

Assignment 3-Design Documentation

SENG 360

Group members:

Claudia Li, V00826657

Courtney Maricle, V00

Adam Leth, V00894769

1. Design Techniques

1.1 Overview

We are using Java and the Java Cryptographic Architecture for implementing the program. A session key is generated every time the client tries to establish a new session with the server. The session key is generated by using the Advanced Encryption Standard (AES) algorithm with ECB mode and PKCS#5 padding mechanism (ie default setting for `KeyGenerator.getInstance("AES")`). The server then encrypts the session key by using the client's public key and transfers it to the client. The client then decrypts the session key by using its private key. The private/public key pairs for client and server are generated before the communication is initiated and stored in a local directory. The keypair is generated, encrypted and decrypted by RSA (Rivest–Shamir–Adleman) algorithm.

1.2 Confidentiality

To be able to implement the confidentiality aspect of the program it is important to note how exactly it is used. For the user to interact with the server in a confidential environment an encryption process is needed. When messages are exchanged over the network they are potentially vulnerable to attacks unless they are encrypted. We used a symmetric key for encrypting and decrypting messages since it is easier to implement. This needs to happen symmetrically in order for the security to have an impact on the vulnerability. Using the AES within the program we achieve a high level of confidentiality in the communication between client and server. The encryption of the communication initially happens on the client side and uses the *encrypt* method. When it receives a message from the server the program uses the *decrypt* method. The message is encrypted and decrypted by using the session key.

1.3 Integrity

The integrity aspect of the program is implemented using the Message Authentication Code (MAC) which provides a high level of data integrity. As it is required to protect the integrity of the message the two parties are required to have a commonly known key that is used in the MAC algorithm to establish a MAC tag. Both the message and the MAC tag is sent to the server which once again uses the shared key to run the message through the MAC algorithm to generate another MAC tag. The newly generated MAC tag is then compared to the MAC tag that was attached to the message from the sender; if the two MACs are identical the integrity of the message is secured.

1.4 Authentication

The authentication aspect of our program is the first security measure that the user comes across. To be able to authenticate that the user is actually authorized to operate the program the entry of a username and password is a basic security measure. A username/password file is pre created and stored in local directory. The the password is protected by storing in

hashed value using SHA-256 one way hashing. To log in to the system the program hash the user input and compare it to the username/password file.

the pre created username and password are listed:

username	password
-----------------	-----------------

guest	guest
-------	-------

admin	admin
-------	-------

you can generate more account by using `accont_generator` program

2. Functions

In the communication between our client and server we have implemented several different functions. These function as messenger to and from the server.

2.1 Server-side:

Our `Server` function is used to start the server and listen on port 8080. It prints out a message if it commences successfully. Furthermore we need to check if the input of the message from the client is readable, which is done in the function `checkInput` that uses a `BufferedReader` to ensure an efficient reading of the lines. In the same way we used the `BufferedReader` to check the input, we are now using the `BufferedWriter` in the `sendOutput` function to send the message back from the server to the client.

For the purpose of the security properties the `optionsSelected` function provides a set of scenarios that is based on if-else sentences which allows for determination of the security options selected for the message. This means that each of the three security aspects is set to either 'true' or 'false' and ensures that the desired security is imposed. This function is closely connected to the `getSecurity` function which figures out what options the user wants by asking the user yes/no questions for each security aspect. As a result you will now be informed of which choices you have made and therefore how you are protected. When the main method is run the program checks if the server and the client has the same security settings and ensures that communication between the two can't be executed until this happens.

2.2 Client-side:

To engage in a communication with the server, the client uses almost the same functions as the server. The client also initiates communication with the server by using the `Client` function to transmit on the port that the server is listening to; 8080. Given that the client is required to be able to communicate with the server we need to implement `send-` and `getMessage` that are used to transmit a message from the client to the server and receive the response message once the server has checked it. The `getSecurity` function on the client side is exactly the same as on the server side and asks the user if it wants each of the three security aspects to be valid during the communication.

As earlier mentioned the confidentiality aspect of the program is using AES which is used to encrypt the keys that are exchanged between the client and the server. To be able to decrypt the message received from the server the AES standard is used again.

3. Limitations

As part of the implementation of this program there are some unfulfilled components that could have been implemented but is missing as a result of shortage of time. The inclusion of nonces and timestamps has been left out of the communication but would provide a radically stronger program which could prevent replay attacks. This would be to secure freshness of the message which would complicate an attacker's efforts in getting unauthorized access to the messages. Another limitation is the project is implemented on one end and it compile and run perfectly but it throws exceptions on another machine, we are not sure why this happened(it could be version of Java is different).

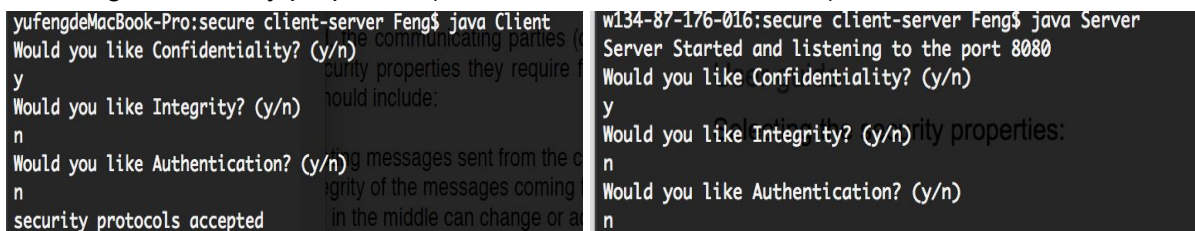
4. Instructions

How to compile and run the program

1. Navigate to the project directory
2. Compile the project by running "javac Server.java" and "javac Client.java"
3. open 2 terminals
4. run "java Server" on 1 terminal
5. run "java Client" on the other

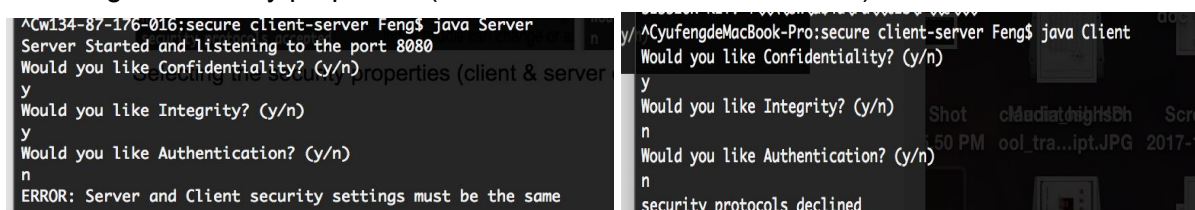
User guide

Selecting the security properties (client & server same selection):



The image shows two terminal windows side-by-side. The left window is the client terminal, and the right window is the server terminal. Both show the same sequence of prompts and user input: 'Would you like Confidentiality? (y/n)' with input 'y', 'Would you like Integrity? (y/n)' with input 'n', and 'Would you like Authentication? (y/n)' with input 'n'. The client terminal ends with 'security protocols accepted' and the server terminal ends with 'Server Started and listening to the port 8080'.

Selecting the security properties (client & server different selection):



The image shows two terminal windows side-by-side. The left window is the server terminal, and the right window is the client terminal. Both show the same sequence of prompts and user input: 'Would you like Confidentiality? (y/n)' with input 'y', 'Would you like Integrity? (y/n)' with input 'n', and 'Would you like Authentication? (y/n)' with input 'n'. The server terminal ends with an error message: 'ERROR: Server and Client security settings must be the same'. The client terminal ends with 'security protocols declined'.

Selecting all 3 security options and ask for authentication after selection. If the username and password matches the record the user will be signed in and can start exchanging messages.

```
Courtneys-MacBook-Pro:seng_360 courtneymaricle$ java Client
Would you like Confidentiality? (y/n)
y
Would you like Integrity? (y/n)
y
Would you like Authentication? (y/n)
y
security protocols accepted
Enter a username:
guest
Enter a password:
guest
signed in as guest

this is a test message from the client
█
```

```
Courtneys-MacBook-Pro:seng_360 courtneymaricle$ java Client
Would you like Confidentiality? (y/n)
y
Would you like Integrity? (y/n)
y
Would you like Authentication? (y/n)
y
security protocols accepted
Enter a username:
guest
Enter a password:
guest
signed in as guest

this is a test message from the client
█
```