Ricardo Baeza

Cryptography Final Project Report

System Design/Algorithms Used/Key Lengths Used

The design of my final project begins with the sender generating a public and private key using a key pair generator that creates the key size of length 1024 bits. This is done by calling the init() method in the receiver package, file: class: RSA followed by an export function which takes the generated public RSA key and converts it to a string and saves it to a txt file.

```
rsa.init();
rsa.exportPublicRSAKey(rsaPublicKeyPath);
```

After having the RSA public key written into a txt file, the sender will load a message from a txt file within the sender package, generate a secret key using the AES key generation algorithm (of a 128 bit key length) and encrypt the message the sender would like to send to the receiver (from txt) using the AES key generated (this encrypted message will be stored in a sendersTransmission.txt file). Then, the AES key that was generated and its IV will be stored in String format so that it may be used later on the receiver's end. (The key and IV are converted to String format using an encode function).

```
aes.exportKeys();
IV = aes.getIV();
```

```
public void exportKeys() { //exports secretKey and IV to strings
    secretKeyString = encode(secretKey.getEncoded());
    System.out.println("Secret Key: " + secretKeyString);
    IVstring = encode(IV);
    System.out.println("IV: " + IVstring);
}

public String getIV() { //returns IV as a string so that we can s
    return IVstring;
}
```

The sender then loads in the receiver's RSA public key, converts the key back to its PublicKey data type by decoding the key String and using a KeySpec (initFromString()). Having the RSA public key and the AES key, the AES key is then encrypted using the receivers RSA public key.

```
rsa.loadInPublicKey(rsaPublicKeyPath);
rsa.initFromString();
aesKey = aes.getAESKey();
encryptedAESKey = rsa.encrypt(aesKey);
```

```
public void initFromString() throws NoSuchAlgorithmException { //convert public RSA Stri
    try {
    X509EncodedKeySpec keySpecPublic = new X509EncodedKeySpec(decode(publicKeyString));
    KeyFactory keyFactory = KeyFactory.getInstance("RSA");
    publicKey = keyFactory.generatePublic(keySpecPublic);
    }catch(Exception ignored) {}
}
```

The encrypted AES key with the receiver's RSA public key, the AES keys IV, alongside the previously mentioned encrypted AES message are written to the sendersTransmission.txt file.

As one of the final steps for the receiver, the receiver generates a Message Authentication Code so that the receiver would be able to authenticate and verify that data sent by the sender is correct and authentic. The sender generated the MAC by generating a SecretKey of key length 256 bits which is then converted to string format using an encode function. After, the actual MAC is computed using the MAC generation algorithm HMACSHA512 that takes the SecretKey generated and the original message (from our sendersMessage.txt file) as inputs (this MAC is

then converted into string format. The MAC (code) that was generated is then encrypted with the receiver's RSA public key. Finally, the MAC SecretKey and MAC encrypted with the receiver's RSA public key are then written into the sendersTranmission.txt file for the receiver to use the information written down in the file.

```
MAC = mac.computeMAC(message);
encryptedMAC = rsa.encrypt(MAC); //encrypt MAC with RSA Pu
sender.writeToLine(encryptedMAC, transmittedDataPath, 4);
sender.writeToLine(MACKey, transmittedDataPath, 5);
```

Going back to the receiver, the receiver loads in data such as AES encrypted message sent by the sender, the AES Secret Key that was encrypted with the receivers RSA public key, the AES Key's IV, the senders encrypted MAC (code) with the receivers RSA public key and the MAC Key (the data loaded is assigned to their respective variables in the program). After the sender's data is loaded, the receiver loads in their previously stored RSA public and private keys and decrypts the encrypted AES Key. Once the AES Key is decrypted, the AES Key (in String form) and the AES Key's IV (in String form) are re-converted into the SecretKey format by using SecretKeySpec and a decode function. Now having the AES Key, the sender's message that was encrypted with the sender's AES Key, is now able to be decrypted. Even though the receiver is able to read the message, the receiver is still unsure whether the message sent by the sender is actually from the sender and not by someone else. In this case, the receiver takes in the receivers MAC decrypts it with its RSA private key, takes the MAC key (from sender) and the message from the sender and generates the MAC (code) with the same HMACSHA512 generation algorithm and compares it to the senders MAC. If they are the same, a message will appear that the sender's message is authentic and can be trusted and if they are not the same a message will appear stating that the message was not authenticated and that it should not be trusted.

```
rsa.loadPublicteKey();
rsa.loadPrivateKey();
rsa.initFromString();

aesKey = rsa.decrypt(aesEncryptedKeyWithRSA);
aes.initFromStrings(aesKey, IV);
sendersOGMessage = aes.decrypt(aesEncryptedMessage);

sendersMAC = rsa.decrypt(sendersEncryptedMACWithRSA);
System.out.println("Senders MAC: " + sendersMAC);

mac.initFromStrings(MacKey); //macKey is the symmetri

recieversMac = mac.computeMAC(sendersOGMessage);
```

```
if (sendersMAC.equals(recieversMac)) {
    authenticated = true;
    System.out.println("The message sent by the sender was successfuly authenticated");

}else {
    authenticated = false;
    System.out.println("This message cannot be authenticated. The sender should not be
```

Now, in terms of the receiver sending a message to the receiver (where the receiver sends a message to the sender and sender has to generate a public and private RSA keys and do the operations the receiver performs (as mentioned above)) are all performed the exact same way, with the same 1024 bit RSA Keys, 128 bit AES Key, 256 bit HMAC Key, HMACSHA512 MAC generation algorithms, procedures and executions.

Additional System Design/How To Run Programs/Program Runthrough

Sender to Receiver:

Receivers End

Before running the program start off with the receivers program, open the Reciever.java file. There will be three sections Part 1 and Part 2 (which will be used for sender to receiver communication) and Part 3 (which will be used for receiver to sender communication). When Running this for the first time, make sure that all the contents of Part 1 are uncommented and the contents of Part 2 and Part 3 are commented (Basically lines 34 and on are commented).

```
//      //Part 1
        rsa.init();
        rsa.exportPublicRSAKey(rsaPublicKeyPath);

////        Part 2

//      aesEncryptedMessage = loadEncrAESKMess.getEncrAESMess(encryptedAESMessage
//      System.out.println("AES Encrypted Message: " + aesEncryptedMessage);
//
//      aesEncryptedKeyWithRSA = loadSenders.getLine(2);
//      aesEncryptedKeyWithRSA = loadSenders.getLine(1);
//      System.out.println("RSA Encrypted AES Key: " + aesEncryptedKeyWithRSA);
```

Saving to files

Another thing that must be taken into account is setting the path (set a path where you would like to save the RSA Public key (make sure the end directory is a txt file).

E.g.: (This can be found on line 10 of the Reciever.java file)

```
String rsaPublicKeyPath = "/Users/ricardobaeza/Documents/publicRSAKey.txt"; //**
```

One of the last things before running Reciever.java is to set the path to where the privateRSAKey.txt will be saved so that it can be used in Part 2. This path can be found in the savePrivateKet() function on (line 72 of the RSA.java file.

E.g.:

```
File output = new File("/Users/ricardobaeza/Documents/privateRSAKey.txt"); //**
```

Now, you can finally run the Reciever.java class.

You will see the following in the console and in the respective .txt files stored to the path chosen.

```
RECIEVER
Public key: MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCBcDEh99VDw5hjF8zYzwicSG+TDNkzvb/UnF
Private Key Written to File: MIICdgIBADANBgkqhkiG9w0BAQEFAASCAmAwggJcAgEAAoGBAIFwMSH31
Private key: MIICdgIBADANBgkqhkiG9w0BAQEFAASCAmAwggJcAgEAAoGBAIFwMSH31UPDmGMXzNjPCJxIb
Public Key From Init: Sun RSA public key, 1024 bits
  params: null
  modulus: 90894630032753436684280201694358713232461897309480022913902394497710337734
  public exponent: 65537
```

publicRSAKey.txt

```
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCBcDEh99VDw5hjF8zYzwicSG+TDNkzvb/
UnFXc8hEY8zTPbFMc+P9Zel3DRs0MOWJC6BpQq5q3nrqPB20lFKAzd+OMXXhLBX50cQINDVMcGDPgtRvO7tQwV3WfG
Zb5IFLV4xmTOr3YP3Vb1uZxvUcB8KgaZAE9OWoW+hZq+sSA/wIDAQAB
```

privateRSAKey.txt

```
MIICdgIBADANBgkqhkiG9w0BAQEFAASCAmAwggJcAgEAAoGBAIFwMSH31UPDmGMXzNjPCJxIb5MM2TO9v9ScVdzyER
jzNM9sUxz4/1l6XcNGzQw5YkLoGlCrmreeuo8HbSUUoDN344xdeEsFfnRxAg0NUxwYM+C1G87u1DBXdZ8ZlvkgUtXj
GZM6vdg/dVyW5nG9RwHwgBpkAT05ahb6Fmr6xID/AgMBAAECgYAEzCC2DpZz7YLvYUlxW7MI/9JK0/
YU6dp7SdQQ3VPu5cy2R2CJOV1VdqiU/
UHKSzIuIqYfCU0xswtdE7xi3HHXiP8A+um5CqWAPruGY5d+B63pG13FzSGiw0IdWtLJk/
wCSTOEekL5Y3XRD6goOV5Lw8Kf7/
Yqt4OUoAwRT7U1YOJBAL6Skkl4cTKdqKyZptapE5ARX4TitDDDq0/5Mesmn5HRfF3Mw8tiCBeMaA0z809zOqws8o6m
WclxC8277/szvA8CQOCt4IZdfWEBFtHOVskqWH+2npMuca7Ud8L2UeNwZmaLSBe7tCEjUpNMWrjSvO/
hzpmgV8Z+ZQpBdo1fgGJcJrwRAkBc6Lp3JUgSxCTqgRKMzuthroaiE6CDsZ4ATF5g5XigL8BxPpYqPsXxLRvBbBVwI
550OImcsE9P3VRC3iBWmNdTAkEAoTrLHTG/
m2Tvk2fCDXQWqN4OGEjuFrIepI12Wql5eLmW2ETFXZ2aXHjbXP45aadLqPT7aASFKAKoQOApVM3OIQJAJUGvrgnuzC
Ha6ilEBwi9cquT4fAaqvL0GnuwV02j+VnKmJ75RY+VbzJBbGFLNDEHoJ/3+OBWodYreKvGSC2QLA==
```

Senders End

In this Sender.java program there will also be a Part 1 and Part 2, make sure Part 2 is commented here.

On line 40 of the Sender program will load in the message that it would like to send to the user which is the sendersMessage.txt which is already hard coded, however the messages path must be changed to your respective path in which the txt file is loaded (if it makes it easier you can create your own sendersMessage.txt and change the path of 'messagesPath' on line 16.

```
String messagesPath = "/Users/ricardobaeza/Documents/ComputerScience/EclipseProjects/Crypto Final Sender/src/sender/sendersMessage.txt";
```

Once the program has encrypted the message (sendersMessage.txt) on line 44, the sender will write the encrypted message to the transmittedDataPath which can be found on line 19. Make sure to change the path to somewhere you would like to store this .txt file (keep the sendersTransmission.txt directory).

```
String transmittedDataPath = "/Users/ricardobaeza/Documents/sendersTransmission.txt";
```

Now, on line 51, there will be a loadInPublicKeyMethod() that takes in the rsaPublicKeyPath that was previously determined in the receiver's end. Make sure to change it to what was previously determined based on your machine's directory.

```
String rsaPublicKeyPath = "/Users/ricardobaeza/Documents/publicRSAKey.txt";
```

Also make sure that the path to the encryptedMessage is specific to your machine's directories. The rest of writing to the transmittedDataPath will be taking care of writing things such as encryptedMessage, encryptedAESKey, MAC Key etc.

So now, you can finally run the Sender.java file. As a result you will see the following on your console and respective files.

```
Message Loaded: Hi reciever!
Secret Key: Vllq/w9dt8qLCjKhSyfGZg==
IV: 7AKZ5dGKI/BEj4us
RSA Public Key: MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCBcDEh99VDw5hjF8zYzwicSG+TDNkzvb,
Encrypted AES Key: W2/0ztZDQlaoy0LOlk0GNKJbElemEwuRkTQOfj/ONew9TxCGkYNR11aL/dVrj0WVEte(
Encrypted Message: T2a4yvcMIXspxTYpslKP4kv7y5RPtN1ywgjNVA==
MAC Key: JqbJtDqFPNAksQakWSSHMLUGrwBMggIjuUKWTkpo9rg=
MAC : a2Y//dEpEE1BgsfkUapXoBshd/jFkbSPZnxtk/Ww4YmaxT5XKSvgfWqZdUEaW2jsEUZnr6jQh/jBNY6l
```

sendersTransmission.txt

```
T2a4yvcMIXspxTYpslKP4kv7y5RPtN1ywgjNVA==T2a4yvcMIXspxTYpslKP4kv7y5RPtN1ywgjNVA==
W2/0ztZDQlaoy0LOlk0GNKJbElemEwuRkTOOfj/ONew9TxCGkYNR11aL/
dVrj0WVEteOcWQ8GNvi17YMPqxdSwhqCnaqcFuXMUX/
Y9sleUVkqx4Mf8XE+RqtwJQ4xATkfYOubf1hh1ac1Hb48SJ7vJx0tiCBx/fw6mSFWfeoPxk=

7AKZ5dGKI/BEj4us


eNAPakMcoeFuzpPimiREFZd3E1rn404uGE5Re9z/Y9XKzDNqZWPvW9XqfWfwX3zdkBIwyav5V6uYBw2lshk7N6mJm/
5dS5B23l5TqjM0mdXrCBA1yCsVbZNaqIkbm+zDJPWRg65PCJSDK0l+PfqQuAHgT+kt97JvS4ixhd7qKyo=



JqbJtDqFPNAksQakWSSHMLUGrwBMqqIjuUKWTkpo9rq=
```

Back to the Receiver

At this time, make sure to comment out Part 1 lines 31-32 and uncomment which is line 36 to line 81.

```
30 /       //Part 1
31 /       rsa.init();
32 /       rsa.exportPublicRSAKey(rsaPublicKeyPath);
33
34 ///         Part 2
35
36         aesEncryptedMessage = loadEncrAESKMess.getEncrAESMess(e
37         System.out.println("AES Encrypted Message: " + aesEncry
38
39         aesEncryptedKeyWithRSA = loadSenders.getLine(2);
```

On line 39 there will be aesEncryptedKeyWithRSA and calls the getLine of loadSenders. Here, make sure to go to the LoadSendersTrans.java file, go to the BufferedReader br on line 14 and make sure to change the path to where you previously stored the sendersTransmission.txt file.

```
BufferedReader br = new BufferedReader(new FileReader("/Users/ricardobaeza/Documents/sendersTransmission.txt"));
```

The rest of the code will take care of loading in the content that was stored in that sendersTransmission.txt file (such as IV, MACKey, etc.)

Now, on lines 55 and 56, there will be a rsa.loadPublicteKey() method and a rsa.loadPrivateKey. In order for these to work, make sure to go to the RSA.java file and change the path to where you previously stored the public and private key .txt files.

```
public String loadPrivateKey() { //load public RSA key from sender so that we can encrypt the AES key
    try {
        privateKeyString = new String(Files.readAllBytes(Paths.get("/Users/ricardobaeza/Documents/privateRSA
    } catch (IOException e) { e.printStackTrace(); }

    System.out.println("String Private Key Loaded: " + privateKeyString);
    return privateKeyString;
}

public String loadPublicteKey() { //load public RSA key from sender so that we can encrypt the AES key
    try {
        publicKeyString = new String(Files.readAllBytes(Paths.get("/Users/ricardobaeza/Documents/publicRSAKe
    } catch (IOException e) { e.printStackTrace(); }

    System.out.println("String Public Key Loaded: " + publicKeyString);
    return publicKeyString;
}
```

After setting the public and private key paths in the RSA.java file, the Reciever.java file can now be ran.

As a result the following will be presented to the console:

```
RECIEVER
AES Encrypted Message: T2a4yvcMIXspxTYpslKP4kv7y5RPtN1ywgjNVA==
RSA Encrypted AES Key: W2/0ztZDQlaoy0LOlk0GNKJbElemEwuRkTQOfj/ONew9TxCGkYNR11aL/dVrj0W
IV: 7AKZ5dGKI/BEj4us
Senders RSA Encrypted MAC:eNAPakMcoeFuzpPimiREFZd3E1rn404uGE5Re9z/Y9XKzDNgZWPvW9XgfWfw
MAC Key:JqbJtDqFPNAksQakWSSHMLUGrwBMggIjuUKWTkpo9rg=
String Public Key Loaded: MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCBcDEh99VDw5hjF8zYzwic
String Private Key Loaded: MIICdgIBADANBgkqhkiG9w0BAQEFAASCAmAwggJcAgEAAoGBAIFwMSH31UPI
Senders MAC: a2Y//dEpEE1BgsfkUapXoBshd/jFkbSPZnxtk/Ww4YmaxT5XKSvgfWqZdUEaW2jsEUZnr6jQh,
hmac Key loaded: javax.crypto.spec.SecretKeySpec@588167a
MAC : a2Y//dEpEE1BgsfkUapXoBshd/jFkbSPZnxtk/Ww4YmaxT5XKSvgfWqZdUEaW2jsEUZnr6jQh/jBNY6lu
Recievers MAC: a2Y//dEpEE1BgsfkUapXoBshd/jFkbSPZnxtk/Ww4YmaxT5XKSvgfWqZdUEaW2jsEUZnr6j(
The message sent by the sender was successfuly authenticated
Decrypted Message: Hi reciever!
Decrypted MAC:JqbJtDqFPNAksQakWSSHMLUGrwBMggIjuUKWTkpo9rg=
```

As it can be seen, the message was deemed as authentic and the message that was sent by the sender was decrypted so that the receiver can read their message in plaintext.

Receiver to Sender:

Senders End

Before running the program start off with the senders program, open the Sender.java file. There will be three sections Part 1 (which will be used for receiver to sender communication) (done previously) Part 2 (which will be used for sender to receiver communication) and Part 3. When Running this for the first time, make sure that all the contents of Part 1 and Part 3 are commented and Part 2 are uncommented (Basically lines 40 to 74 are commented & line 82 and on). Another thing to keep in mind is to make sure you choose a correct path on your machine to store the sendersRSAPublicKeyPath(); and also in the RSA.java file for the savePrivateKey method, also choose a designated place on your machine to store the sendersPrivateRSAKey.txt file.

```
//Part 2 where sender will send its own RSA keys
String sendersRsaPublicKeyPath = "/Users/ricardobaeza/Documents/sendersPublicRSAKey.txt"; //**
rsa.init();
rsa.exportPublicRSAKey(sendersRsaPublicKeyPath);
```

```
public void savePrivateKey(String privateKey) throws IOException {
    System.out.println("Private Key Written to File: " + privateKey);
    File output = new File("/Users/ricardobaeza/Documents/sendersrPrivateRSAKey.txt"); //**
    FileWriter writer = new FileWriter(output);
    writer.write(privateKey);
    writer.flush();
    writer.close();
}
```

After changing the paths to save the RSA Public and Private keys you may now run the Sender.java file. The following will be shown on the console and the respective files.

```
Public key: MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDHi8Q1JXqPbXECE3npMYW39nb009JL3mcwo
Private Key Written to File: MIICeAIBADANBgkqhkiG9w0BAQEFAASCAmIwggJeAgEAAoGBAMeLxDUl
Private key: MIICeAIBADANBgkqhkiG9w0BAQEFAASCAmIwggJeAgEAAoGBAMeLxDULeo9tcQITeekxhbf2
Public Key From Init: Sun RSA public key, 1024 bits
  params: null
  modulus: 14012594028444949514314247797626337358833488800961196331567577930491046215
  public exponent: 65537
```

sendersrPrivateRSAKey.txt

```
MIICeAIBADANBgkqhkiG9w0BAQEFAASCAmIwggJeAgEAAoGBAMeLxDULeo9tcQITeekxhbf2dvTT0kveZzCgeIc2Uq
Fs6gKesAvxDMPGVZN01O8Q2V3nHlrHqV/xwK30oopxVwOLZLIvRrNLDwDTJV/FD5s5Ul/0xxl160Azfko5c+/
YUamgo/Mtn6w3opnPdp5/
arbtu6p1ECawNqciJNWEThbdAqMBAAECgYEAtB05Tkt1FA9SCPCxqsZ7i3B9MUIdOPQXqnG+QTXUYfbqRtsbxruerP
vR9JZpORy5RyQ9Zu1/uaWKKLp/GH1zbpuqE4QnSU3/
zhIsqmxA9yVteU7Q1WvRVW9kUxV6v0sU5XCoOlVdV8kgKuTrF4fCxi1kDkWKuamXNam77pzhqlUCQQDxw+rR7oQ7CK
xGqP0srcYY6awJSepZFFzGEtGGxFT+j0SmiZ3iu2eL2blTPu5xi/
ykuMVPUvNTLiDAtJ4HUdnHAkEA00t8K90G0IZx1FnzufiuyVxV7EADHqC1u7u87ZF1r5GRa05Xq6vSkS/
3A7BwbiOMM680xaud3VfhGKdp+ZfqOwJBAQc7tWpYsMwtvWhP1OZYxnGpfUvdEHFm0lz4PsKhIZHfwY9vbD4WTiEqa
uC9/
h1IALojp0P5WJUkN1tY5RSnKp0CQQCeE00Ua8lbXBnw6lMDAN+S2beWhdVPpqAPkUao3TgUX21RwEvqrGNeJ2o/
olKas/
FFAWSl0dzK6aIHlcgc6OBdAkBvJRcDkZSvs+JlnW3WZdICTqK70qK13h5JtVz9T8CDid9YAhl5rEJ3qaeCaGTWm9yI
qNOxl6s5vIy29zeLmOVd
```

sendersPublicRSAKey.txt

```
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDHi8Q1JXqPbXECE3npMYW39nb009JL3mcwoHiHNlKhbOoCnrAL8Q
zDxlWTdNTvENld5x5ax6lf8cCt9KKKcVcDi2SyL0azSw8A0yVfxQ+bQVJf9McZdetAM35KOXPv2FGpoKPzLZ+sN6KZ
z3aef2q27buqdRAmsDanIiTVhE4W3QIDAQAB
```

Receivers End

In the reiever.java program there will be a Part 1, Part 2, and Part 3 make sure Part 3 is the only one uncommented. Lines 31 to 81 should be commented.

On line 89 of the Reciever program, the program will load in the message that it would like to send to the user which is the recieversMessageToSender.txt which is already hard coded, however the messages path must be changed to your respective path in which the txt file is loaded (if it makes it easier you can create your own sendersMessage.txt and change the path of ‘messagesPath’ on line 85

```
message = load.loadinMessage(messagesPath);
```

```
String messagesPath = "/Users/ricardobaeza/Documents/ComputerScience/EclipseProjects/Crypto
```

Now on line 86, make sure to change the path of the sendersRsaPublicKey path that was previously chosen in the Sender.java file so that the receiver file can properly load in the public RSA key.

```
String sendersRsaPublicKeyPath = "/Users/ricardobaeza/Documents/sendersPublicRSAKey.txt";
```

Now on line 87, it is important to choose the path where you would like to store recieversTransmission.txt file, make sure to change the path specific to your machines directories.

```
String transmittedDataPath = "/Users/ricardobaeza/Documents/recieversTransmission.txt";
```

On line 94, make sure to write a path where you would like to save the AES encrypted message as it cannot be read of the transmission file on the sender's end.

```
String AESencryptedMessagePath = "/Users/ricardobaeza/Documents/sendersAESEncryptedMessage.txt";
```

Now, you are able to run the Reciever.java file. As a result of running the file, you will see the following text on the console and the respective txt file.

```
RECIEVER
Message Loaded: Hi sender!
AES Encrypted Message: ClCRTtxAhzMiLUT7sqUUaptjfxAqZuW9y2A=
Secret Key: zPo4jG7QGRfnlvG1Wd5ElA==
IV: Ciz21vB97l5fuknf
RSA Public Key: MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDHi8Q1JXqPbXECE3npMYW39nb009JL3m
Encrypted AES Key: ENqe5koQvvJ/bSFHQ+SqOZ4kug4hzhKBvGa74HiZ9Z9D/pZRChoYc9//QAm9DsqHXNv
Encrypted Message: ClCRTtxAhzMiLUT7sqUUaptjfxAqZuW9y2A=
MAC Key: ZmXF3ADhw3MIh+46KeF2FSphdQlhD+UbufUDKGA6Qpo=
MAC : bFJAfxqJ6lXjNagkWdQMUs7cNbCPlxq/VnUrAlp6hJkfhNpTOrfU4q6hhvt0nSM6oE2EH4TvBFPsViG3
```

recieversTransmission.txt

```
ClCRTtxAhzMiLUT7sqUUaptjfxAqZuW9y2A=
ENqe5koQvvJ/bSFHQ+SqOZ4kug4hzhKBvGa74HiZ9Z9D/pZRChoYc9//
QAm9DsqHXNvv3QrLEr9X0a4x8UD3WtH6AvnFEMWVPIjhWUS+M7LZDxyrRMbZPVKXpaB/
Sj7tZtLmCrytVNZ1K72AizcqP4tWPc8J3/WqAlNq1FcC64A=

Ciz21vB97l5fuknf


gNYirvs1QuoCUIh571GD3s01ldC7mR8100Ii3EJ6q9LAtA8owyc3iFt907lqkDNI4AeKFtNXpkr+EXufoG/
CJr1TmB92jaP1s/NKqXuLmHjnCt9Q3GRShELybAsPZZS7+Chqb5uXrjpSFKQicr1xFlprCzZC9Ydn4zlfArwJCEI=



ZmXF3ADhw3MIh+46KeF2FSphdQlhD+UbufUDKGA6Qpo=
```

Back to the Sender

In this portion we will be going back to the Sender and go to the LoadRecieversTransmission.java file and change the BufferReader br path to the actual path that was previously set to the recieversTransmission.txt file based on your machine.

```
BufferedReader br = new BufferedReader(new FileReader("/Users/ricardobaeza/Documents/recieversTransmission.txt"));
```

On line 90 of the sender.java file enter the encryptedAESMessagePath that was previously set based on the path chosen previously specific to your machine.

```
String encryptedAESMessagePath = "/Users/ricardobaeza/Documents/sendersAESEncryptedMessage.txt";
```

In the RSA.java file there are also two methods the loadPrivateKey method and the load publicteKey method, make sure to place the correct path to where you previously stored the keys previously.

```
public String loadPrivateKey() { //load public RSA key from sender so that we can encrypt the AES key
    try {
        privateKeyString = new String(Files.readAllBytes(Paths.get("/Users/ricardobaeza/Documents/sendersrPrivateRSAKey.txt")));
    } catch (IOException e) { e.printStackTrace(); }

    System.out.println("String Private Key Loaded: " + privateKeyString);
    return privateKeyString;
}

public String loadPublicteKey() { //load public RSA key from sender so that we can encrypt the AES key
    try {
        publicKeyString = new String(Files.readAllBytes(Paths.get("/Users/ricardobaeza/Documents/sendersPublicRSAKey.txt")));
    } catch (IOException e) { e.printStackTrace(); }

    System.out.println("String Public Key Loaded: " + publicKeyString);
    return publicKeyString;
}
```

Once the correct paths have been set in the program, you are now able to run it. As a result you will see the following information presented on the console:

```
AES Encrypted Message ClCRTtxAhzMiLUT7sqUUaptjfxAqZuW9y2A=
RSA Encrypted AES Key: ENqe5koQvvJ/bSFHQ+SqOZ4kug4hzhKBvGa74HiZ9Z9D/pZRChoYc9//QAm9Dsql
IV: Ciz21vB97l5fuknf
Senders RSA Encrypted MAC:gNYirvs1QuoCUIh571GD3s01ldC7mR8100Ii3EJ6q9LAtA8owyc3iFt907lql
MAC Key:ZmXF3ADhw3MIh+46KeF2FSphdQlhD+UbufUDKGA6Qpo=
String Public Key Loaded: MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDHi8Q1JXqPbXECE3npMYW3
String Private Key Loaded: MIICeAIBADANBgkqhkiG9w0BAQEFAASCAmIwggJeAgEAAoGBAMeLxDUleo9
Senders MAC: bFJAfxqJ6lXjNagkWdQMUs7cNbCPlxq/VnUrAlp6hJkfhNpTOrfU4q6hhvt0nSM6oE2EH4TvBl
hmac Key loaded: javax.crypto.spec.SecretKeySpec@fa77c91e
MAC : bFJAfxqJ6lXjNagkWdQMUs7cNbCPlxq/VnUrAlp6hJkfhNpTOrfU4q6hhvt0nSM6oE2EH4TvBFPsViG3
Recievers MAC: bFJAfxqJ6lXjNagkWdQMUs7cNbCPlxq/VnUrAlp6hJkfhNpTOrfU4q6hhvt0nSM6oE2EH4T
The message sent by the sender was successfuly authenticated
Decrypted Message: Hi sender!
Decrypted MAC:ZmXF3ADhw3MIh+46KeF2FSphdQlhD+UbufUDKGA6Qpo=
```

As it can be seen, the message sent by the receiver was able to be fully decrypted (“Hi sender!”) and the message was able to be authenticated using HMACSHA512, letting the sender know that indeed the receiver was the one who sent the message and no one else is impersonating the receiver.

Security Analysis/Countermeasures

This system was designed to securely send a message from the sender to the receiver and vice versa as best as it possibly could have been done. Some of the things implemented to make sure the system was as secure as can be was using algorithms such as RSA, AES and HMACSHA512 in order to send messages between the two parties. Although these algorithms were used to encrypt data, I do feel that the key size chosen for these algorithms were not the best (which is one of the shortcomings). Take for example in RSA the key length chosen was 1024 bits whereas despite their being key sizes such as 2048; in terms of AES and HMACSHA512 the key length chosen was 128 bits, despite there being an option for 256 bits which can add on more security. Further analyzing the systems design, in a general manner I feel that if an attacker wanted to perform a man in the middle attack and replay attack they would be able to. Firstly, this would be able to take place when the receiver is sharing their private key to the sender, an attacker can take that public key, save it, later replay it to the sender, but at the same time the attacker would be able to generate their own AES, MAC generation algorithms and message, discard the senders info and send their own data (attackers data) to the receiver and pretend to be the sender and perform man in the middle. This is possible since there is really no nonce or challenge that is being made by the receiver or sender in this case, which makes this a really easy attack for the attacker. In addition, after even further inspection of my design, some of the things that I noticed

was that I did not encrypt my HMAC Key when sending it from the sender to receiver (and vice versa) which can possibly be intercepted by the attacker and the attacker may be able to take the message and change the HMAC Key; although I don't think that much damage can be done since the attacker (hypothetically) does not have the original message sent by the sender. Another thing that I noticed was that the AES IV was also not encrypted, but I don't think that it would be as big of an issue since they don't have the AES key, but I do think that it would have been good practice to at least encrypt it with the receivers public RSA key. Now for the biggest vulnerability that I found in my system was storing the RSA private key to a txt file on my own machine. Ideally speaking, an attacker may not have access to my computer's files, which means they wouldn't have access to the RSA private key (but is still really bad practice), and if they did most of the entire communication between the sender and receiver would be intercepted by the attacker. So with that being said, I do not think that the system I designed is that secure despite following the protocols that help maintain security and integrity (RSA, HMAC and AES) which comes to show that there can still be gaps in a system and they must be found in order to maximize the security and integrity of communication.