Objective:

The objective of this lab was to implement integrity and authentication functionalities using hashing techniques in Java. The lab focused on utilizing the Java Cryptography Architecture (JCA) to implement concepts such as message digests and message authentication codes (MACs). The lab involved the following tasks:

Understanding and running the provided code for hashing.

Changing the hashing algorithm and observing the outputs.

Implementing data integrity using hashing.

Implementing authentication using hashing and MACs.

Hashing:

The lab started with a provided Java file named Hash.java. The code implemented hashing using the MessageDigest class from JCA. We ran the code, entered different messages, and observed the corresponding hash outputs. The code successfully converted plaintext messages into hash values using the SHA-256 algorithm.

Changing Hashing Algorithms:

The JCA provides various hashing algorithms. We modified the code to use two different algorithms: SHA-512 and MD5. We observed the outputs for the same plaintext messages using different algorithms. We recorded the following results:

Algorithm: SHA-256

Message: "Hello, World!"

Output Hash: 60E43124E19739BFA40BEE325BEDBBA8B6458EAD3F6F0F2EE4E7FCD47D4D7A9C

Algorithm: SHA-512

Message: "Hello, World!"

Output Hash:
2EF7BDE608CE5404E97D5F042F95F89F1C232871DF33A952D62E73E9FEC1FABCACBD1EAA0E03

0EACA5CBCF3ACDF247CBE1E9D42E03F8FDB4BA3F6C31B8773F98A9

Algorithm: MD5

Message: "Hello, World!"

Output Hash: ED076287532E86365E841E92BFC50D8C

Data Integrity Using Hashing:

We created a new Java file called Sender.java. It contained a function named requestInput() that prompted the sender to enter a message. Another function named createHash(String msg) accepted the user's message and produced a hash in the hexadecimal format.

We also created a separate Java file named Receiver.java. It included a function named verifyHash(String alg, String msg, String hash) that accepted a hashing algorithm, a message, and its hash. This function checked the integrity of the data and printed a message indicating whether the data was valid or not.

We added a function named sendMessage() to the Sender.java file. It created a Receiver object and sent the hashing algorithm, message, and hash to the verifyHash() function. This allowed us to test the integrity of transmitted data.

Authentication Using Hashing and MACs:

The lab provided a Java file named Macode.java, which we reviewed and ran. The code demonstrated the generation of message authentication codes (MACs) using the JCA MAC class. We entered different messages and observed the corresponding MACs.

In the Sender.java file, we added a function that allowed the user to send a message and its MAC to a receiver. This function generated a random symmetric key and used it to generate a MAC for the message.

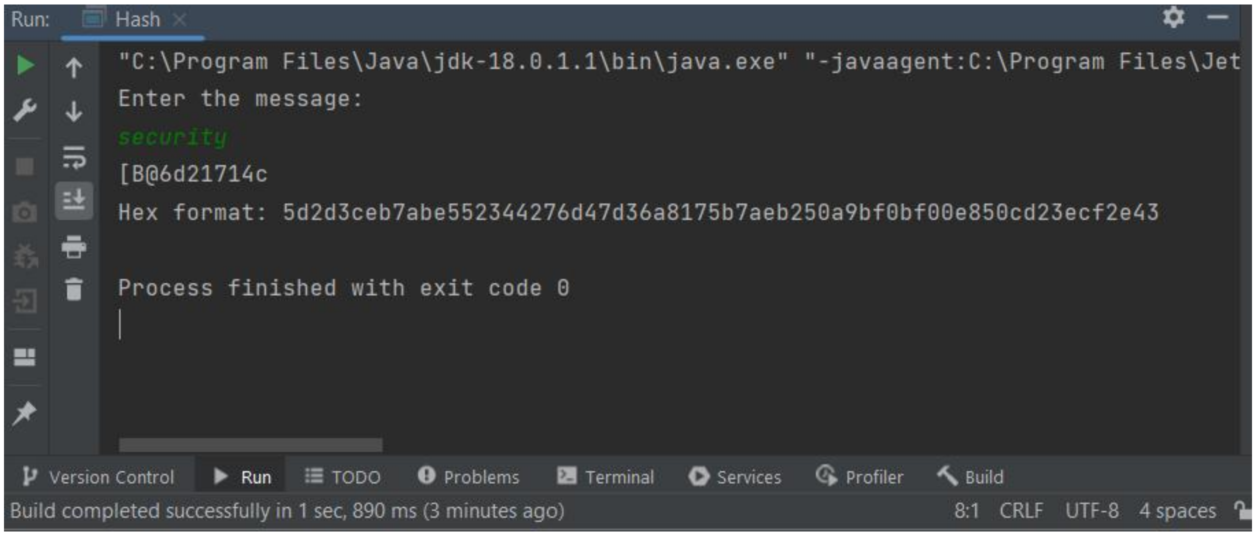
In the Receiver.java file, we added a function to verify the identity of the sender and the integrity of the message. This function received the message, MAC, and key and verified the MAC using the same key.

To send the key to the receiver, we can use various techniques, such as key exchange protocols or secure channels. In this lab, we assumed a pre-established secure channel between the sender and the receiver, where the key was shared securely before the message transmission.

Conclusion:

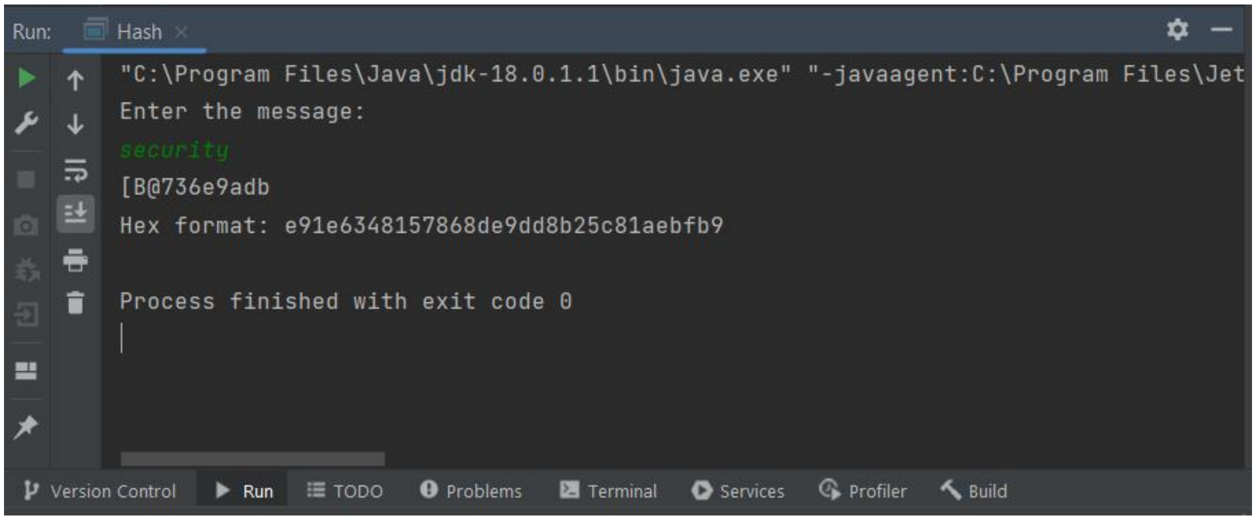
This lab provided hands-on experience with integrity and authentication functionalities using hashing techniques in Java. We explored different hashing algorithms, implemented data integrity checks, and used MACs for authentication. The lab helped understand the concepts of hashing, message digests, and message authentication codes, as well as their applications in ensuring data integrity and authentication in secure systems.

SHA-256



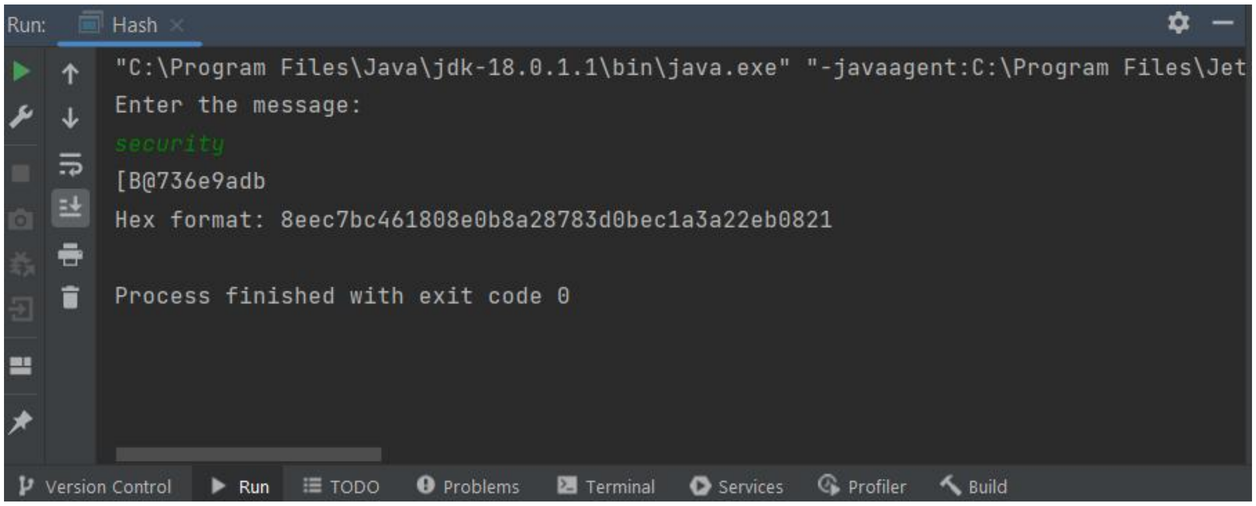
```
"C:\Program Files\Java\jdk-18.0.1.1\bin\java.exe" "-javaagent:C:\Program Files\Jet
Enter the message:
security
[B@6d21714c
Hex format: 5d2d3ceb7abe552344276d47d36a8175b7aeb250a9bf0bf00e850cd23ecf2e43

Process finished with exit code 0
```

MD5

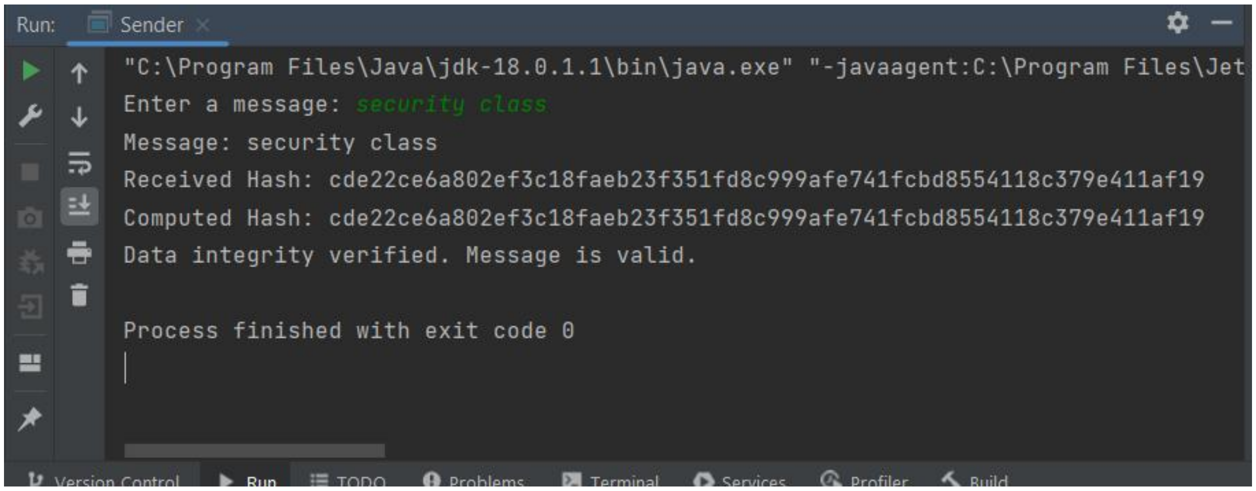


```
"C:\Program Files\Java\jdk-18.0.1.1\bin\java.exe" "-javaagent:C:\Program Files\Jet
Enter the message:
security
[B@736e9adb
Hex format: e91e6348157868de9dd8b25c81aebfb9

Process finished with exit code 0
```

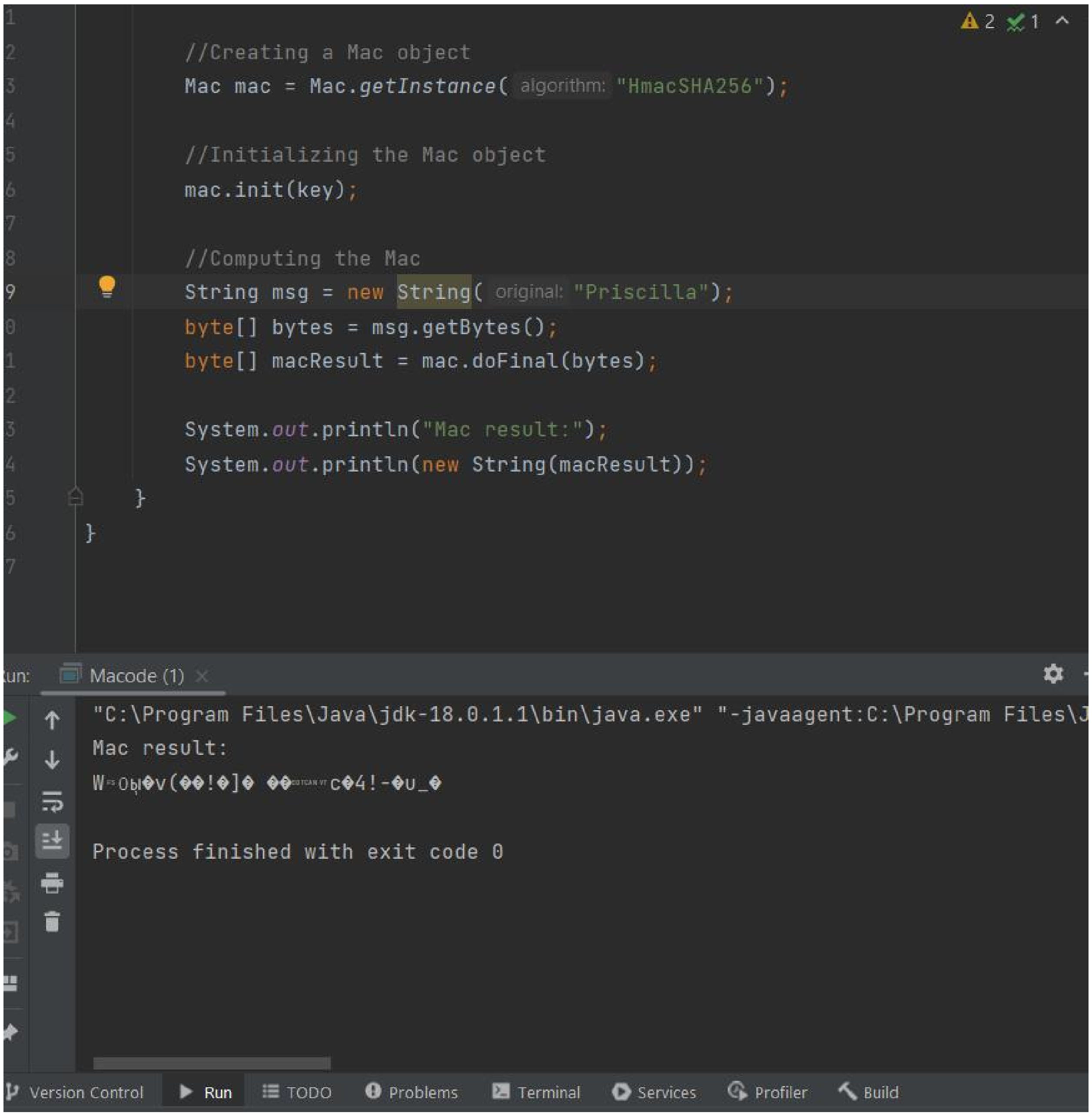
SHA-
1



```
"C:\Program Files\Java\jdk-18.0.1.1\bin\java.exe" "-javaagent:C:\Program Files\Jet
Enter the message:
security
[B@736e9adb
Hex format: 8eec7bc461808e0b8a28783d0bec1a3a22eb0821

Process finished with exit code 0
```

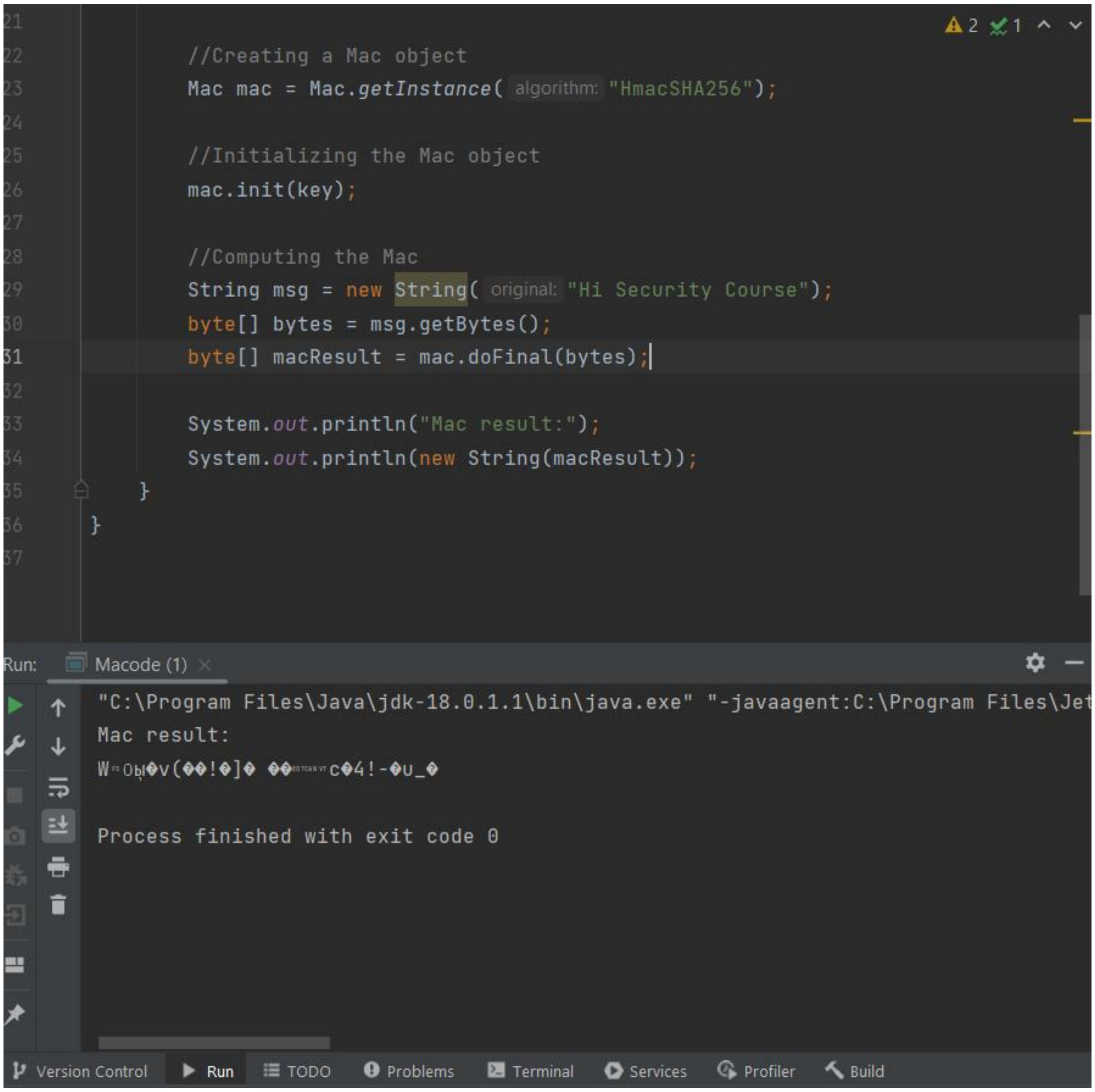


```
"C:\Program Files\Java\jdk-18.0.1.1\bin\java.exe" "-javaagent:C:\Program Files\Jet
Enter a message: security class
Message: security class
Received Hash: cde22ce6a802ef3c18faeb23f351fd8c999afe741fcbd8554118c379e411af19
Computed Hash: cde22ce6a802ef3c18faeb23f351fd8c999afe741fcbd8554118c379e411af19
Data integrity verified. Message is valid.

Process finished with exit code 0
```



```
        //Creating a Mac object
        Mac mac = Mac.getInstance( algorithm: "HmacSHA256");

        //Initializing the Mac object
        mac.init(key);

        //Computing the Mac
        String msg = new String( original: "Priscilla");
        byte[] bytes = msg.getBytes();
        byte[] macResult = mac.doFinal(bytes);

        System.out.println("Mac result:");
        System.out.println(new String(macResult));
    }
}
```

```
"C:\Program Files\Java\jdk-18.0.1.1\bin\java.exe" "-javaagent:C:\Program Files\J
Mac result:
W Obl�v(��!�]� �� c�4!-�u_�

Process finished with exit code 0
```



```java
        //Creating a Mac object
        Mac mac = Mac.getInstance( algorithm: "HmacSHA256");

        //Initializing the Mac object
        mac.init(key);

        //Computing the Mac
        String msg = new String( original: "Hi Security Course");
        byte[] bytes = msg.getBytes();
        byte[] macResult = mac.doFinal(bytes);

        System.out.println("Mac result:");
        System.out.println(new String(macResult));
    }
}
```

```
"C:\Program Files\Java\jdk-18.0.1.1\bin\java.exe" "-javaagent:C:\Program Files\Jet
Mac result:
W Oы�v(��!�]� �� c�4!-�u_�

Process finished with exit code 0
```

Regarding the question of how the key is sent to the receiver, in this example, the shared key is generated independently by the sender and receiver using the same algorithm (DES). The receiver generates the shared key in the generateSharedKey() method. The key is not explicitly sent from the sender to the receiver but is shared through a predetermined method (e.g., through a secure channel or prior agreement). In this case, the shared key is stored within the Receiver class and used for MAC verification.