Hibernate Password Encryption with Jasypt in Spring MVC

Create Password

After downloaded Jasypt CLI Tools, execute following code by using **encrypt.sh** for linux based OS, or **encrypt.bat** file for Windows located in **bin** folder:

```
encrypt.bat input="secret" password=encryptorpassword algorithm=PBEWithMD5AndTripleDES
```

Output looks like this: `AdK2HjMDfxTABg9ZP3kXSWsKo3t4rSn7`

Note: Whenever run above command in command prompt, you will get different password each time because **PBEWithMD5AndTripleDES** algorithm and many other algorithms use **random salt generator**. For more information please click

Add Maven Dependencies

```
<dependency>
    <groupId>org.jasypt</groupId>
    <artifactId>jasypt</artifactId>
    <version>1.9.2</version>
    <scope>compile</scope>
</dependency>
<dependency>
    <groupId>org.jasypt</groupId>
    <artifactId>jasypt-spring31</artifactId>
    <version>1.9.2</version>
    <scope>compile</scope>
</dependency>
<dependency>
    <groupId>org.jasypt</groupId>
    <artifactId>jasypt-hibernate4</artifactId>
    <version>1.9.2</version>
    <scope>compile</scope>
</dependency>
```

Hibernate.cfg.xml File

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
          "-//Hibernate/Hibernate Configuration DTD//EN"
          "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
    <hibernate-configuration>
        <session-factory>
            <property name="connection.provider_class">
                com.codesenior.config.EncryptedPasswordC3P0ConnectionProvider
            </property>
            <property name="connection.encryptor_registered_name">
                strongHibernateStringEncryptor
            </property>
            <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
            <property name="connection.url">
                jdbc:mysql://localhost:3306/Test?autoReconnect=true&
                useUnicode=true&characterEncoding=utf-8
            </property>
            <property name="connection.username">root</property>
            <property name="connection.password">ENC(AdK2HjMDfxTABg9ZP3kXSWsKo3t4rSn7)</property>
            <property name="c3p0.min_size">5</property>
            <property name="c3p0.max_size">20</property>
            <property name="c3p0.timeout">1800</property>
            <property name="c3p0.max_statements">50</property>
        </session-factory>
    </hibernate-configuration>
```

At line 16, we changed password value with above encrypted value. Please note that, we should use **ENC()** when specifying password.

At line 8, I have created a custom class named as EncryptedPasswordC3P0ConnectionProvider because **org.hibernate.service.jdbc.connections.internal.C3P0ConnectionProvider** class was moved to **org.hibernate.c3p0.internal** package in **Hibernate 5.0.4.Final**, but Jasypt's EncryptedPasswordC3P0ConnectionProvider class extends **old C3P0ConnectionProvider.** Also `public void configure(Map props)` method defined in **C3P0ConnectionProvider** in Hibernate 4 and Hibernate 5 is different than `configure(java.util.Properties props)` method of the EncryptedPasswordC3P0ConnectionProvider. If this custom class will not be used, you can not **decrypt** encrypted password because configure method should be overrriden. Lets look at the custom **EncryptedPasswordC3P0ConnectionProvider** class:

```
1   import java.util.Map;
2   import org.hibernate.c3p0.internal.C3P0ConnectionProvider;
3   import org.hibernate.cfg.AvailableSettings;
4   import org.jasypt.encryption.pbe.PBEStringEncryptor;
5   import org.jasypt.exceptions.EncryptionInitializationException;
6   import org.jasypt.hibernate4.connectionprovider.ParameterNaming;
7   import org.jasypt.hibernate4.encryptor.HibernatePBEEncryptorRegistry;
8   import org.jasypt.properties.PropertyValueEncryptionUtils;
9
10  public final class EncryptedPasswordC3P0ConnectionProvider extends C3P0ConnectionProvider {
11      private static final long serialVersionUID = 5273353009914873806L;
12
13      public EncryptedPasswordC3P0ConnectionProvider() {
14          super();
15      }
16
17      public void configure(Map props) {
18          final String encryptorRegisteredName =
19          (String) props.get(ParameterNaming.ENCRYPTOR_REGISTERED_NAME);
20
21          final HibernatePBEEncryptorRegistry encryptorRegistry =
22          HibernatePBEEncryptorRegistry.getInstance();
23          final PBEStringEncryptor encryptor =
24          encryptorRegistry.getPBEStringEncryptor(encryptorRegisteredName);
25
26          if (encryptor == null) {
27              throw new EncryptionInitializationException(
28              "No string encryptor registered for hibernate " +
29              "with name \"" + encryptorRegisteredName + "\"");
30          }
31
32          // Get the original values, which may be encrypted
33          final String driver = (String) props.get(AvailableSettings.DRIVER);
34          final String url = (String) props.get(AvailableSettings.URL);
35          final String user = (String) props.get(AvailableSettings.USER);
36          final String password = (String) props.get(AvailableSettings.PASS);
37
38          // Perform decryption operations as needed and store the new values
39          if (PropertyValueEncryptionUtils.isEncryptedValue(driver)) {
40              props.put(
41              AvailableSettings.DRIVER,
42              PropertyValueEncryptionUtils.decrypt(driver, encryptor));
43          }
44          if (PropertyValueEncryptionUtils.isEncryptedValue(url)) {
45              props.put(
46              AvailableSettings.URL,
47              PropertyValueEncryptionUtils.decrypt(url, encryptor));
48          }
49          if (PropertyValueEncryptionUtils.isEncryptedValue(user)) {
50              props.put(
51              AvailableSettings.USER,
52              PropertyValueEncryptionUtils.decrypt(user, encryptor));
53          }
54          if (PropertyValueEncryptionUtils.isEncryptedValue(password)) {
55              props.put(
56              AvailableSettings.PASS,
57              PropertyValueEncryptionUtils.decrypt(password, encryptor));
58          }
```

```
59
60          // Let Hibernate do the rest
61          super.configure(props);
62
63          }
64      }
```

At line 11, **strongHibernateStringEncryptor** id reference points to a Spring bean which is set using **registeredName** property:

```
1   <bean id="hibernateStringEncryptor"
2        class="org.jasypt.hibernate4.encryptor.HibernatePBEStringEncryptor">
3       <property name="registeredName" value="strongHibernateStringEncryptor"/>
4       <property name="algorithm" value="PBEWithMD5AndTripleDES"/>
5       <property name="password" value="${MAIL_SERVICE_3DES_PASSWORD}"/>
6   </bean>
```

Note: MAIL_SERVICE_3DES_PASSWORD is a system environment variable. To access an environment variable from Spring configuration file, we should firstly add `<context:property-placeholder />` element in the Spring configuration xml file and should use dollar sign with curly braces.

Lets look at the Spring configuration file:

```
1   <beans xmlns="http://www.springframework.org/schema/beans"
2          xmlns:context="http://www.springframework.org/schema/context"
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xmlns:tx="http://www.springframework.org/schema/tx"
5          xmlns:util="http://www.springframework.org/schema/util"
6          xsi:schemaLocation="http://www.springframework.org/schema/beans
7           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
8           http://www.springframework.org/schema/context
9           http://www.springframework.org/schema/context/spring-context-3.0.xsd
10          http://www.springframework.org/schema/tx
11          http://www.springframework.org/schema/tx/spring-tx.xsd
12          http://www.springframework.org/schema/util
13          http://www.springframework.org/schema/util/spring-util.xsd">
14
15      <context:component-scan base-package="com.codesenior">
16          <context:exclude-filter type="annotation"
17                                  expression="org.springframework.stereotype.Controller"/>
18      </context:component-scan>
19      <context:property-placeholder /><!--enable accessing system environment variables-->
20      <tx:annotation-driven transaction-manager="transactionManager"/>
21
22      <!--
23      This is the key to link Spring's injection to Hibernate event-based validation.
24      Notice the first constructor argument, this is our Spring ValidatorFactory instance.
25      -->
26      <bean id="customInitialize" class="com.codesenior.config.CustomInitialize">
27          <property name="encryptorPasswordVariable" value="MAIL_SERVICE_3DES_PASSWORD"/>
28      </bean>
29
30      <bean id="hibernateStringEncryptor"
31           class="org.jasypt.hibernate4.encryptor.HibernatePBEStringEncryptor">
32          <property name="registeredName" value="strongHibernateStringEncryptor"/>
33          <property name="algorithm" value="PBEWithMD5AndTripleDES"/>
34          <property name="password" value="${MAIL_SERVICE_3DES_PASSWORD}"/>
35      </bean>
36      <bean id="transactionManager"
37           class="org.springframework.orm.hibernate5.HibernateTransactionManager">
38          <property name="sessionFactory" ref="sessionFactory"/>
39      </bean>
40
41      <bean id="sessionFactory" depends-on="hibernateStringEncryptor"
42           class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
43          <property name="configLocation" value="WEB-INF/hibernate.cfg.xml"/>
44          <property name="packagesToScan" value="com.codesenior.model"/>
45          <property name="hibernateProperties">
46              <props>
47                  <!-- <prop key="hibernate.hbm2ddl.auto">create</prop>
48                  <prop key="hibernate.hbm2ddl.import_files">import.sql</prop>
```

```
                -->
                <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
                <prop key="hibernate.connection.CharSet">utf8</prop>
                <prop key="hibernate.connection.characterEncoding">utf8</prop>
                <prop key="hibernate.connection.useUnicode">true</prop>
                <prop key="hibernate.show_sql">true</prop>
                <prop key="hibernate.format_sql">true</prop>
                <prop key="hibernate.cache.use_second_level_cache">true</prop>
                <prop key="hibernate.cache.use_query_cache">true</prop>
                <prop key="hibernate.generate_statistics">false</prop>
                <prop key="javax.persistence.validation.mode">none</prop>
            </props>
        </property>
    </bean>
</beans>
```

At line 41, `depends-on` attribute provides that hibernateStringEncryptor bean initialization will be completed before **sessionFactory** bean. Therefore, encrypted password is decryped before used.

At line 26, we created a new class CustomInitiliaze. This class will be used to **clear** the MAIL_SERVICE_3DES_PASSWORD variable after the application is started. Thus, an attacker can't find the encryption password.

Lets look at the **CustomInitialize** class:

```
import com.codesenior.telif.service.mail.service.AtomicService;
import org.springframework.beans.factory.InitializingBean;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

/**
  * Runs only once after Spring beans initialized
  */
@Component
public class CustomInitialize implements InitializingBean {
     @Autowired
     private AtomicService atomicService;
     private String encryptorPasswordVariable;

     @Override
     public void afterPropertiesSet() throws Exception {
         if (encryptorPasswordVariable != null) {
             Runtime runtime = Runtime.getRuntime();
             /*
             Set modifies the current shell's (window) environment values,
             and the change is available immediately, but it is temporary.
             the change will not affect other shells that
             are running, and as soon as you close the
             shell, the new value is lost until such time as you run set again.

             setx modifies the value permenantly, which affects all
             future shells, but does not modify the environment
             of the shells already running. you have to exit the shell and reopen
             it before the change will be available, but the value will remain
             modified until you change it again.
              */
             runtime.exec("setx " + encryptorPasswordVariable+ " \"\" /M");//empty value is set
         }
         //atomicService.rebuildIndex();
     }

     public String getEncryptorPasswordVariable() {
         return encryptorPasswordVariable;
     }

     public void setEncryptorPasswordVariable(String encryptorPassword) {
         this.encryptorPasswordVariable= encryptorPassword;
     }
}
```

Result

Jasypt library simplifies encryption and **decryption** operations with huge number of different encryptors such as HibernatePBEStringEncryptor, StringEncryptor etc.

Note: The word encryptor may be confusing at first glance, but encryptors in Jasypt library execute both encryption and decryption operations by using **encrypt()** and **decrypt()** methods.

You don't need to know random salt value when decrypting the encrypted password. Jasypt automatically detects salt values in the encrypted value and removes the salt.