**Final Project**

**System Design and Architecture**

For the final project I designed a two-way communication system utilizing RSA Encryption. There are a total of 3 unique classes, AES, ClientA, and Client B. Both ClientA and ClientB can send a message and receive a message, the code is nearly identical with exceptions to some naming conventions and path directories, but you only need to inspect either ClientA or ClientB to understand how the system works. The program scenarios plays out as such, you may run both programs at the same time or one at a time, but its better when they are both running. Either client will be prompted to select between the Sender or the Receiver. During that process of selection key generation is made regardless, if the keys already exist which they do load the keys, else generate new keys and save them to the files. Note: Only the Client classes have main methods to them, and they are the only runnable source code.

**Sender**: Sender will always have to go first in order to initialize a conversation, the sender will load the public key from the other client assuming they have already have it. The process that the sender will go through is to create an AES key and encrypt the message they want to send to the other client, creating the ciphertext. The RSA key that was loaded from the other client’s folder is used to encrypt the key, and finally to authenticate the validity of the message to detect tampering is to use a HMAC on the original message which is hashed with the public key (very important). Three crucial pieces of data are then sent to the mock server/ file exchange into the transferred directory, written into the data.txt file in order of the encrypted message, encrypted key, finally the Mac. Now it is the receivers turn to continue the process.

**Receiver**: The data is now available to retrieve, so we access the file that contains the encrypted data and load it into our program. The client will utilize his own private key which he loads in his/her own directory to decrypt the messages because their public key was used to encrypt them. In this order we decrypt the encrypted key which contains the AES string which will allow the client to decrypt the ciphertext. Finally, to validate that the message was not altered during the process with the client’s original public key, they hash the encrypted ciphertext with the public key and compare to the HMAC sent from the data file, if there is a match then the authentication is true, if they differ then the message was altered before reaching the client.

**Technical Details**:

RSA Key length used: 512bits, Minimum length

AES class: From howtodoinjava.com

HMAC: I used the “HmacSHA256” Mac Instance.

Overall Process:

Encryption: (RSAe(AESk), AESe(m1), MACrsa\_pub(m1)) 3 items into 1 file

Decryption: (RSAD(AESk), AESD(m1), MACrsa\_pub (m1))

**Example RunGraphical user interface, text, application

Description automatically generated**

Graphical user interface, text, application

Description automatically generated

**ClientB** is Sender, goes first and sends:

Cipher Text: 9IJhZ3Jn5dBlLzb2QN/7Jg==

Cipher Key: [Õ|ˆ\_¾ˆ²Ëõö¦²@å÷W?2I7?ÙšÞBdÕ÷bYÖ²ñ¶eŠkW‘möW²»:D?3w‡

MAC: >ÉÓê›Øº¥VÓ<QÝx Âq´k5‹Ü#þ:ñ}‰ºÂO

**ClientA** receives:

Cipher Text,Cipher Key, and Mac;

Decrypted Key: ClientB\_AES

Decrypted Message: 123abc

New MAC: >ÉÓê›Øº¥VÓ<QÝx Âq´k5‹Ü#þ:ñ}‰ºÂO

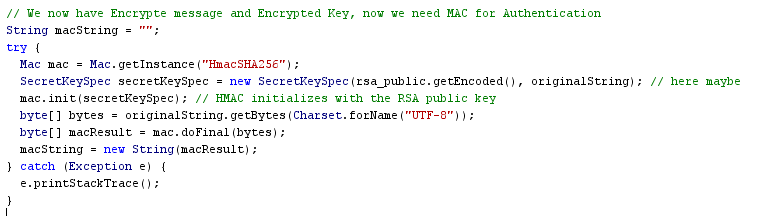
New Mac == Mac, Authentic

**Important Code Snippets**

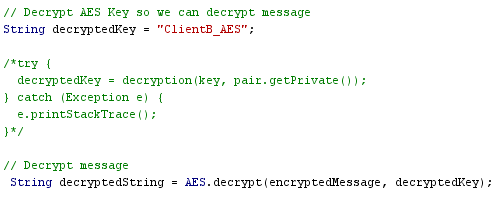
**Graphical user interface, text, application, email

Description automatically generated**

Detect if the key is generated or not and prompt the user if they want to be the sender/receiver.



Since the code was short, I decided not to put this into its own method, but this is the algorithm used for the HMAC hashing utilizing the client’s public key.



Some difficulty when going through the decryption process (padding Errors), so had the hard value placed to continue with the assignment.

**Security analysis**

Given this structure: Encryption: Encryption: (RSAe(AESk), AESe(m1), MACrsa\_pub(m1)), Decryption: (RSAD(AESk), AESD(m1), MACrsa\_pub (m1)) This is a fairly safe communication system between a sender and a receiver. One major issue could be the key length used is only 512 instead of larger or more recommended sizes. If the attacker was able to break into the server and steal either public key the attacker can mimic either client tricking the others to communicate as if they were the intended client, since the same public keys are used until one of the clients decides to renew them. So, a man in the middle attack could possibly be made if they were able to break into the server.

**Solutions**

An easy solution could be made by Timestamping the communication and using new keys for every iteration of the communication (such as TLS), so if the attacker receives the public key, it will be useless. Just in case an attacker does receive the public key that is up to date, using a nonce to help alert and notify the participants in the communication become aware that the communication is compromised and to renew their keys immediately as added protection. Finally increasing the RSA key size would be a great option against brute force attacks, let’s say a very top-secret confidential message or file is left to be kept secret for 20 years (disregarding ethics whether it should or shouldn’t be) between the CIA and the US military, if foreign attackers are willing enough using modern hardware, they could breach those secret documents within months to a few years using RSA 512, which is why 512 bits is minimal but is not the recommended size.