **vault.bat/vault.sh** depending on OS.

You will see like below screen.

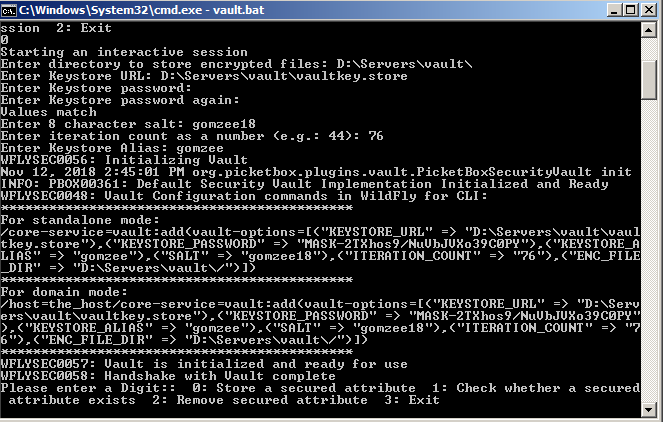


**Screen 1**

Start the interactive session by typing **0.**

In interactive session, it will ask for few details. Refer below screen for same.

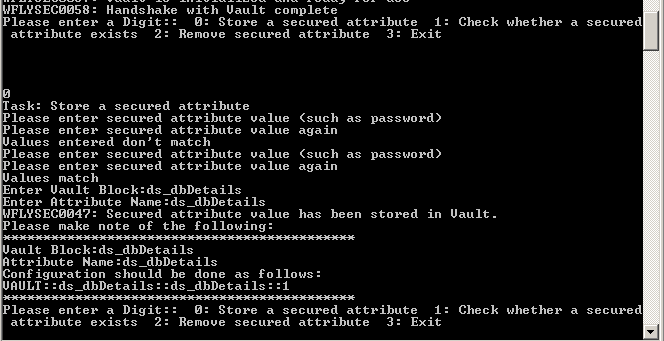
Here, Keystore Password and Alias name should be as per **keytool** command which is run in starting.



**Screen 2**

Now, in order to store secured strings to vault use option **0: Store a secured attribute.**

And you will be prompted with options as per below screen:



**Screen 3**

Make note of value- **VAULT::ds\_dbDetails::ds\_dbDetails::1** as we will be using this to fetch value.

From **<WILDFLY\_HOME>/bin**  RUN **jboss-cli.bat/jboss-cli.sh** based on OS. Before running this make sure your Wildfly server is UP and running.

Once you are connected to server, **run the below command** which you should have got as per Screen 2:

**/core-service=vault:add(vault-options=[("KEYSTORE\_URL" => "D:\\Servers\\vault\\vaultkey.store"),("KEYSTORE\_PASSWORD" => "MASK-2TXhos9/NuVbJVXo39C0PY"),("KEYSTORE\_ALIAS" => "gomzee"),("SALT" => "gomzee18"),("ITERATION\_COUNT" => "76"),("ENC\_FILE\_DIR" => "D:\\Servers\\vault\\")])**

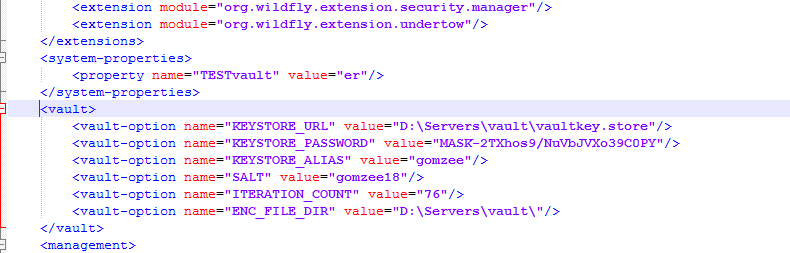
**NOTE:** Make sure of proper path with slashes as per OS.Command as per above screen has only single slash in **KEYSTORE\_URL and ENC\_FILE\_DIR**. Make sure to use **double** slashes.

Successful outcome will be like:



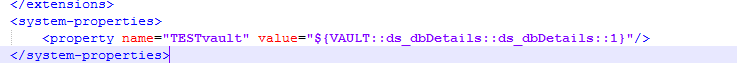
**Screen 4**

If you check your **<WILDFLY\_HOME>\standalone\configuration\standalone.xml** it will have entry of **<vault>** now.



**Screen 5**

Finally we can use **VAULT::ds\_dbDetails::ds\_dbDetails::1** which we have got above like below example screen in our **standalone.xml.**



**Screen 6**

**Another example usage:**

<subsystem xmlns="urn:jboss:domain:datasources:1.0">

<datasources>

<datasource jndi-name="java:jboss/datasources/ExampleDS" enabled="true" use-java-context="true" pool-name="H2DS">

<connection-url>jdbc:h2:mem:test;DB\_CLOSE\_DELAY=-1</connection-url>

<driver>h2</driver>

<pool></pool>

<security>

<user-name>sa</user-name>

**<password>${VAULT::ds\_dbDetails::ds\_dbDetails::1}</password>**

</security>

</datasource>

<drivers>

<driver name="h2" module="com.h2database.h2">

<xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>

</driver>

</drivers>

</datasources>

</subsystem>

**OPI-SANITY-TOOL**

**Source Code:**

****

**Description:**

This project is used to run a sanity check for following API’s:

* Payment
* Collection
* Paynow
* Collection Initiation
* EDDA Type 1 (Registration, Cancel/Approval, Status)
* EDDA Type 2 (Registration, Cancel, Status)

**Starting point:** OpiSanityApplication.java. It reads trustStore from client.jks file and Initiates the further processing.

**Application.properties:** All the sequences of API calls are their configurations are defined in application.properties file.

***api.name***this property is used to define the sequence of API calls. It’s basically an array, so index sequence should be maintained.

All other files like certs, json etc. can be found in **sanity-tool.zip.**

Once the JAR is build from project keep it inside **sanity-tool/FILES/** and then RUN **sanity-tool/sanity-run.bat** (Update sanity-run.bat for paths as per your file system).

**OPI-SANITY-LOAD-TOOL**

**Source Code:**



**Description:**

Code base this is almost same like OPI-SANITY-TOOL. Purpose of this tool is to make n number of parallel requests to EDDA type 1 and EDDA type 2 API calls and generate report.

**TODO:**

Check on JSON request, if they are properly getting created or not.

Different response handling from API calls.

Once the JAR is build from project keep it inside **opi-load-tool/FILES/** and then RUN **opi-load-tool/** **opi-run-load.bat** (Update opi-run-load.batfor paths as per your file system).