King Fahd University of Petroleum and Minerals
College of Computer Science and Engineering
Information and Computer Science Department

SEC 511 - Principles of Information Assurance and Security

Second semester 2019-2020 (192)

Technical Homework #1 [10%]

Secured Instant Messenger

Student Name: Mohammed Alqmase

ID: g201531270

Table of Contents

1. Introduction

"In this homework, we need to implement a secure Instant Messaging program. This program should transmit short messages between a client and a server. The server process is instantiated first. At any time later, the client can initiate a session with the server. After establishing a session, the client and server can exchange messages". *[1]*

1.1 Purpose

The purpose is to understand the security requirements and to be familiar with the core security concepts which are confidentiality, integrity and authentication. In addition, the aim of this project is to apply different security techniques such as encryption using symmetric and asymmetric mechanisms for encryption, applying hashing, etc.

1.2 Project Scope

The scope of the project is to implement simple secure instant messaging where the server receives a message and send it as echo server. The proposed implementation should allow the client to communicate with server using the following options: No security, Confidentiality, Integrity, Authentication, or combinations. No security option means that the exchanged messages are transmitted in plain text. All cryptographic operations such as key exchange, public/private key generations, encryption, decryption, hashing, etc.) should be transparent to the client.

1.3 Software and Algorithms

In this project, java programming language, NetBeans, and Wireshark are used. NetBeans is IDE that helps to write java program. Wireshark is a software that helps to monitor and validate the communication between client and the server. We use RSA, AES and MD5 algorithms.
RSA stands for *Rivest-Shamir-Adleman* which used for encrypting and decrypting data using asymmetric keys. AES stands for Advanced Encryption Standard which used for encrypting and decrypting data using symmetric keys. MD5 stands for *Message Digest Algorithm which used to create a hash. Hashing is one-way encryption. [2]*

1.4 Limitations

This project is intended to utilities the OpenSSL library that provides security facilities to develop the project. For some reasons, I used other pure java library which is the Java Secure Socket Extension (JSSE). "It enables secure Internet communications. It provides a framework and an implementation for a Java version of the SSL, TLS, and DTLS protocols and includes functionality for data encryption, server authentication, message integrity, and optional client authentication." []. "The JSSE provide several functionalities includes functionality for data encryption, server authentication, message integrity, and optional client authentication. Using JSSE, developers can provide for the secure passage of data between a client and a server running any application protocol (such as HTTP, Telnet, or FTP) over TCP/IP. JSSE minimizes the risk of creating subtle but dangerous security vulnerabilities. The JSSE API supports the following security protocols: SSL version 3.0, TLS version 1.2, and DTLS versions 1.2". *[2]*

This project also is limited to the shallow testing. We focus on the testing of the client and server communication.

1.5 Assumptions and Dependencies

In this project, we assume that the key management have been achieve safely. We generate the private and the public keys and internally distribute them.

1.6 Background

There are three major goals of information security which are confidentiality, Integrity and Availability. Confidentiality is the goal of prevent an authorized user to access the information, Not only the stored information also the transmitted data. Integrity is the goal of prevent an authorized user to change or alter the data. It grantees a correct and constant data. Availability is the goal of make the information is available to the authorized entities. Some security services can be implemented using cryptography where it cares about secret writing using encryption and decryption services. Three different approaches are used for encryption and decryption. We can encrypt and decrypt data using symmetric-key, asymmetric-key, and hashing. Symmetric-key encipherment is the service of encrypt and decrypt the data using a shared key. One key can be used to encrypt and decrypt the message. Asymmetric key is the technique of using two keys one for encryption and the other for decryption. Hashing is used to validate the integrity of data by detecting if there and change to send/receive messages. The digest is a hash of a message that encrypted by the shared/secret key. The digital signature is a hash of the message that has been encrypted using public/private key. Both digest and digital signature can be used to check the integrity of a message. *[1]*

2. Requirement Description

In this section, the project is described using use cases and using some dialogs and interfaces.

2.1 Use Case

In this homework, we have the following use cases: login with no security requirements, login with security requirements, configure and establish the communication, generate keys, send and receive messages between client and server. Figure 1 shows the use case diagram to simplify the project requirements.

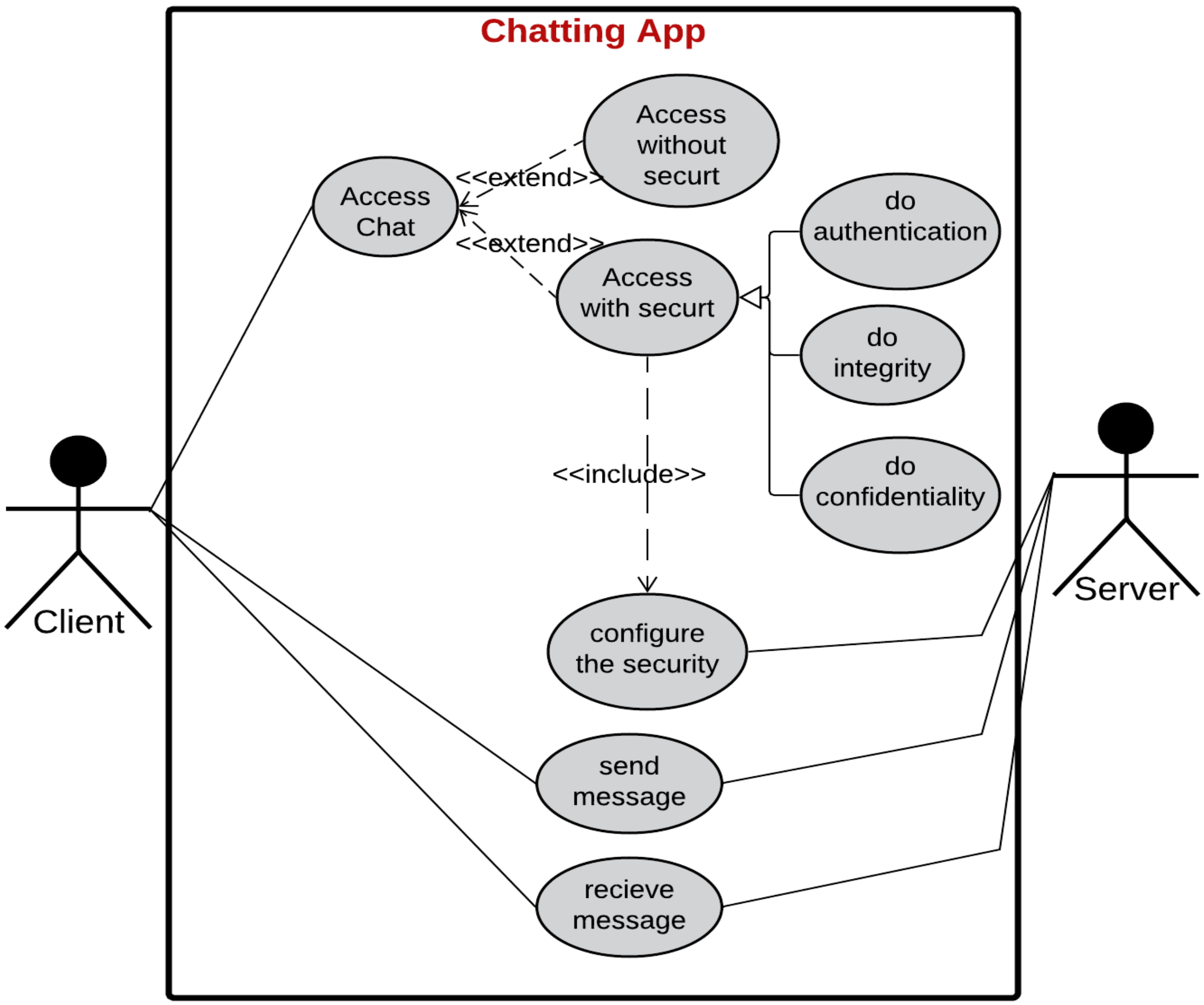


Figure 1. Describe the requirement using use case diagram

2.2 Design and Interfaces

In this section, implementation requirements is described by showing the required user interface panels, dialogs, buttons, labels and other required components. Figure 2 shows the login window and Figure 3 presents the chat room.

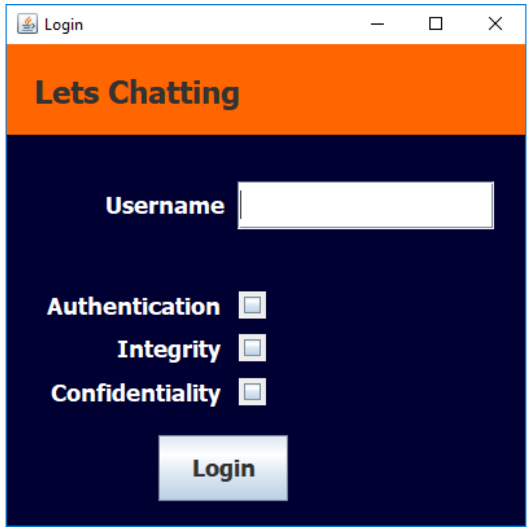


Figure 2. User interface for login functionality

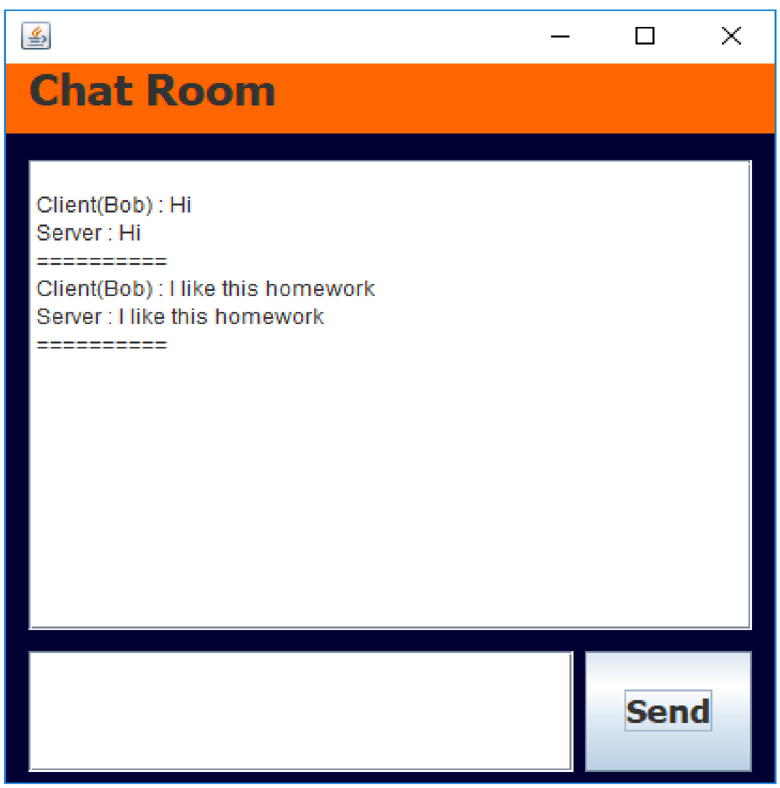


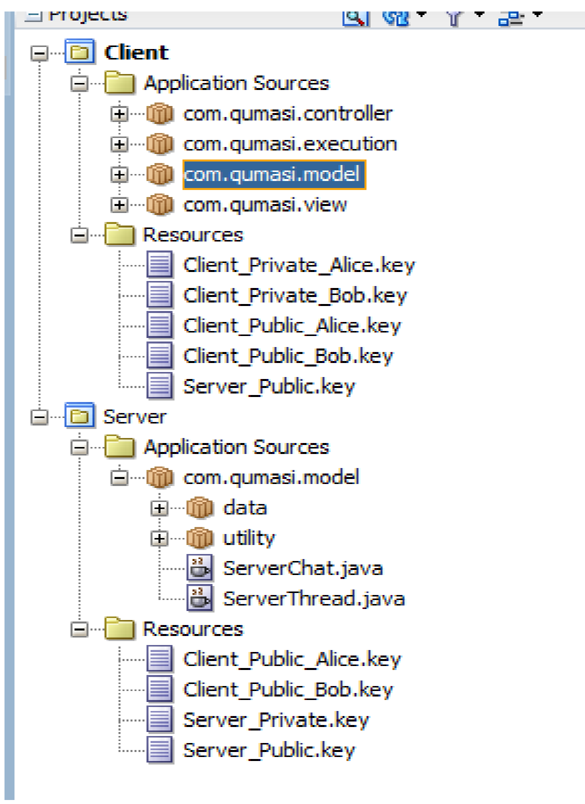
Figure 3. Simple user interface for chat room

3. Implementation

3.1 Generate Public and Private Keys

The following function is implemented to generate public/private pair keys. In this code, we use the RSA algorithm generator to generate the required keys.

```
private void generatePairKeys() {
    try {
        KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
        keyPairGenerator.initialize(2028);
        KeyPair keyPair = keyPairGenerator.genKeyPair();
        PublicKey publickey = keyPair.getPublic();
        PrivateKey privateKey = keyPair.getPrivate();
        KeyFactory kedFactory = KeyFactory.getInstance("RSA");
        RSAPublicKeySpec rsaPublicKeySpec = kedFactory.getKeySpec(publickey, RSAPublicKeySpec.class);
        RSAPrivateKeySpec rsaPrivateKeySpec = kedFactory.getKeySpec(privateKey, RSAPrivateKeySpec.class);
        saveKeys(PUBLIC_KEY_FILE, rsaPublicKeySpec.getModulus(), rsaPublicKeySpec.getPublicExponent());
        saveKeys(PRIVATE_KEY_FILE, rsaPrivateKeySpec.getModulus(), rsaPrivateKeySpec.getPrivateExponent());
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    } catch (InvalidKeySpecException e) {
        e.printStackTrace();
    }
}
```



3.2 Generate Shared Key

The following function is implemented to generate shared/secret key. In this code, we use the AES algorithm generator to generate the required shared key.

```
public static SecretKey generateSharedKey() {
    try {
        KeyGenerator generator = KeyGenerator.getInstance("AES");
        generator.init(128); // The AES key size in number of bits
        SecretKey secKey = generator.generateKey();
        return secKey;
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    }
    return null;
}
```

3.3 Generate Hash

The following function is implemented to generate a hash of a given message.

```
public static byte[] hashing(String data) {
    String dataToHash = data;
    try {
        MessageDigest md = MessageDigest.getInstance("MD5");
        md.update(dataToHash.getBytes());
        byte[] digest = md.digest();
        return digest;
    } catch (NoSuchAlgorithmException e) {
    }
    return null;
}
```

3.4 Encrypt and Decrypt using Public/Private Keys

The following function is implemented to encrypt the data using Private Key.

```java
public static byte[] encriptDataByPrivateKey(byte[] dataToEncrpt, String privateKey) {
    byte[] encrptedData = null;
    try {
        PrivateKey priKey = KeyStore.readPrivateKeyFromFile(privateKey);
        Cipher cipher;
        cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.ENCRYPT_MODE, priKey);
        encrptedData = cipher.doFinal(dataToEncrpt);
    } catch (NoSuchPaddingException e) {
        e.printStackTrace();
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    } catch (InvalidKeyException e) {
        e.printStackTrace();
    } catch (BadPaddingException e) {
        e.printStackTrace();
    } catch (IllegalBlockSizeException e) {
        e.printStackTrace();
    }
    return encrptedData;
}
```

The following function is implemented to decrypt the data using public Key.

```java
public static byte[] decriptDataByPublicKey(byte[] dataToDecrpt, String publicKey) {
    byte[] decrptedData = null;
    try {
        PublicKey pubKey = KeyStore.readPublicKeyFromFile(publicKey);
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.DECRYPT_MODE, pubKey);
        decrptedData = cipher.doFinal(dataToDecrpt);
    } catch (NoSuchPaddingException e) {
        e.printStackTrace();
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    } catch (InvalidKeyException e) {
        e.printStackTrace();
    } catch (BadPaddingException e) {
        e.printStackTrace();
    } catch (IllegalBlockSizeException e) {
        e.printStackTrace();
    }
    return decrptedData;
}
```

3.5 Encrypt and Decrypt using shared Key

The following function is implemented to encrypt the data using shared Key.

```java
public static byte[] encriptBySharedKey(byte[] dataToEncript, SecretKey secKey) {
    try {
        Cipher aesCipher = Cipher.getInstance("AES");
        aesCipher.init(Cipher.ENCRYPT_MODE, secKey);
        byte[] encriptedData = aesCipher.doFinal(dataToEncript);
        return encriptedData;
    } catch (NoSuchPaddingException e) {
        e.printStackTrace();
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    } catch (InvalidKeyException e) {
        e.printStackTrace();
    } catch (BadPaddingException e) {
        e.printStackTrace();
    } catch (IllegalBlockSizeException e) {
        e.printStackTrace();
    }
    return null;
}
```

The following function is implemented to decrypt the data using public Key.

```java
public static byte[] decriptBySharedKey(byte[] dataToDecript, SecretKey secKey) {
    try {
        Cipher aesCipher = Cipher.getInstance("AES");
        aesCipher.init(Cipher.DECRYPT_MODE, secKey);
        byte[] decriptedData = aesCipher.doFinal(dataToDecript);
        return decriptedData;
    } catch (NoSuchPaddingException e) {
        e.printStackTrace();
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    } catch (InvalidKeyException e) {
        e.printStackTrace();
    } catch (BadPaddingException e) {
        e.printStackTrace();
    } catch (IllegalBlockSizeException e) {
        e.printStackTrace();
    }
    return null;
}
```

3.6 Login with No Security Requirements

In this option, we allow any user to join the chat room and communicate with server without any encryption and decryption mechanism where the data send and receive as plain text.

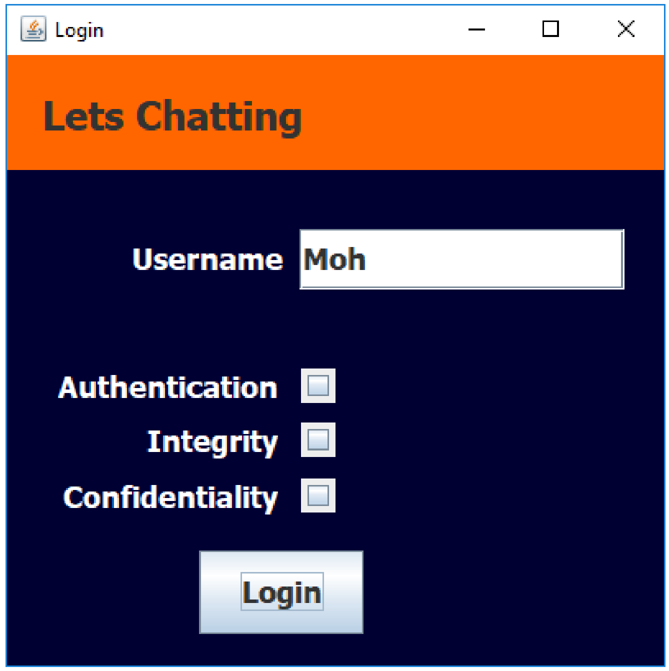


Figure 4. Login Window

The following figures present the received and the sent message between client and server. Figure 5 shows the data sent form client to server, and Figure 6 shows the data transferred from server to client.

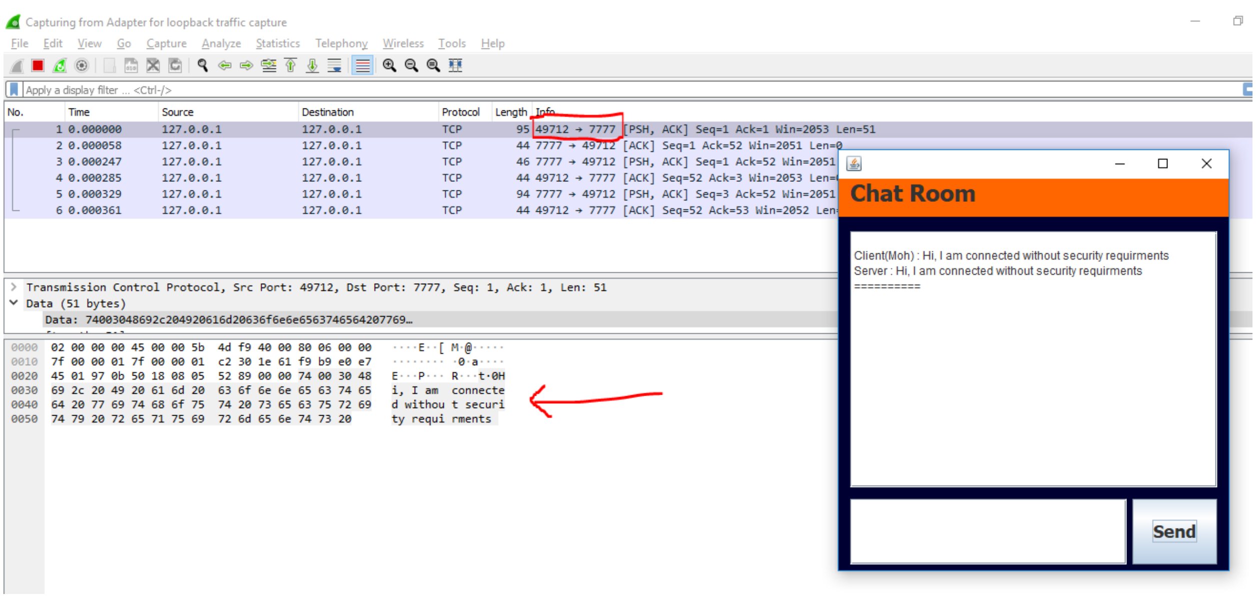


Figure 5. Wireshark screenshot to show the transmitted data with no security requirement

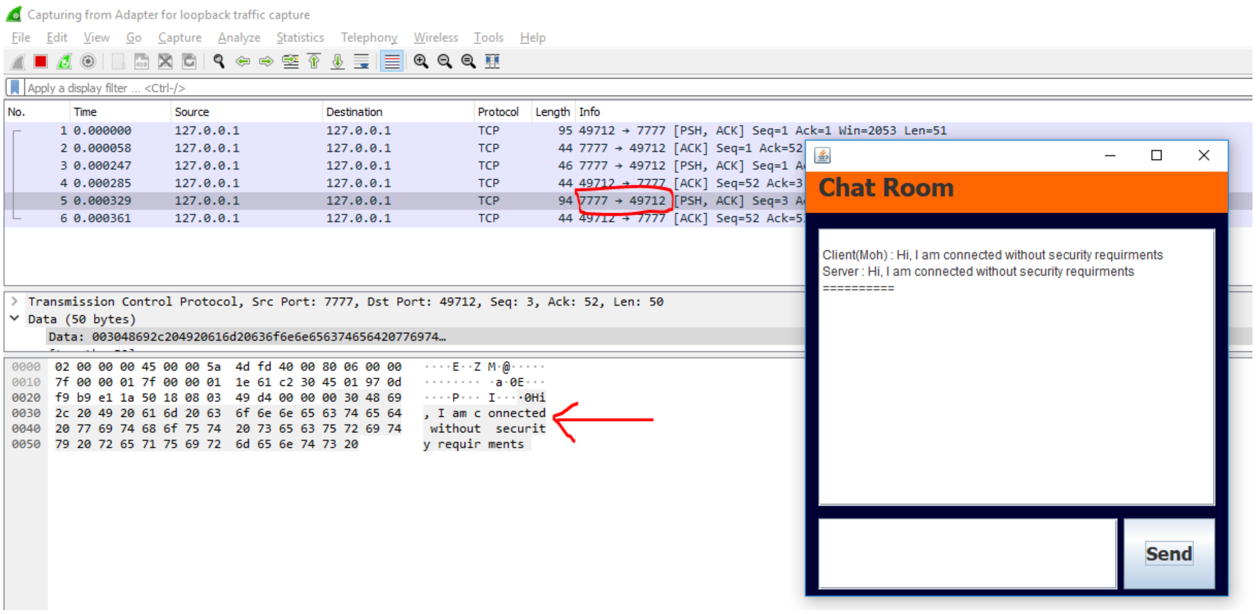


Figure 6.Wireshark screenshot to show the transmitted data with no security requirement 2

3.7 Login with Confidentiality Security Requirements

The following scenario shows the proposed solution to allow the client and the server to communication with only confidentiality. The following steps describe how the confidentiality is implemented:

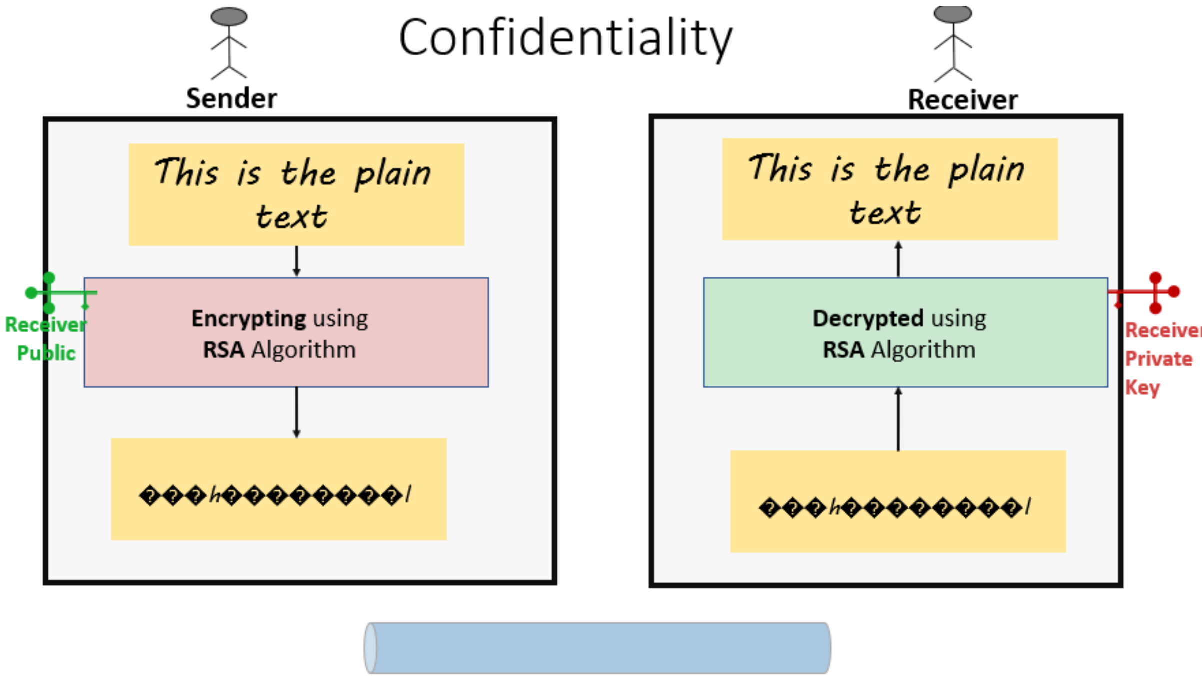
Client
======

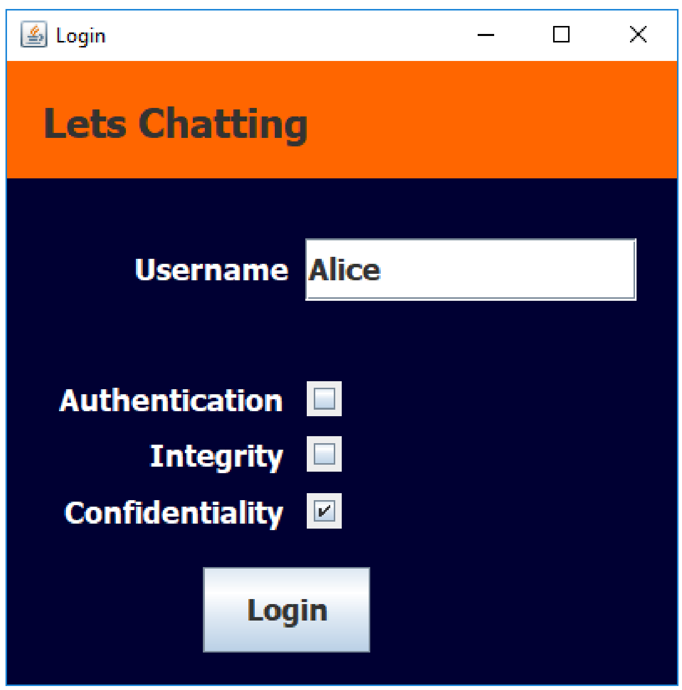
1. *Use asymmetric key to encrypt and decrypt message using with RSA algorithm.*
2. *Encrypt the Message (M) using the server public key.*
3. *Send the Encrypted Message E(M) to the server.*
4. *Wait to receive the encrypted echo from the server.*
5. *Receive the encrypted echo from the server.*
6. *Decrypt the received message using the client private key.*
7. *Display the decrypted message in the chat area.*

Server
======

1. *Use asymmetric key to encrypt and decrypt message using with RSA algorithm.*
2. *Receive the Encrypted Message E(M) from the client.*
3. *Decrypt the received message using the server private key. M=D(E(M)).*
4. *Encrypt the Message (M) using the client private key.*
5. *Send the Encrypted Message E(M) to client.*
6. *Wait for next Message.*

The following figure describes how the confidentiality is implemented.





The following figures present the received and the sent message between client and server. Figure 7 shows the data sent form client to server, and Figure 8 shows the data transferred from server to client.

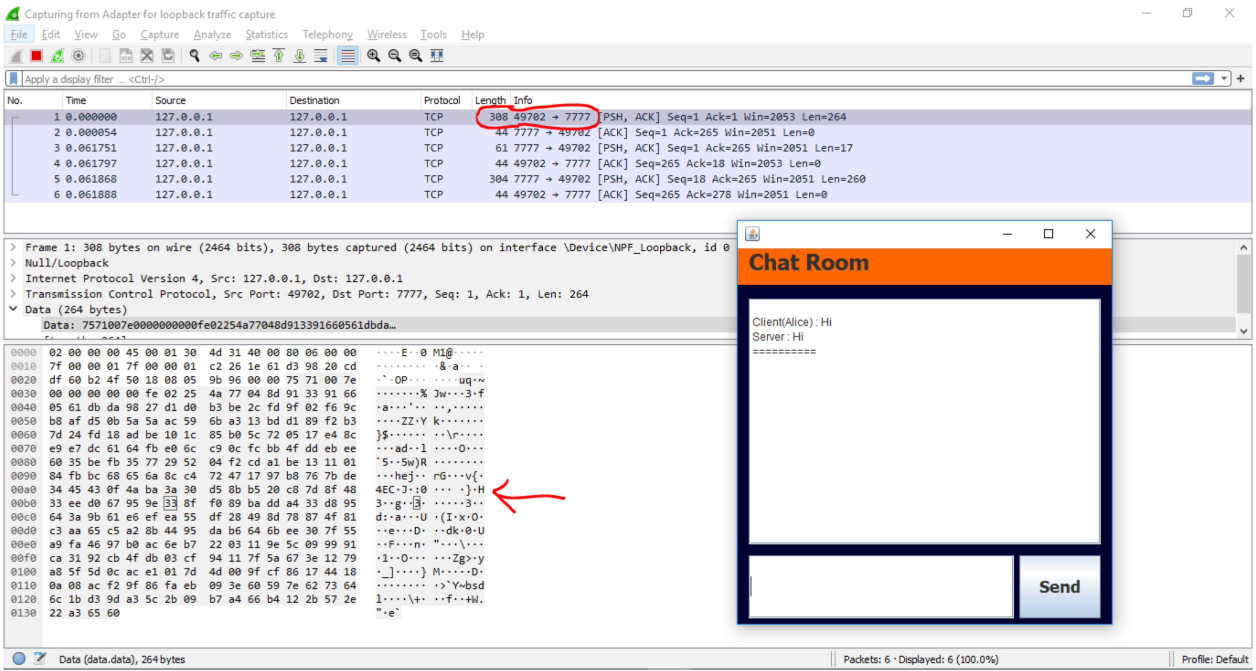


Figure 7. Wireshark screenshot to show the transmitted data with confidentiality

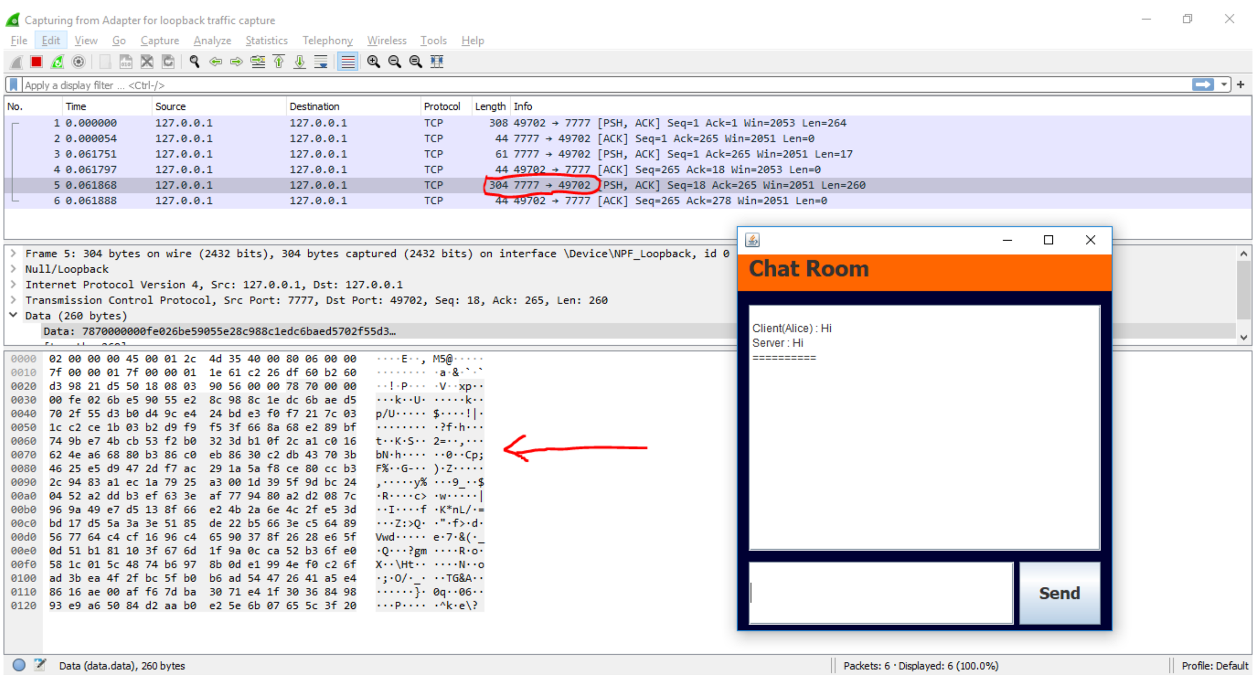


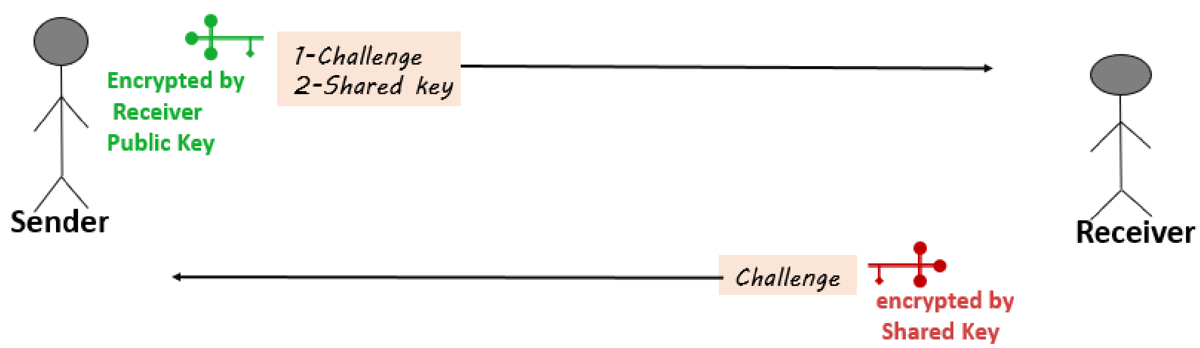
Figure 8. Wireshark screenshot to show the transmitted data with confidentiality 2

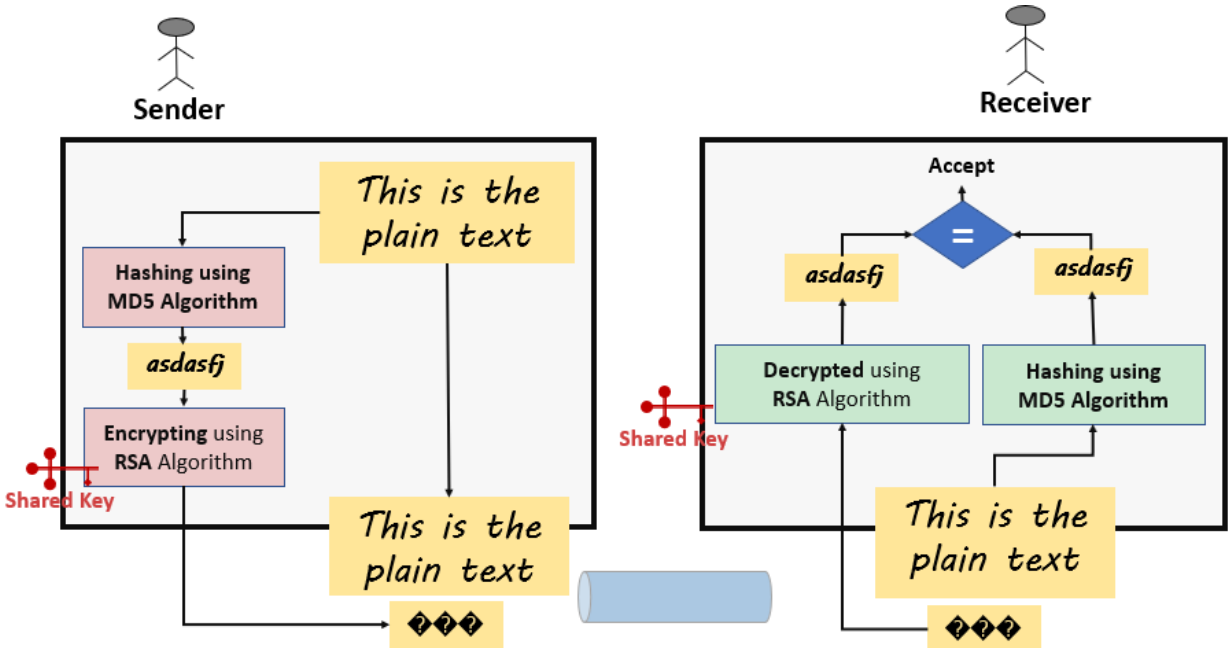
3.8 Login with Integrity Security Requirements

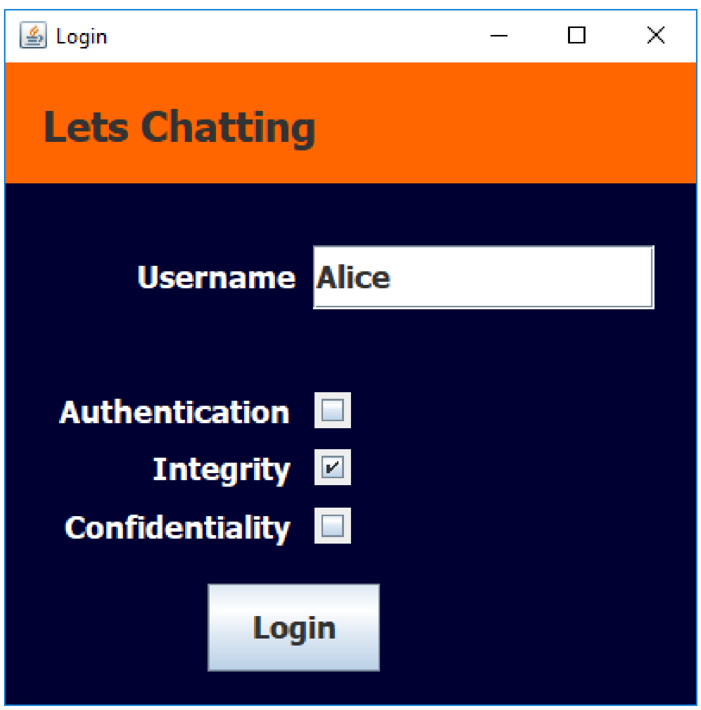
The following scenario shows the proposed solution to allow the client and the server to communication with only confidentiality. The following steps describe how the confidentiality is implemented:

1. ***Client****: Generate a symmetric key using AES algorithm.*
2. ***Client****: Generate a challenge (Ex. HOLLO).*
3. ***Client****: Use the server public to encrypt the compound token: (challenge + the generated secret key).*
4. ***Client****: Send the encrypted token: (challenge + the generated secret key) to the server.*
5. ***Server****: Receive the encrypted token: (challenge + the generated secret key).*
6. ***Server****: Decrypt the token: (challenge + the generated secret key) using the server private key.*
7. ***Server****: Use the secret key to encrypt the same challenge that was sent by client.*
8. ***Server****: Send the encrypted challenge to the client.*
9. ***Client****: Receive the encrypted challenge.*
10. ***Client****: Decrypt the received challenge using the generated secret key.*
11. ***Client****: Validate the received challenge by compare it to the one that was sent.*
12. ***Client*** *and Server: After the challenge was validated, use the secret key to send and receive messages.*
13. *done :)*

The following figure describes how the confidentiality is implemented.







The following figures present the received and the sent messages between client and server. Figure 9 shows the data sent form client to server, and Figure 10 shows the data transferred from server to client.

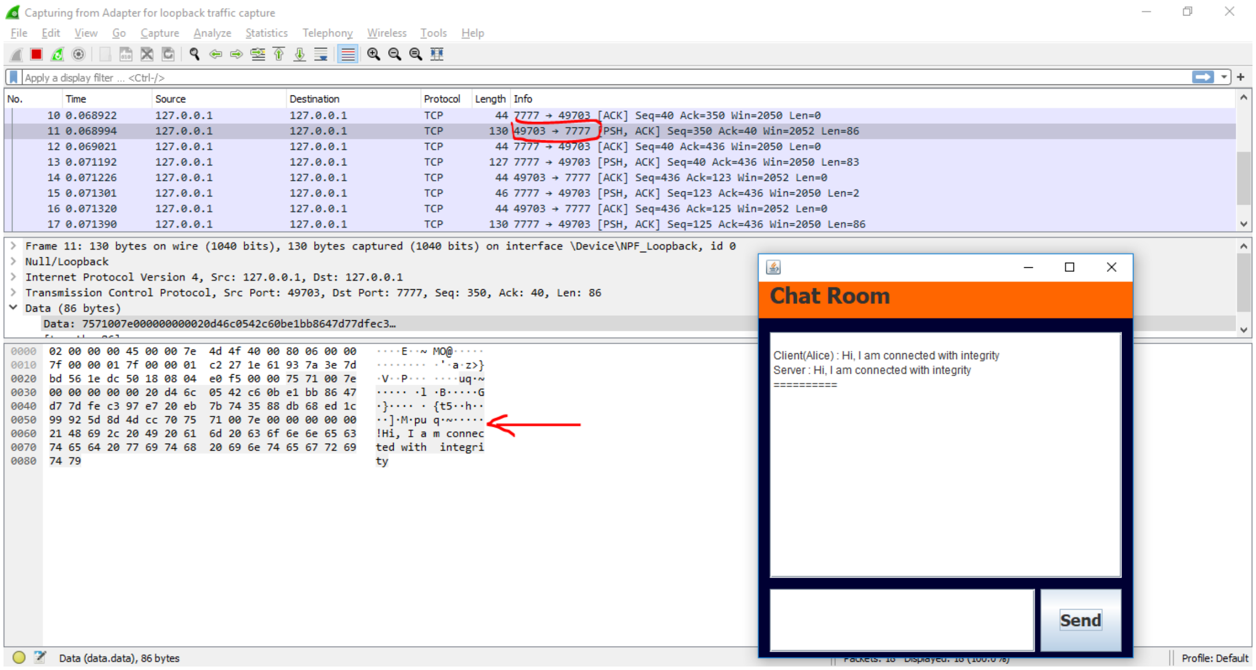


Figure 9. Wireshark screenshot to show the transmitted data with integrity 1

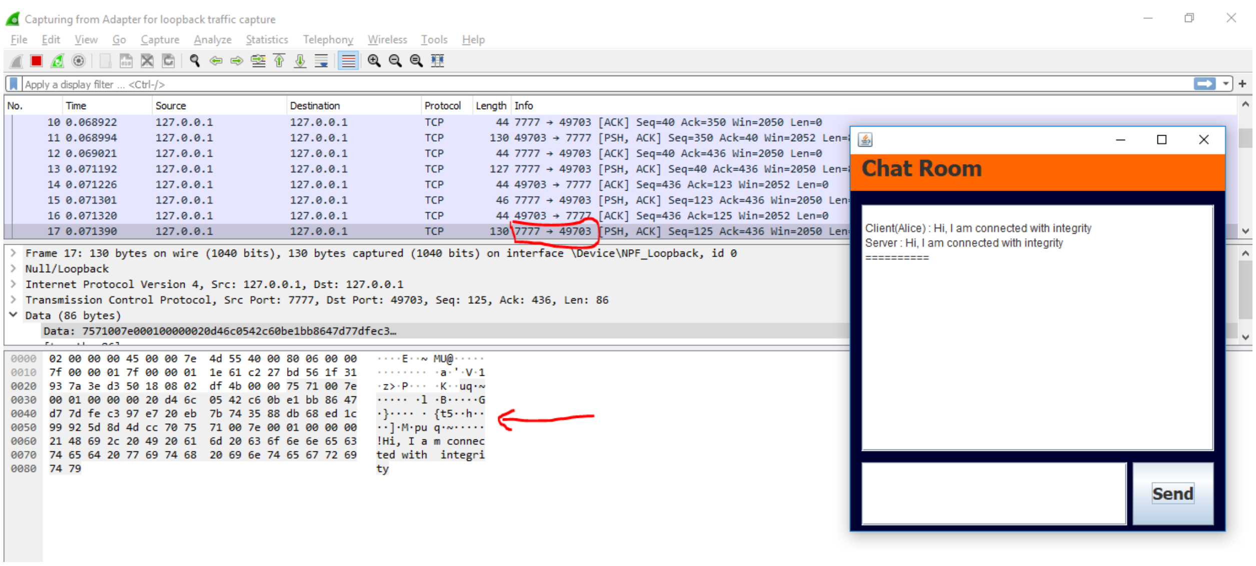


Figure 10. Wireshark screenshot to show the transmitted data with integrity 2

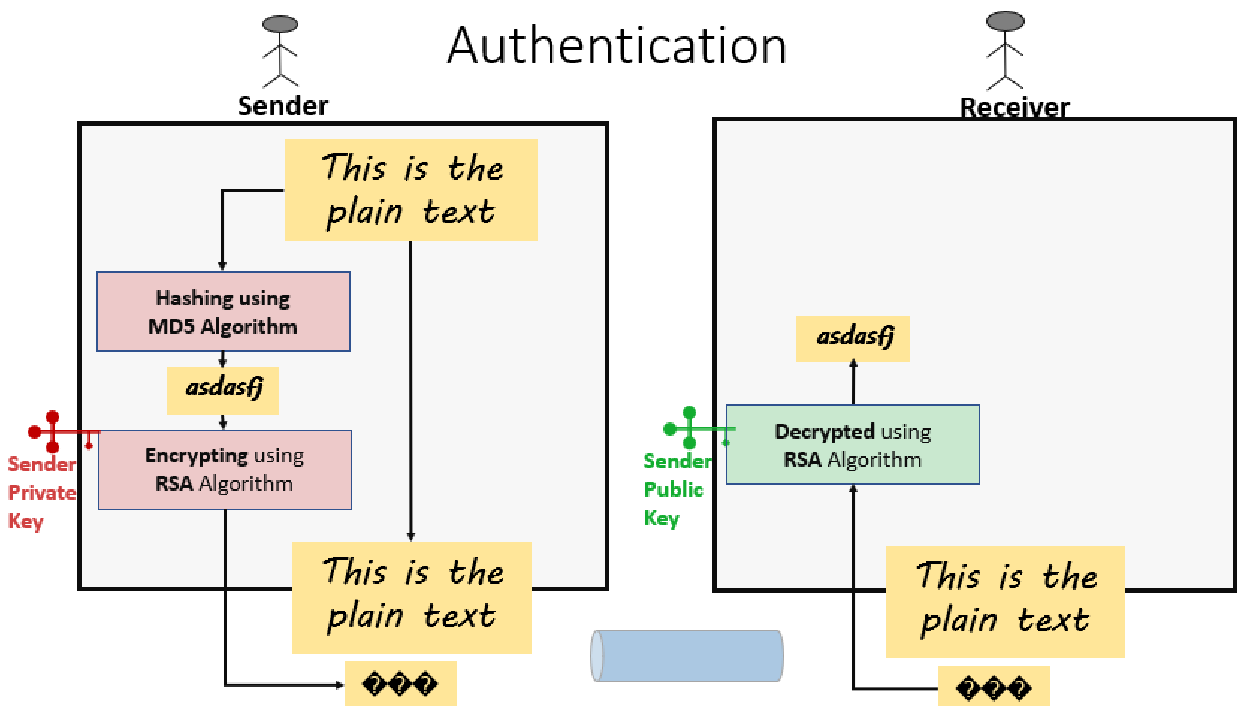
3.9 Login with Authentication Security Requirements

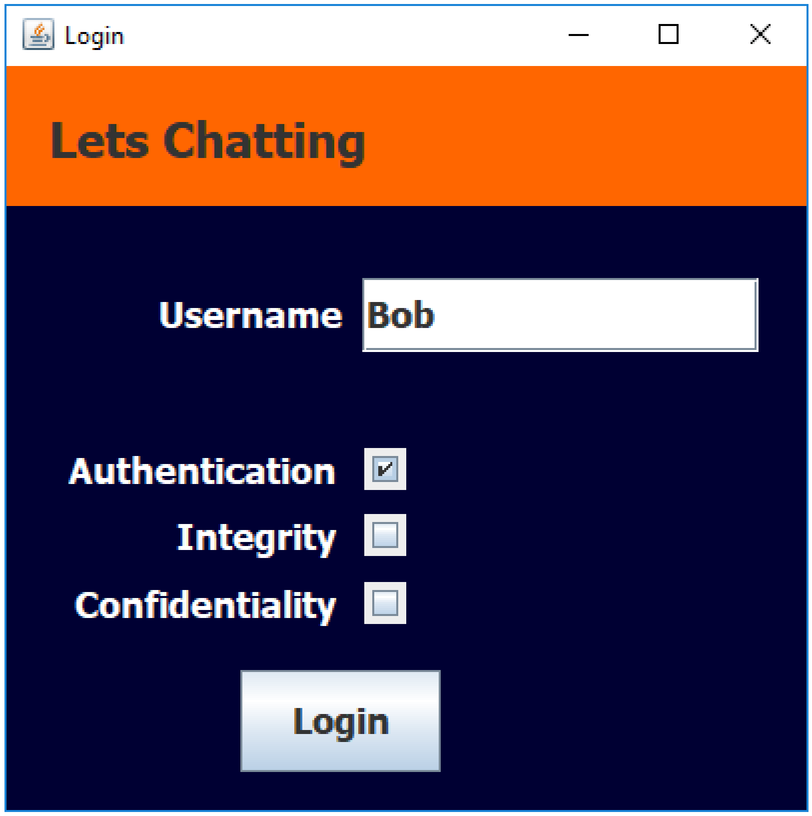
The following scenario shows the proposed solution to allow the client and the server to communication with only confidentiality. The following steps describe how the confidentiality is implemented:

Use asymmetric key to encrypt and decrypt message using with RSA algorithm.

1. ***Client***: *Write message*
2. ***Client***: *Generate hash from the message using (MD5) hash function.*
3. ***Client***: *Encrypt the hash using the client private key to create Digital Signature.*
4. ***Client***: *Send data that contains both (the message and the digital signature) to the server.*
5. ***Server***: *Receive data from client.*
6. ***Server***: *Decrypt the digital signature using the client public key to get the encrypted hash and authenticate the client.*
7. ***Server***: *Done ☺ no Integrity check is needed.*
8. ***Server***: *Generate hash from the message using (MD5) hash function.*
9. ***Server***: *Encrypt the hash using the server private key to create Digital Signature.*
10. ***Server***: *Send data that contains both (the message and the digital signature) to the client.*
11. ***Client***: *Receive echo from client.*
12. ***Client***: *Decrypt the digital signature using the server public key to get the encrypted hash and authenticate the server.*
13. ***Client***: *Done ☺ no Integrity check is needed.*
14. ***Client***: *Display the received message in the chat area.*

The following figure describes how the authentication is implemented.





The following figures present the received and the sent message between client and server. Figure 11 shows the data sent form client to server.

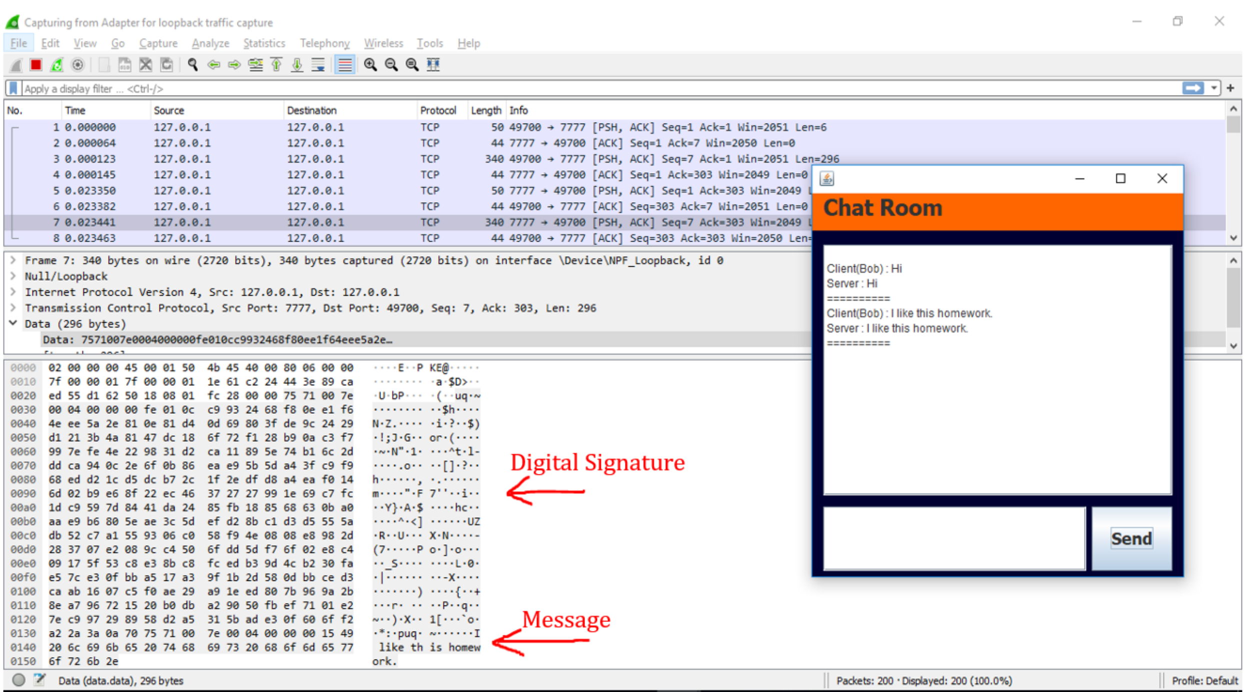
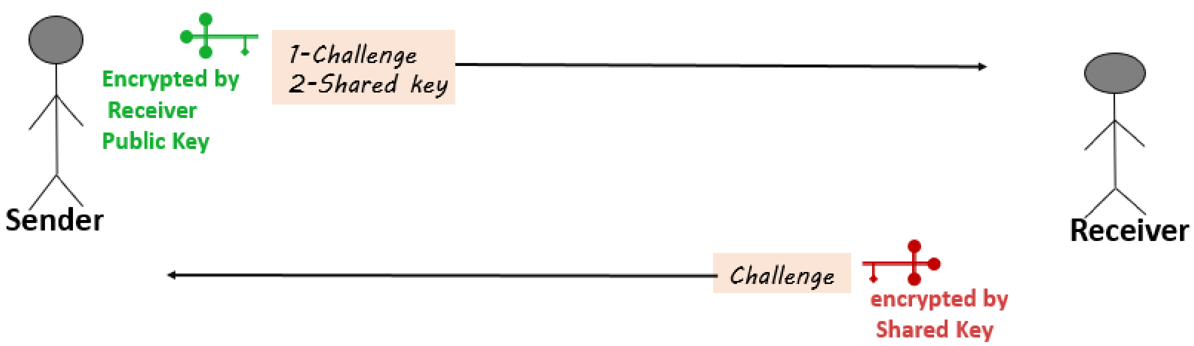


Figure 11. Wireshark screenshot to show the transmitted data with authentication 1

3.10 Login with Confidentiality and Integrity

The following scenario shows the proposed solution to allow the client and the server to communication with only confidentiality. The following steps describe how the confidentiality is implemented:

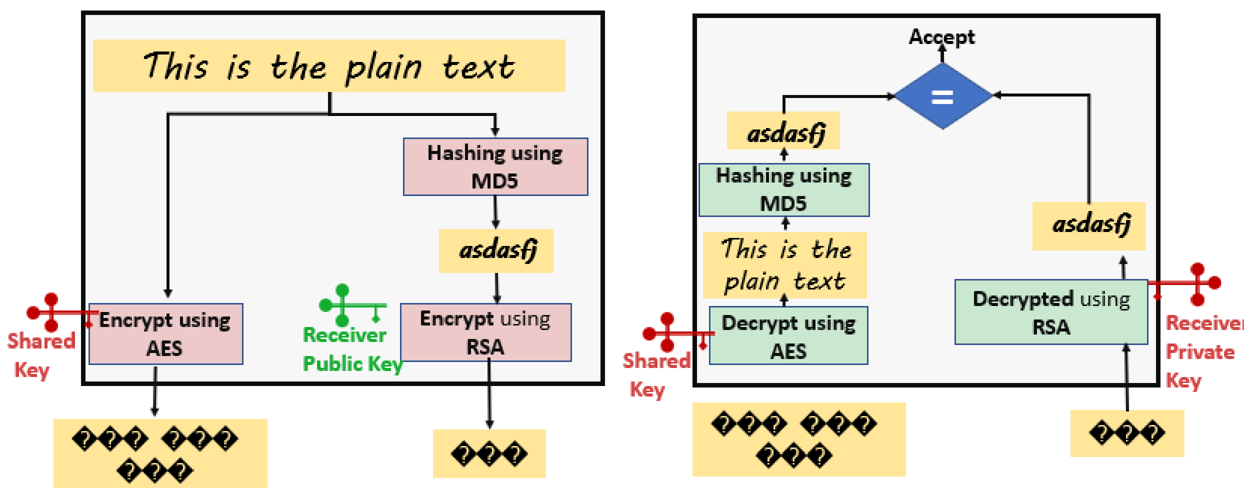
1. **Client**: Generate a symmetric key using AES algorithm.
2. **Client**: Generate a challenge (Ex. HOLLO).
3. **Client**: Use the server public to encrypt the compound token: (challenge + the generated secret key).
4. **Client**: Send the encrypted token: (challenge + the generated secret key) to the server.
5. **Server**: Receive the encrypted token: (challenge + the generated secret key) .
6. **Server**: Decrypt the token: (challenge + the generated secret key) using the server private key.
7. **Server**: Use the secret key to encrypt the same challenge that was sent by client.
8. **Server**: Send the encrypted challenge to the client.
9. **Client**: Receive the encrypted challenge.
10. **Client**: Decrypt the received challenge using the generated secret key.
11. **Client**: Validate the received challenge by compare it to the one that was sent.

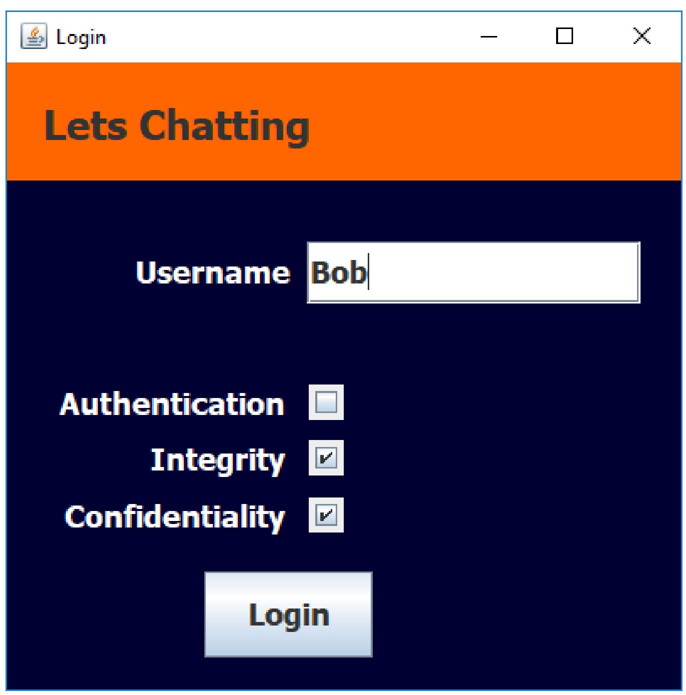


12. **Client**: After the challenge was validated, start chatting.
13. **Client**: Write message.
14. **Client**: Generate the hash form the message.
15. **Client**: Encrypt the hash using the server public key.
16. **Client**: Encrypt the message using the secret key.
17. **Client**: Send both the encrypted hash and the encrypted message to server.
18. **Server**: Receive the encrypted hash and the encrypted message.
19. **Server**: Decrypt the hash using the server private key.
20. **Server**: Decrypt the message using the secret key.
21. **Server**: Generate the hash from the decrypt message.
22. **Server**: Validate integrity by compare both the generated hash and the decrypted hash.
23. **Server**: Encrypt the generated hash using the client public key.
24. **Server**: Encrypt the message using the secret key.
25. **Server**: Send both the encrypted hash and the encrypted message to client.

26. **Client**: Receive the encrypted hash and the encrypted message.
27. **Client**: Decrypt the hash using the client private key.
28. **Client**: Decrypt the message using the secret key.
29. **Client**: Generate the hash from the decrypt message.
30. **Client**: Validate integrity by compare both the generated hash and the decrypted hash.

The following figure describes how the confidentiality and integrity is implemented.





The following figures present the received and the sent message between client and server. Figure 12 shows the data sent form client to server, and Figure 13 shows the data transferred from server to client.

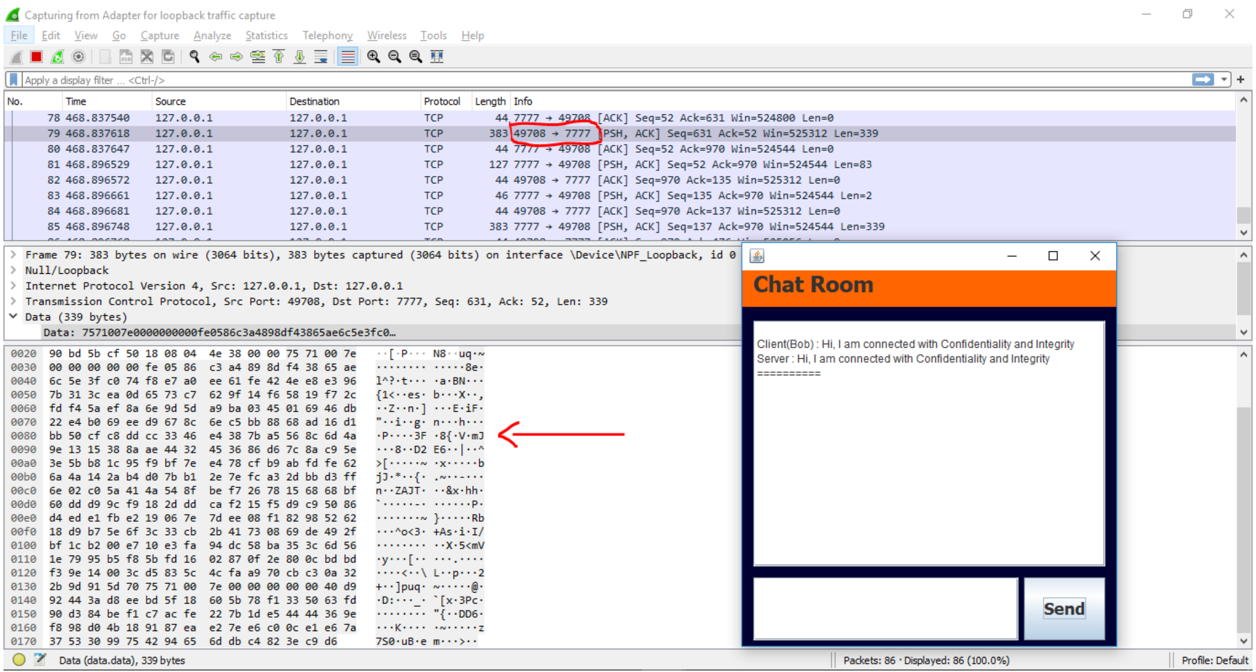


Figure 12. Wireshark screenshot to show the transmitted data with confidentiality and integrity 1

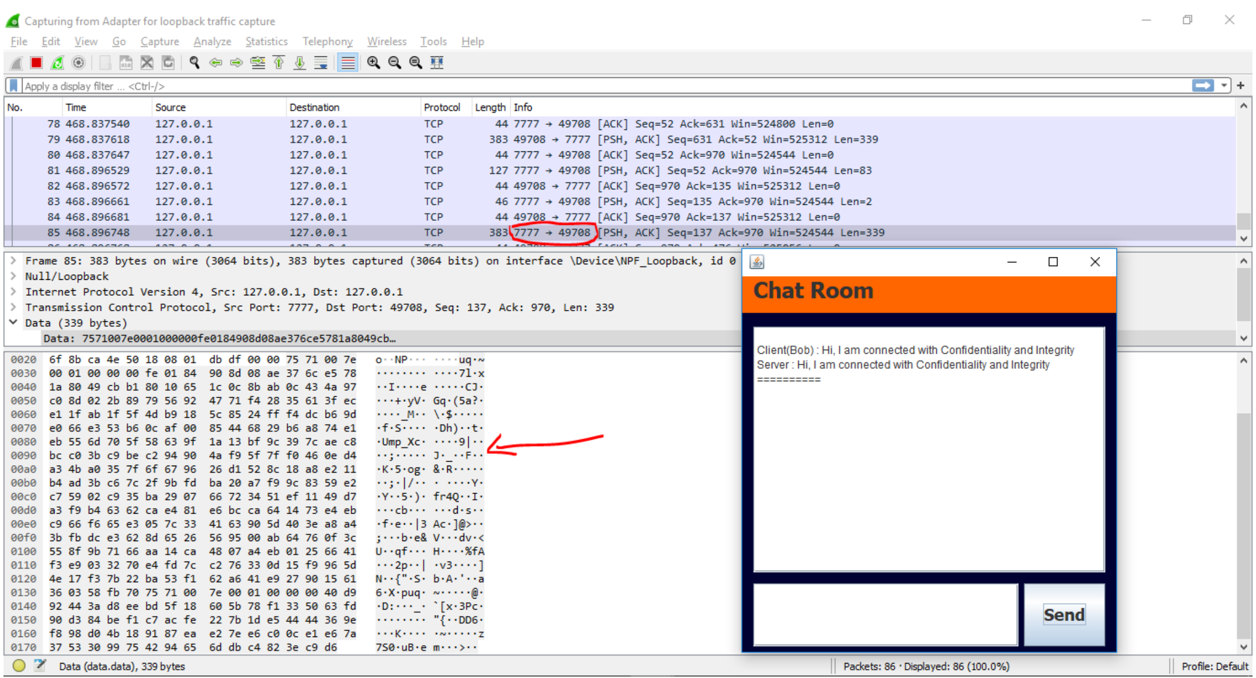


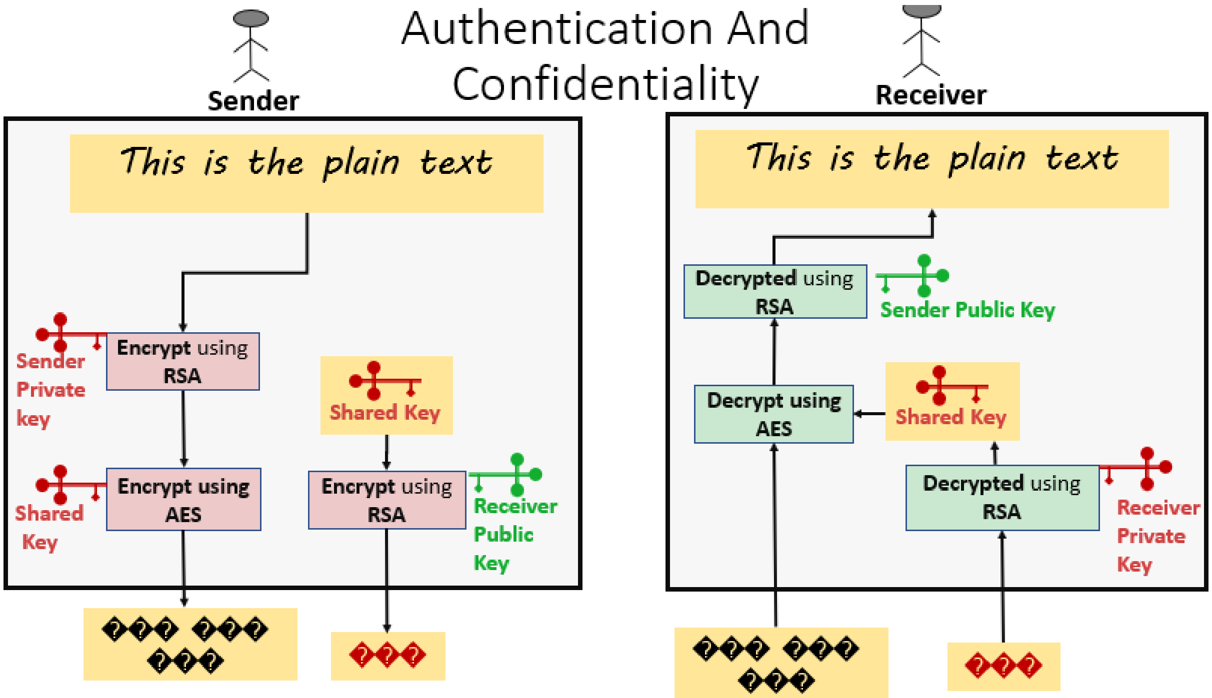
Figure 13. Wireshark screenshot to show the transmitted data with confidentiality and integrity 1

3.11 Login with Authentication and Confidentiality

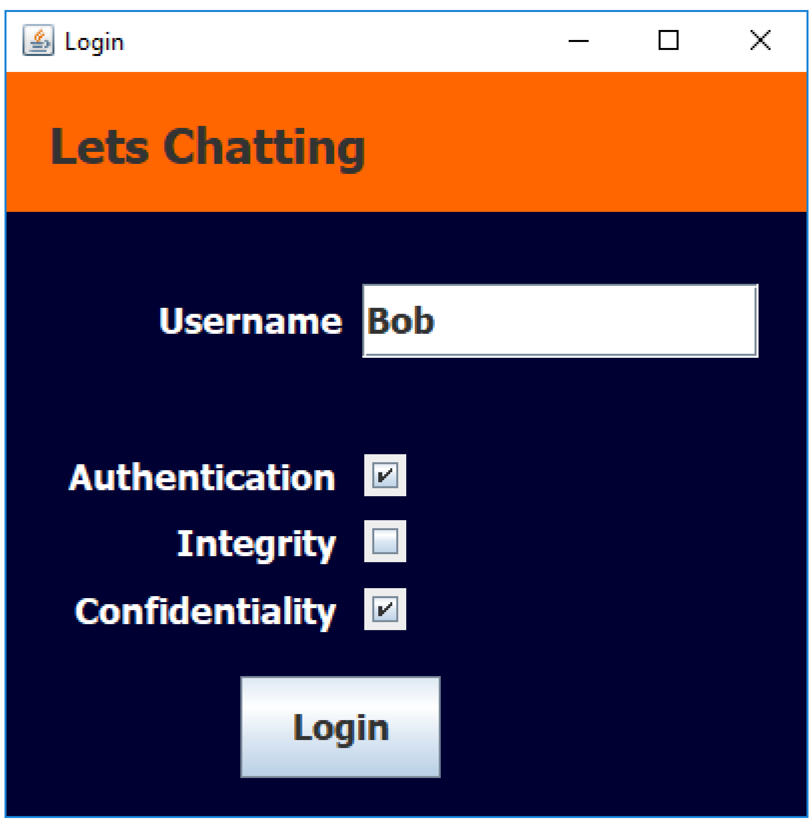
The following scenario shows the proposed solution to allow the client and the server to communication with only confidentiality. The following steps describe how the confidentiality is implemented:

1. **Client**: Generate a symmetric key using AES algorithm.
2. **Client**: Writing Message.
3. **Client**: Encrypt the message using the Client Private Key (CPK) for authentication (The encrypted message ->CPK(M))
4. **Client**: Encrypt the message AES(SPK(M)) using the generated secret key where the message gets two nested encryptions: one using the client private key, then the result is encrypted again using the generated secret key. Now, the message contains information about authentication and validate confidentiality.
5. **Client**: Encrypt the generated secret key using the Server public key.
6. **Client**: Send the encrypted message and the encrypted secret key.
7. **Server**: Receive the encrypted secret key and the encrypted message.
8. **Server**: Decrypt the secret key using the server private key.
9. **Server**: Use the decrypted secret key to decrypt the received message.
10. **Server**: Decrypt again the message using the client public key which help to authenticate the client.
11. **Server**: To do echo message.
12. **Server**: Encrypt the message using the Server Private Key (SPK) for authentication (The encrypted message ->SPK(M)).
13. **Server**: Encrypt the message AES(SPK(M)) using the secret key.
14. **Server**: Send the message to the client.
15. **Client**: Receive the encrypted message.
16. **Client**: Use the secret key to decrypt the received message.
17. **Client**: Decrypt again the message using the server public key which help to authenticate the server.
18. **Client**: display the message into the chat area.

The following figure describes how the authentication and confidentiality is implemented.



Authentication And Confidentiality
Sender
Receiver
This is the plain text
Sender Private key
Encrypt using RSA
Shared Key
Encrypt using AES
Shared Key
Encrypt using RSA
Receiver Public Key
This is the plain text
Decrypted using RSA
Sender Public Key
Decrypt using AES
Shared Key
Decrypted using RSA
Receiver Private Key



Login
Lets Chatting
Username
Bob
Authentication
Integrity
Confidentiality
Login

The following figures present the received and the sent message between client and server. Figure 14 shows the data sent form client to server, and Figure 15 shows the data transferred from server to client.

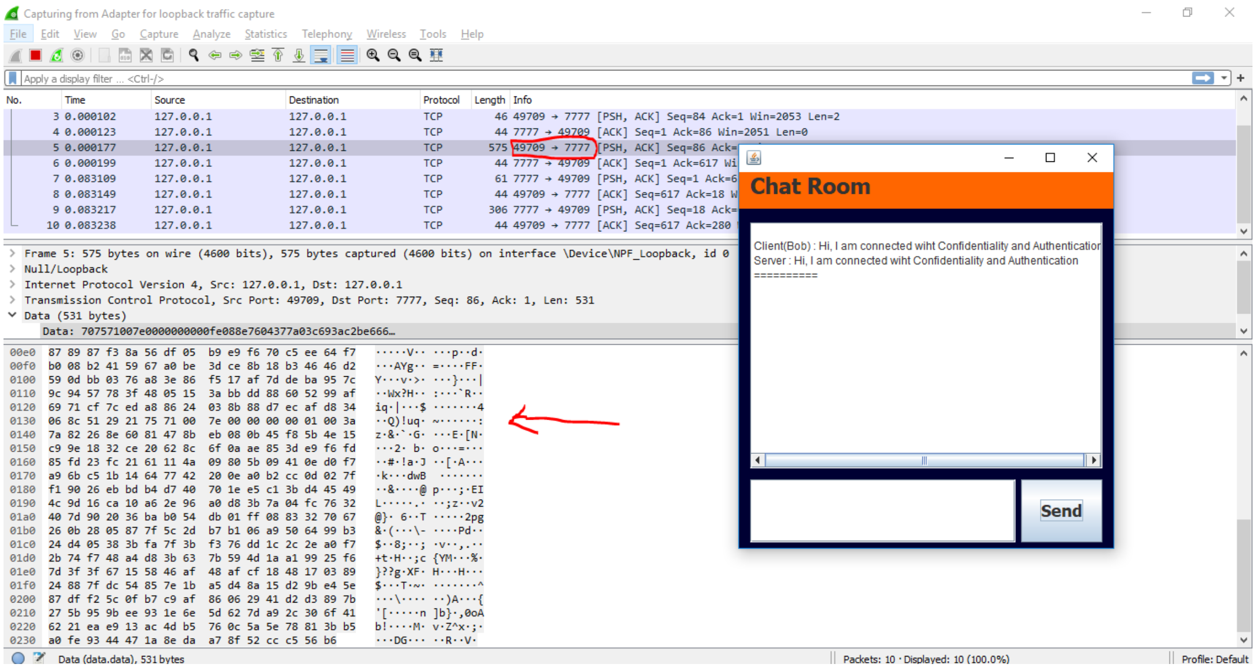


Figure 14. Wireshark screenshot to show the transmitted data with confidentiality and authentication 1

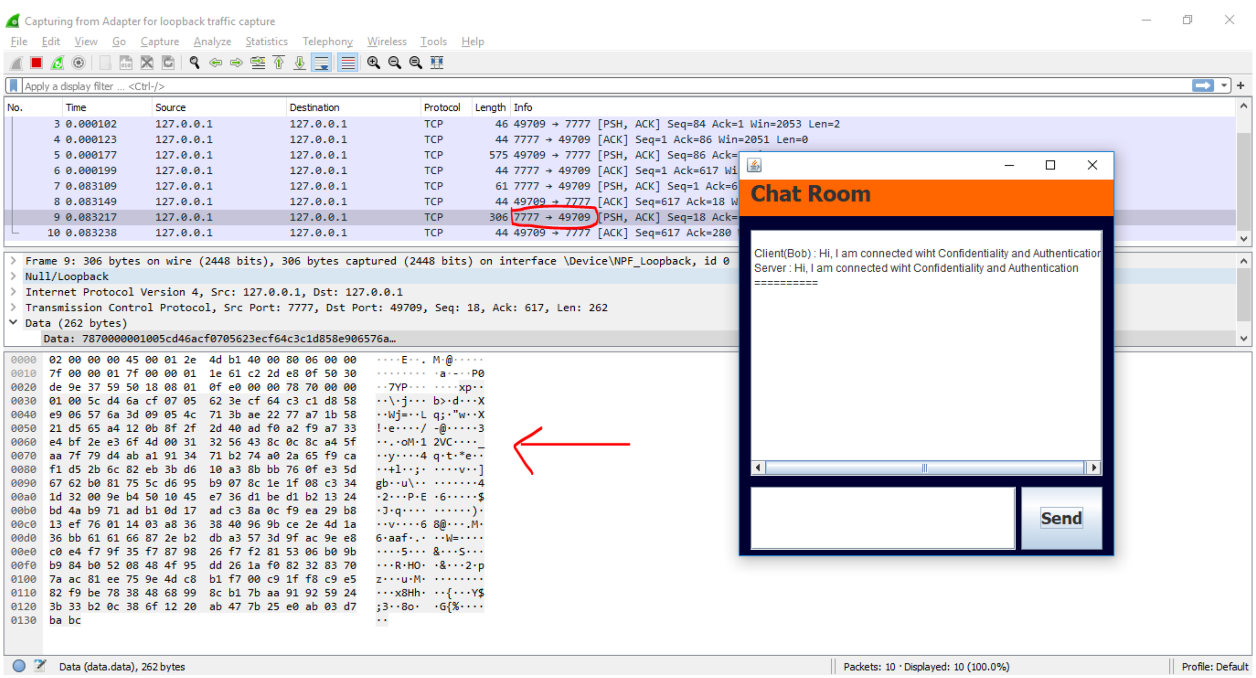


Figure 15. Wireshark screenshot to show the transmitted data with confidentiality and authentication 2

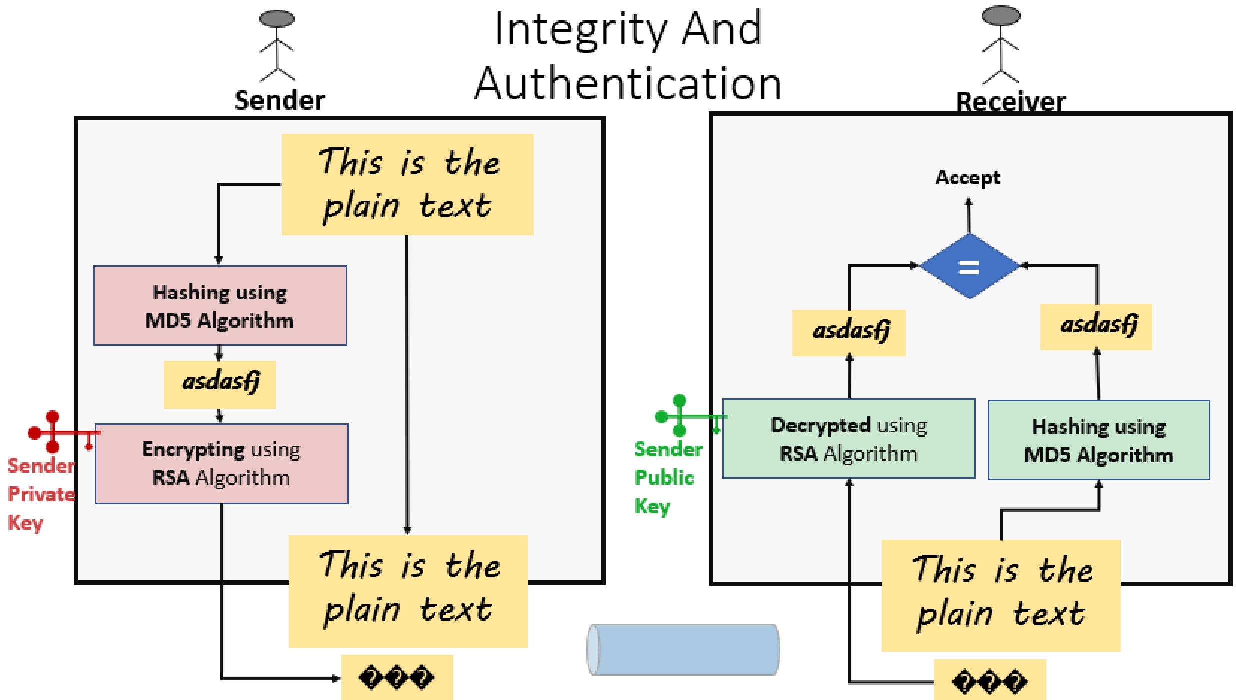
3.12 Login with Authentication and Integrity

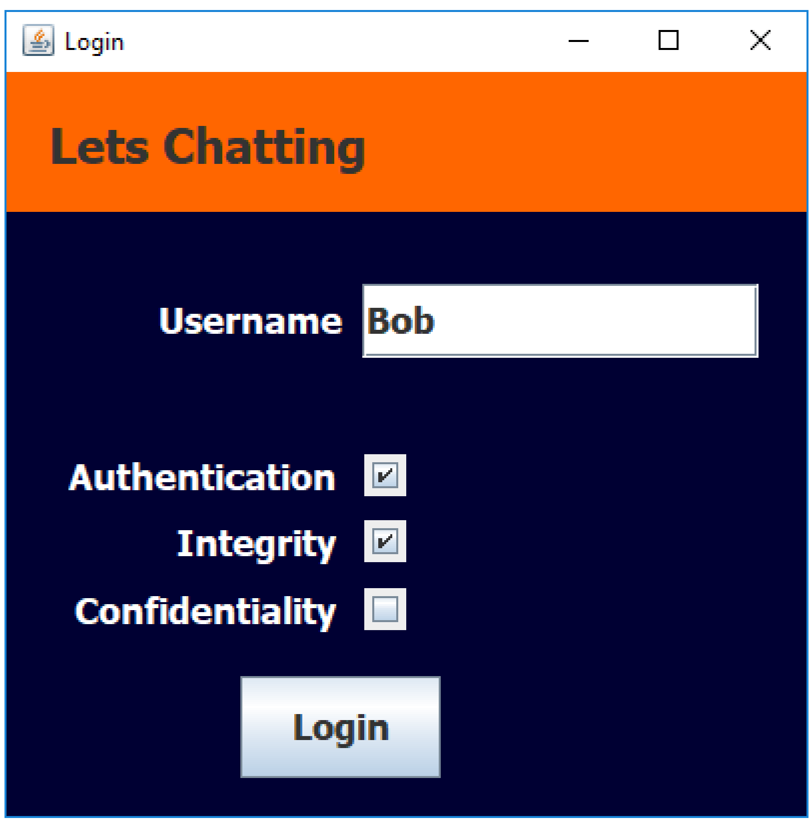
The following scenario shows the proposed solution to allow the client and the server to communication with only confidentiality. The following steps describe how the confidentiality is implemented:

Use asymmetric key to encrypt and decrypt message using with RSA algorithm.

1. **Client**: Write message
2. **Client**: Generate hash from the message using (MD5) hash function.
3. **Client**: Encrypt the hash using the client private key to create Digital Signature.
4. **Client**: Send data that contains both (the message and the digital signature) to the server.
5. **Server**: Receive data from client.
6. **Server**: Decrypt the digital signature using the client public key to get the encrypted hash and authenticate the client.
7. **Server**: Generate hash from the message using (MD5) hash function.
8. **Server**: Validate the integrity by comparing the decrypted hash and the generated hash.
9. **Server**: Encrypt the hash using the server private key to create Digital Signature.
10. **Server**: Send data that contains both (the message and the digital signature) to the client.
11. **Client**: Receive echo from client.
12. **Client**: Decrypt the digital signature using the server public key to get the encrypted hash and authenticate the server.
13. **Client**: Generate hash from the message.
14. **Client**: Generate hash from the message using (MD5) hash function.
15. **Client**: Validate the integrity by comparing the decrypted hash and the generated hash.
16. **Client**: Display the received message in the chat area.
17. done :)

The following figure describes how the authentication and integrity is implemented.


Integrity And Authentication
Sender
Receiver
This is the plain text
Hashing using MD5 Algorithm
asdasfj
Encrypting using RSA Algorithm
Sender Private Key
This is the plain text
���
Accept
=
asdasfj
asdasfj
Sender Public Key
Decrypted using RSA Algorithm
Hashing using MD5 Algorithm
This is the plain text
���


Login
Lets Chatting
Username
Bob
Authentication
Integrity
Confidentiality
Login

The following figures present the received and the sent message between client and server. Figure 16 shows the data sent form client to server, and Figure 17 shows the data transferred from server to client.

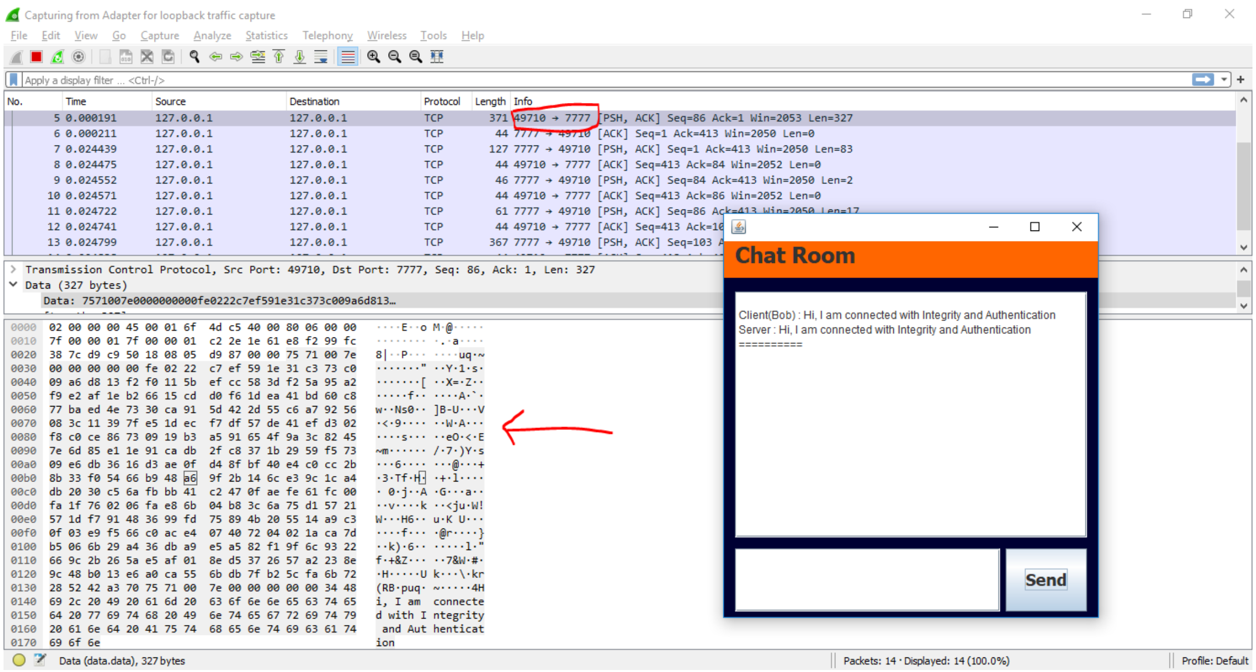


Figure 16. Wireshark screenshot to show the transmitted data with integrity and authentication 1

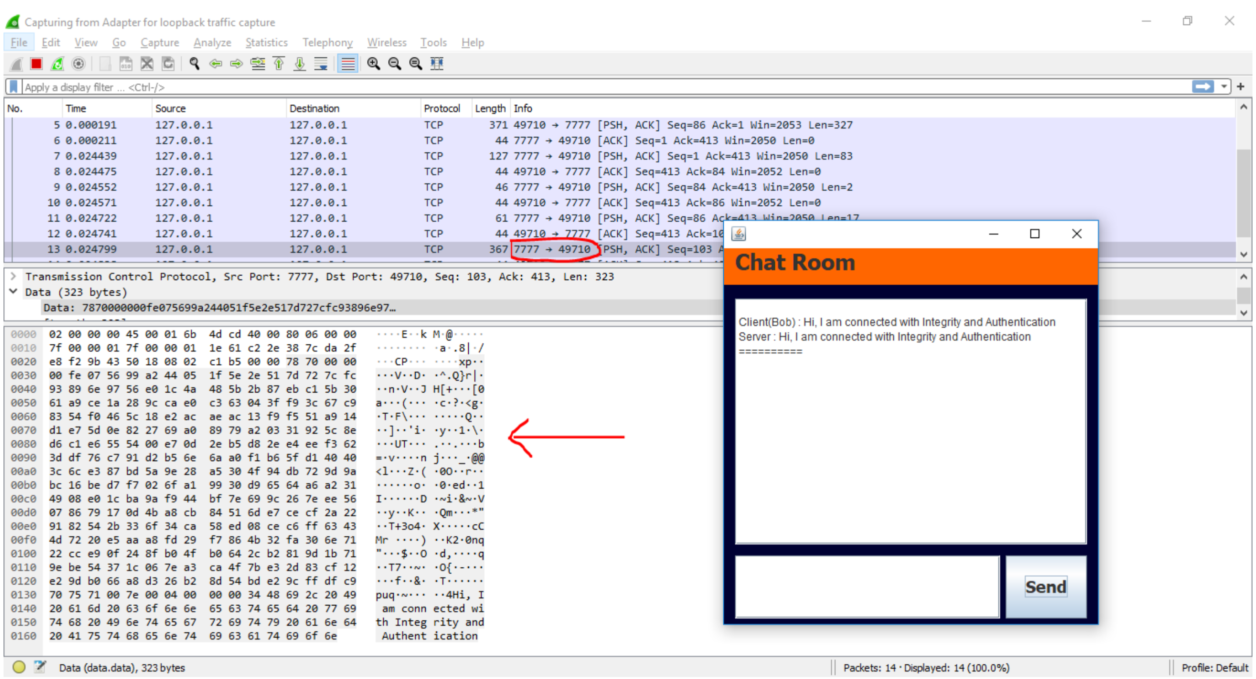


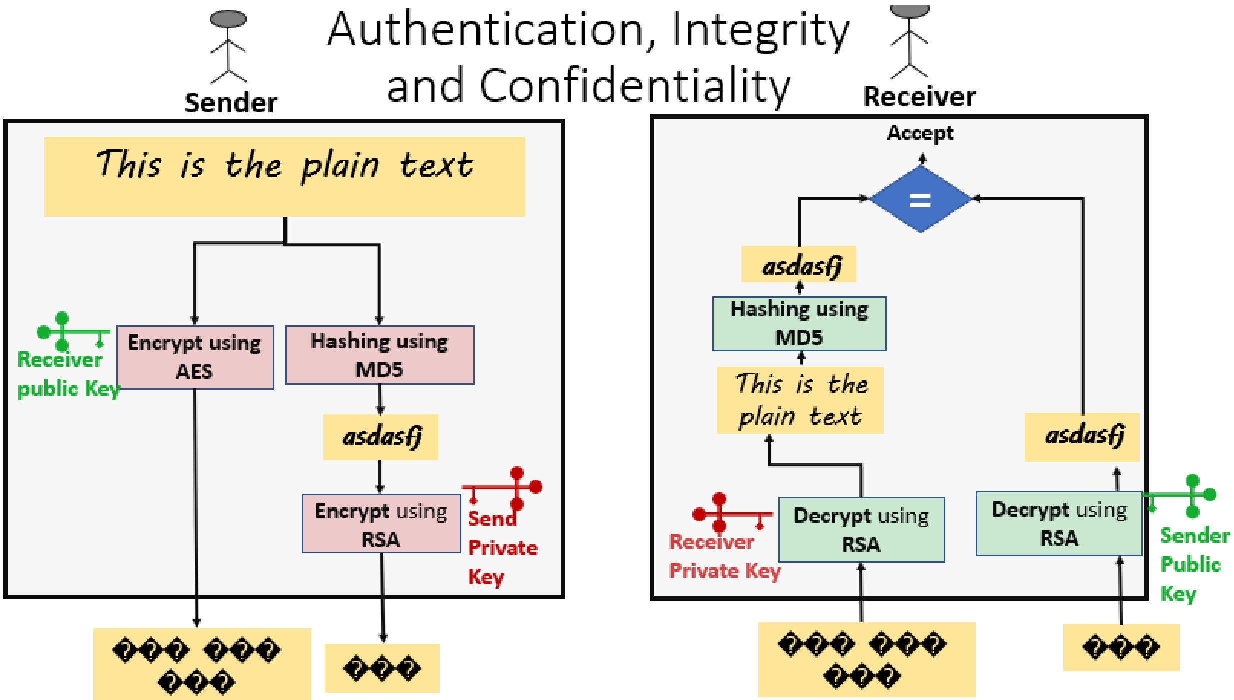
Figure 17. Wireshark screenshot to show the transmitted data with integrity and authentication 2

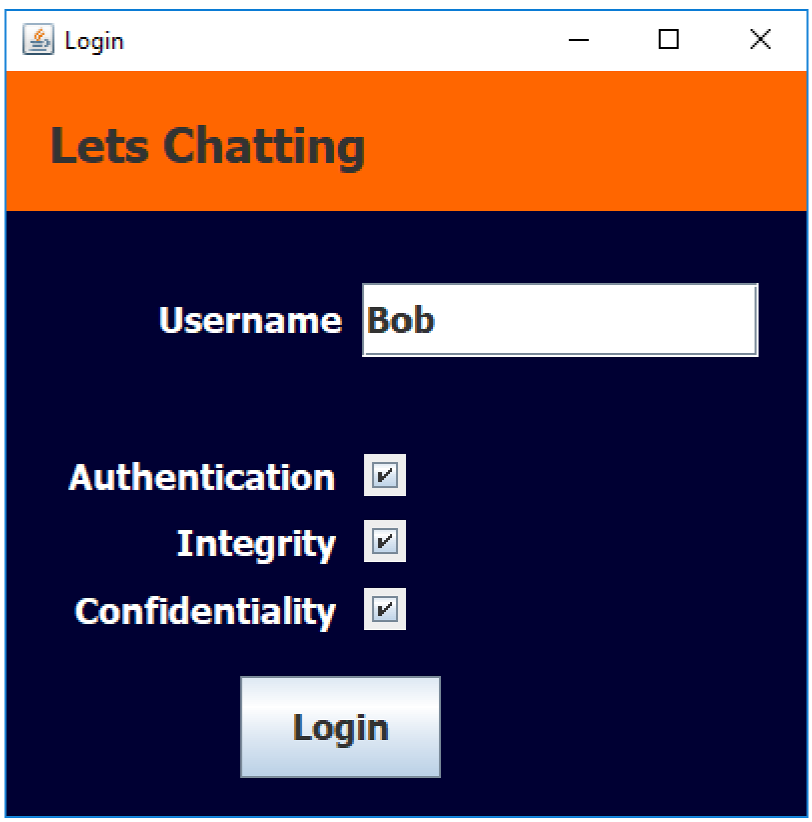
3.13 Login with Authentication, Confidentiality, and Integrity

The following scenario shows the proposed solution to allow the client and the server to communication with only confidentiality. The following steps describe how the confidentiality is implemented:

1. **Client**: Writing message
2. **Client**: Generate hash from the message.
3. **Client**: Encrypt the message using the server public key to ensure confidentiality.
4. **Client**: Encrypt the generated hash using the client private key to ensure authentication.
5. **Client**: Send the encrypted hash and the encrypted message.
6. **Server**: Receive the encrypted hash and the encrypted message.
7. **Server**: Decrypt the hash using the client public key.
8. **Server**: Decrypt the message using the Server private key.
9. **Server**: Generate the hash from the decrypted message.
10. **Server**: validate integrity by comparing the generated hash and the decrypted hash.
11. **Server**: Prepare to send echo.
12. **Server**: Encrypt the message using the client public key to ensure confidentiality.
13. **Server**: Encrypt the generated hash using the server private key to ensure authentication.
14. **Server**: Send the encrypted hash and the encrypted message to client.
15. **Client**: Receive the encrypted hash and the encrypted message.
16. **Client**: Decrypt the hash using the server public key.
17. **Client**: Decrypt the message using the client private key.
18. **Client**: Generate the hash from the decrypted message.
19. **Client**: validate integrity by comparing the generated hash and the decrypted hash.
20. **Client**: display the decrypted message in the chat area.

The following figure describes how the authentication, confidentiality and integrity is implemented.


Authentication, Integrity and Confidentiality
Sender
Receiver
This is the plain text
Receiver public Key
Encrypt using AES
Hashing using MD5
asdasfj
Encrypt using RSA
Send Private Key
��� ��� ���
���
Accept
=
asdasfj
Hashing using MD5
This is the plain text
asdasfj
Receiver Private Key
Decrypt using RSA
Decrypt using RSA
Sender Public Key
��� ��� ���
���


Login
Lets Chatting
Username
Bob
Authentication
Integrity
Confidentiality
Login

The following figures present the received and the sent message between client and server. Figure 18 shows the data sent form client to server, and Figure 19 shows the data transferred from server to client.

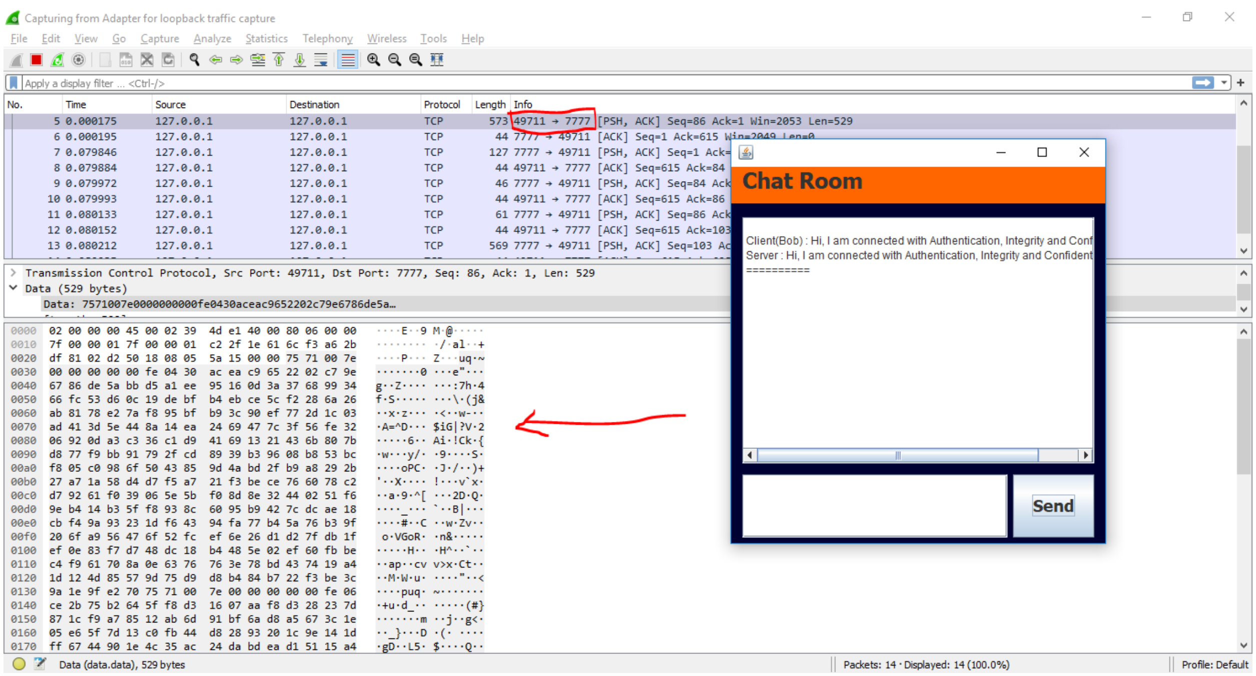


Figure 18. Wireshark screenshot to show the transmitted data with integrity, authentication and authentication 1

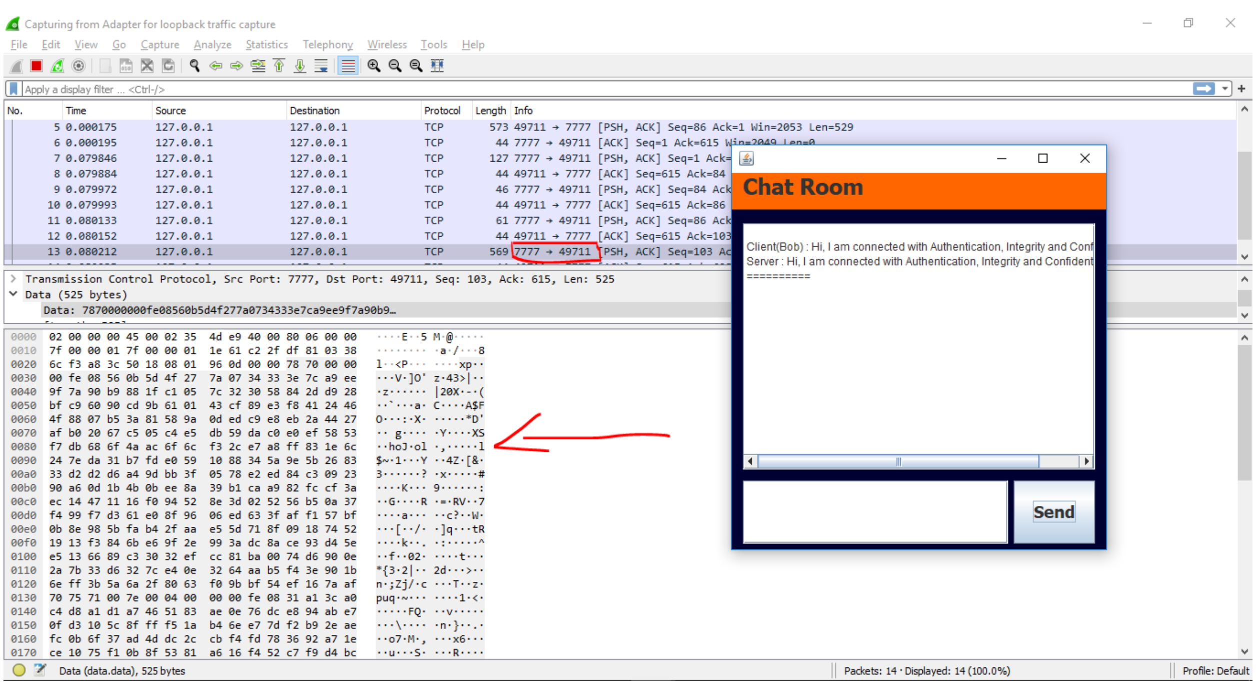
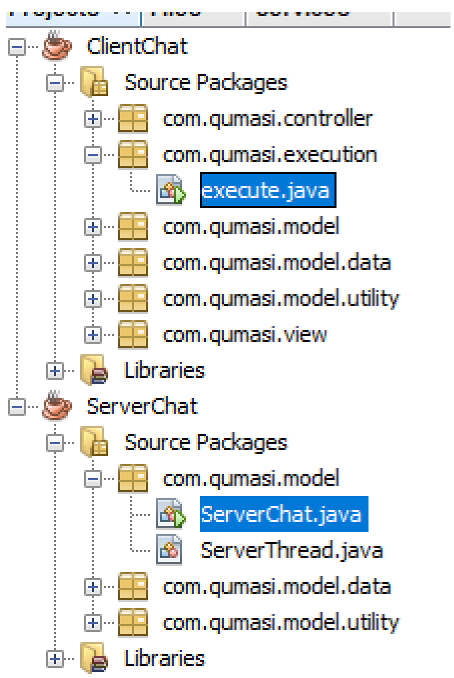


Figure 19. Wireshark screenshot to show the transmitted data with integrity, authentication and authentication 2

4. How to run the programs

1. Open the ClientChat Project using the NetBeans as shown in the following figure.
2. Open the ServerChat project using the NetBeans as shown in the following figure.
3. To run the server, run the file called ServerChat.java file in the following package “com.qumasi.model”.
4. To run the client, run the file called execute.java file in the following package “com.qumasi.execution”.
5. Login and enjoy chatting with server.



5. References

[1] F. Azzedin, "HW1- SEC 511 - Principles of Information Assurance and Security - Secured Instant Messenger," 2020.

[2] Oralce, Java Platform, Standard Edition Security Developer’s Guide, 2018.

[3] B. FOROUZAN, "Security Information," in *Data Communications AND Networking*, McGraw-Hill , 2013, pp. 1077-1177.