**CA 3 - Lab 3 – SALT Report**

Daniel Gallagher

L00158616

[L00158616@atu.ie](mailto:L00158616@atu.ie)

7th May 2023

# Introduction

A method for enhancing the security of user passwords that are stored is password salting. Before hashing, each password is given a special, random string of characters, which greatly reduces the likelihood that an attacker will be successful in cracking it. With the help of this lab exercise, I will be able to see how password salting is used in Java and gain an understanding of its significance for password security. Password salting is frequently used in conjunction with other security measures, like hashing algorithms (like MD5 or SHA-1), to further enhance the security of passwords that are stored. This lab aims to provide a deeper understanding of salting and how it is used in Java code.

# Aims/Objectives

* Implement a generateSalt() method to create a unique salt for each password.
* Test the password salting functionality using the provided Java code template.
* Understand the significance of password salting in enhancing security.
* Learn how to apply password salting in combination with hashing algorithms for better security.

# Method

**Step 1: Completing the code in the generateSalt() method to generate salt.**

A screen shot of a computer program

Description automatically generated with low confidence

*Fig 1 – The generateSalt() method in the java code*

To produce a random and distinct salt, I implemented the generateSalt() method using the SecureRandom class. To create an instance of SecureRandom, the "SHA1PRNG" algorithm calls the getInstance() method. Then, to store the salt, I made an 8-byte array and filled it with random bytes using the nextBytes() method (Fig. 1).

**Step 2: Adding the test code to the main() method to encrypt the password and authentic to check if they match.**

A picture containing text, screenshot, software, operating system

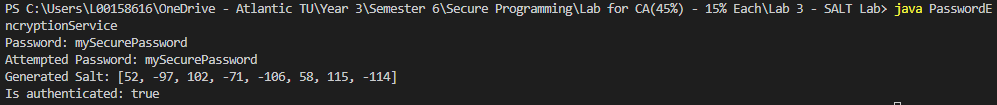
Description automatically generated

*Fig 2 – Test code to the main() method for password encryption and authentication*

To demonstrate password encryption and authentication using the salt, I added the test code to the main() method in this step. I created a password and attempted Password to see how it worked. I then created an encrypted password using the getEncryptedPassword() method and generated a salt using the generateSalt() method. I then used the authenticate() method to see if the password that was entered matched the one that was used, and I printed the results (Fig. 2).

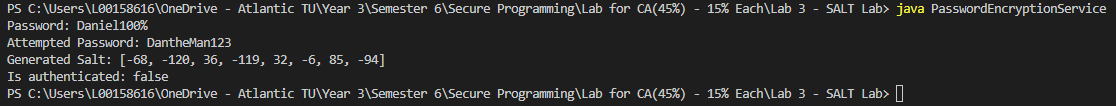
# Results and Testing

I ran the main() method and looked at the results. The initial password, the compromised password, the generated salt, and the authentication outcome were all visible to me. I was able to comprehend how the salt affects the encrypted password and the overall authentication process by analysing the results (Fig. 3).



*Fig – 3 Output of the main() method and the original password and the attempted password is the same, as well as the generated salt and the authentication result.*

For instance, the output will indicate that the authentication was successful if the attempted password matches the original password. The password salting and encryption mechanisms are operating as intended, though, if the attempted password is incorrect, the authentication would fail (Fig. 4).



*Fig – 4 Output of the main() method and the original password and the attempted password is not the same.*

**Salt Code**

The generateSalt() method worked as intended, producing a special, random salt for each password. Since the SecureRandom class offers a higher level of security than the regular Random class, it was used to create the salt. The getEncryptedPassword() method then employs this salt in conjunction with the hashing algorithm to create a more secure encrypted password.

**Understanding/Explanation**

By adding a salt, the user's password becomes more complex, making it more challenging for attackers to break it using precomputed tables (like rainbow tables). Additionally, if an attacker manages to break into the system, using a different salt for each user helps prevent them from simultaneously cracking multiple passwords.

# Conclusion

A crucial method for ensuring the security of user passwords is password salting. I used Java to implement a password salting mechanism in this lab exercise. Each password is given a special, random salt by the generateSalt() method, making it more difficult for an attacker to guess the password. This method is essential for boosting the security of password storage systems and can be used with a variety of programming languages. It becomes even more difficult for attackers to gain unauthorized access to user accounts when password salting and hashing algorithms are combined to further strengthen the security of stored passwords. By looking at the outcomes of our implementation, I was able to better understand how password salting contributes to improved password security.

In conclusion, putting password salting into practice is a crucial step for enhancing password storage security and guarding sensitive user data from unauthorized access.