# Building A Secure Communication Channel

There is a small update on step 6.a (apologize); But please do not worry: this step does not affect follow-up steps and won’t be graded (all updates are colored in purple).

Update: The GitHub repo has been cleaned up one more time. Please pull again to make sure everything is synchronized.

​​Update: Regarding the reference client, we decided to not provide it, in case you guys are too smart and spend all the time on reversing the bytecode.

~~Update: the public repo is back online!~~

## Description

In this assignment, you are required to build a secure communication channel between a server and a client, where symmetric encryption is used to encrypt the messages and RSA is used to ensure the integrity of the messages. To facilitate this assignment, we have created the code of the server; we have also prepared all the crypto APIs in RSAUtils.java. **You are required to complete the code of the client** (a wrapper program for the client will be provided with necessary preparations, you will just need to fill in the code).

**Note: You may need to understand the code of Server.java and RSAUtils.java when creating the client. In particular, you may need to learn how to use the related APIs based on how they are used in the server.**

## The communication will take the following steps:

1. The server will open a socket listening to “localhost” port number 1337 and wait for connections
2. The client will need to generate a pair of RSA public+private keys. RSAUtils.java contains an API (getKeyPair) for that. You can reuse that. Regarding how to use that API: please refer to line 310 of RSAUtils.java
3. The client will open a socket trying to connect to the server. You have to figure it out yourself
4. Once connected to the server, the client will send the first message to the server
   1. The message will consist of two parts: client RSA public key + the signature of client RSA public key
      1. Format of the message: [signature of the client RSA public key + “\n” +**string** of the client RSA public key + “\n”]. Here “+” means concatenation and “\n” means the new line character.
      2. To get the signature of some data, you can reuse the API (sign) from RSAUtils.java. You can refer to line 326 of RSAUtils.java regarding how to use this API.
      3. When you use sign to get the signature of some data, the data needs to be a signature. That means if you want to apply sign to the client RSA public key, you will need to convert the client RSA public key to a string. Line 311 and 321 of RSAUtils.java show how to convert RSA private/public keys into strings. This is needed since you will need to attach the string of the public key in the message.
5. The server will send a message back to the client, which the client should receive
   1. Format of the message: [**string** of an encrypted AES key + “\n” + a random string + “ ” + Client IP + “\n” + signature of the “encrypted AES key + random string” ]. Here “+” means concatenation and “\n” means the new line character; Here “ ” means space
   2. The encrypted AES key is an AES key picked by the server and encrypted by the server using the client public key
   3. The random string is a string picked by the server; the size is 10 bytes
   4. The signature of the “encrypted AES key + random string” is obtained also using the sign interface and the server private key
6. Once the client received the above message, it will need to (i) verify the signature of the encrypted AES key (ii) decrypt the encrypted AES key to get the original AES key (iii) encrypt the random string with the original AES key, and (iv) send a message which contains the encrypted random string as well as its signature back to the server
   1. Verifying the signature of the “encrypted AES key + random string” can reuse the verify interface from RSAUtils.java with the public key of the server as an argument. Line 329 in RSAUtils.java shows an example of doing so
   2. Decrypting the AES key will need to use the client private key. The decryptByPrivateKey interface in RSAUtils.java can be used to do so. Line 322 in RSAUtils.java shows an example of doing so
   3. Encrypting the random string with the original AES key can be done with the aesEncrypt interface from RSAUtils.java. Line 295 - Line 298 in RSAUtils.java shows an example of doing so
   4. The message will follow the format of [signature of the encrypted random string + "\n" + the encrypted random string + "\n"]
      1. The signature will be done with the private key of the client. Please refer to the above regarding how to do it
7. Once receiving the message from the client, the server will send an OK message back
   1. Format of the message: [ “OK” + “\n” signature of “OK.” + “\n” + string of the client IP address ]
   2. The signature will be based on the private key of the server
8. Once receiving the message from the server, the client will verify the signature; if the signature is OK, close the connection.

## Public Code

<https://github.com/junxzm1990/CS-4440-2022-Spring/tree/main/assignment-2>

From this repo, you can have code for Server.java and RSAUtils.java. You will also have a wrapper for Client.java. You just have to develop the code in Client.java, following the specifications above.

To check whether your client is working correctly, you can check the output of the server. Reference output of the server has been posted in the README file in the above repo.

## Submission

Please submit your Client.java to <https://www.gradescope.com/courses/347861> by March 20th, 11:59 PM.