**LAB 4 – SALT**

**Student Name:** Eryk Gloginski **(L00157413)**

**Course:** BSc Computing

**Module:** Secure Programming

**Lecturer:** Maria Griffin

**Submission Date:** 22/03/2023

## **1. Introduction:**

Salting is a well-known and often used security technique in programming that adds an extra layer of protection to user passwords. Salting involves adding a random string of characters to each password before encrypting and storing it. When a user logs in, the system retrieves the salt from the database and adds it to the user’s entered password.

## **2. Aims**:

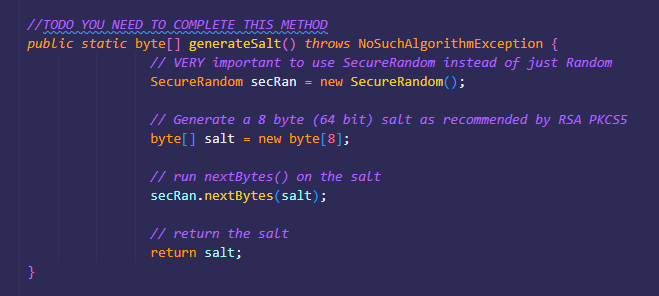
The aims of this lab are to:

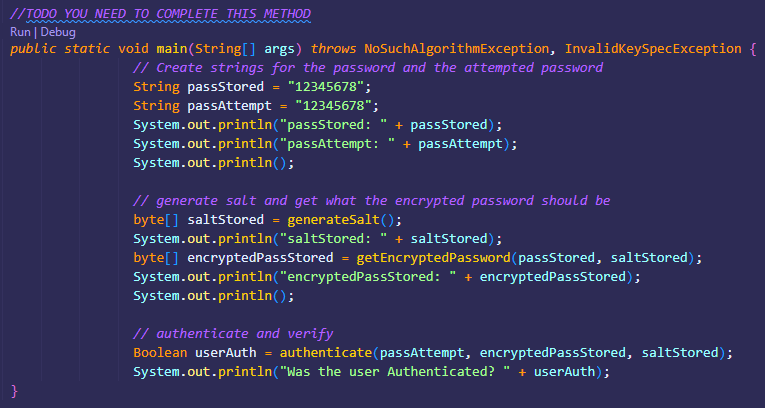
* Use the code provided for the class **PasswordEncryptionService.java**.
* Complete the code in the **generateSalt()** method to generate a salt.
* Add test code to the **main()** method to encrypt the password and check if the authentication matches the attempted password.

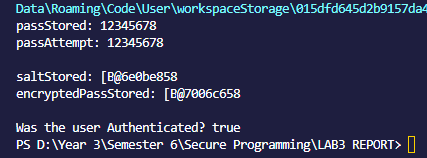
## **3. Method:**

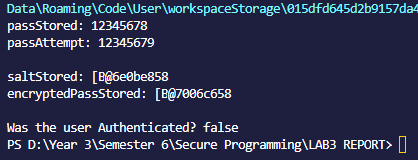
The method that we will be using for salting is called “**PBKDF2WithHmacSHA1**”. “**PBKDF2**” stands for Password-Based Key Derivation Function 2 while “**WithHmacSHA1**” is a cryptographic algorithm used to hash the password with the salt. Together, it is a key derivation function that takes the password and salt as an input and produces a derived key as output.

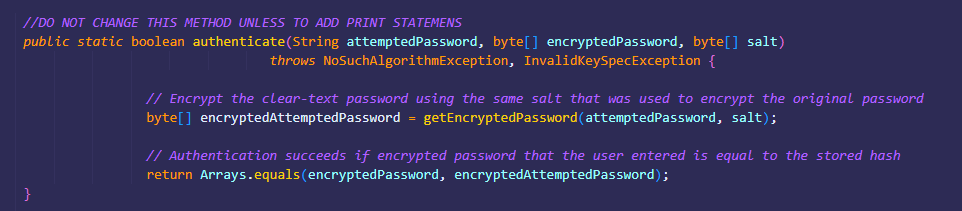
## **4. Results:**

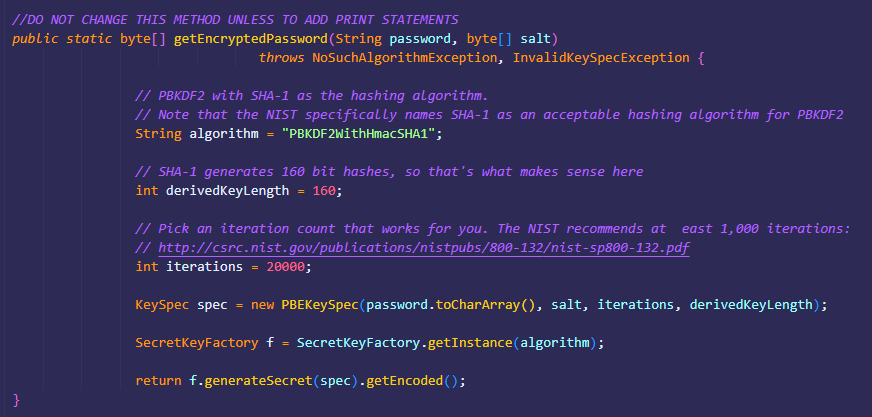
We start by completing the **generateSalt()** method so it returns a **byte[] salt**. We use the **SecureRandom** class to create a **secRan** object, define a **byte[] salt** variable by giving it 8 bytes and then run the **nextBytes()** method on the **salt**. With that, we can generate and return a **salt**. The code in the method below is for generating the salt.  


After that, we make the **test main()** method where we test out all the methods, starting with defining variables for the **stored** password, **attempted** password. We generate the salt using the **generateSalt**() method and we print out the stored salt. The code below is to check if the **passStored** and **passAttempt** store the same values.

We can see below the output of the code in the main method. It prints out the **passStored** and **passAttempt** to verify that they are indeed the same values. After it prints out the generated salt as **saltStored** and it encrypts the password using that generated salt with the **getEncryptedPassword()** method, which is also printed out. Lastly, it verifies if the user has been authenticated using the **authenticate()** method which passes the **passAttempt**, **encryptedPassStored** and **saltStored** and returns a Boolean.  


However, what happens if **passStored** and **passAttempt** are not the same values? When that happens, the **authenticate()** method takes care of it and returns a false, rather than a true.

The code below is a snippet of the provided **authenticate()** method. We can see here that the **attemptedPassword** and **salt** parameters are being used in the **getEncryptedPassword()** method to retrieve an **encryptedAttemptedPassword**. After that it returns a Boolean with the **Arrays.equals()** method using **attemptedPassword** and **encryptedAttemptedPassword**.  


The code below is also a snipped of the provided **getEncryptedPassword()** method. We can see here that there are predefined **algorithm**, **derivedKeyLength** and **iterations** variables. It gets the **KeySpec spec** object by using the **password**, **salt**, **iterations** and **derivedKeyLength** variables. The **SecretKeyFactory** object is made using the **getInstance()** method and using the **algorithm** variable as a parameter. Finally, it returns the **SecretKeyFactory** object which uses the **generateSecret()** method which also uses the **getEncoded()** method.  


## **5. Conclusion:**

In conclusion, salting refers to the process of adding a salt value to a user’s password before storing it in the database. It helps to increase the security of the stored passwords by making it harder for attackers to crack them.

The salt value is generated randomly and is unique to each user. It is added with the user password before applying a cryptographing function hashing function. The hash is then stored in the database alongside the salt value.

When a user attempts to login, the salt is retrieved from the database and added to the entered password. The retrieved hash is compared with the hash in the database. If they match, the user is granted access.

## **6. References:**

1. <https://www.baeldung.com/java-password-hashing>
2. <https://supertokens.com/blog/password-hashing-salting>
3. <https://docs.oracle.com/javase/8/docs/api/java/security/SecureRandom.html>
4. <https://docs.oracle.com/javase/7/docs/api/javax/crypto/SecretKeyFactory.html>
5. <https://docs.oracle.com/javase/7/docs/api/javax/crypto/spec/PBEKeySpec.html>