CYBR372 Assignment 2

Part 3

The first part of Part3 is that the client generates a random key and sends it to the Client before the server sends it back to verify that that is the session key to use. It uses asymmetric encryption to send the key back and forth.

Client:

```
$ java EchoClient "C:\Program Files\Java\jre1.8.0_201\bin\cybr372.jks" badpassword clientpassword 12
Stored keys at C:\Program Files\Java\jre1.8.0_201\bin\cybr372.jks
Client sending ciphertext ETAkUekrq2EAJNDImxm8wbWfH/rTqrgHD1COmLnbXkjNx2er09Wt2gS3br1qLEnHiEp3JHv676+4LOOMkzWC2UdF/t4z8dtftOCKqJpyuaKR
+7DlzKZMkvwgFxUaeLPUWC3wsRTF+TityIT9gJKCkCeAs7twR+szKcErdABRbHXyPpjCzlfG5FVWcWKlKGwY5wCu4se0C893KHOuCjuOSRxsxxZakKgkPGoxIWU1GYvUylaHAD
3Ov/GYX+aaK/tdwviDmR4y8Gi5OwYObtsl9JOMfvK65WC10ZuAilptOQ65Nx6mXORZNCDhgi9lmi9lwcHJBOXS1WElL3x7NMkL/A==
Signature Valid
Keys exchanged
```

The Client randomly generates an AES 128 key before sending it to the server. Once the server has received the key, it is set back to the client. The server's response is read in by the client and checks that the key is valid and that completes the key exchange.

Server:

```
$ java EchoServer "C:\Program Files\Java\jre1.8.0_201\bin\cybr372.jks" badpassword serverpassword
Stored keys at C:\Program Files\Java\jre1.8.0_201\bin\cybr372.jks
Waiting to complete Exchange
Key Received
Signature Valid
```

The server reads the key that the client has sent and decrypts it. It keeps that as the session key and encrypts it and sends it back to the client.

From there the session key is used to generate a client-server key and a server-client key before using those keys the implement a block cipher to send messages back and forth.

Client:

```
Client sending cleartext 12345678901234567890123456789012
Client sending ciphertext V2N/ZGRexw60xMH3oHkS1nCkhQSgy+ACO/5K+Ied3gwFlUL8jBxtGIy35apO3vqa
Client received cleartext 12345678901234567890123456789012
Client sending cleartext ABCDEFGH
Client sending ciphertext VxSzYBQ47N8IuEOVRVxe/xowqNeXOIaB+Df659w6efMITK9xgkYiZ56eamih4TkP
Client received cleartext ABCDEFGH
Client sending cleartext 87654321
Client sending ciphertext bB/EirIA/a9XBs7t45sDT4zsnAHeVSJ32Ta2bzdEOBFuINIwv3pIjaRM+INLOSWZ
Client received cleartext 87654321
Client sending cleartext 1
Client sending ciphertext ydB914OVTwwJlZyhTSik7CvDQCnoJKrRONXqF/I3r8BWtw3pRhw+hg892tzbJzKu
Client received cleartext 1
Client sending cleartext HGFEDCBA
Client sending ciphertext IDI9UdMXzRl/hSxrDt7GpZriqW3sjZoOlJe1SmAcOuIGIXWghI+LfJ9tNuiH6sqN
Client received cleartext HGFEDCBA
```

The Client encrypts the message with the client-server key and using the block cipher to encrypt it and then sending it to the server. When the server sends a message back, the client users the server-client key to decrypt and prints the message that the client has received.

Server:

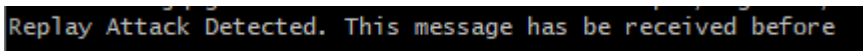
The server receives the encrypted message that the client has sent and decrypts it using the client-server key. It then sends the message back but encrypting using the server-client key.

As shown in the screen shots above, a message of any size can be sent. The message 12345678901234567890123456789012 which is 32 characters and 1 is sent which is 1 character so it can send messages from 1 – 32 characters.

Both the server and the client check that they get messages in the correct order. Therefore they can both detect if there are out of order errors.

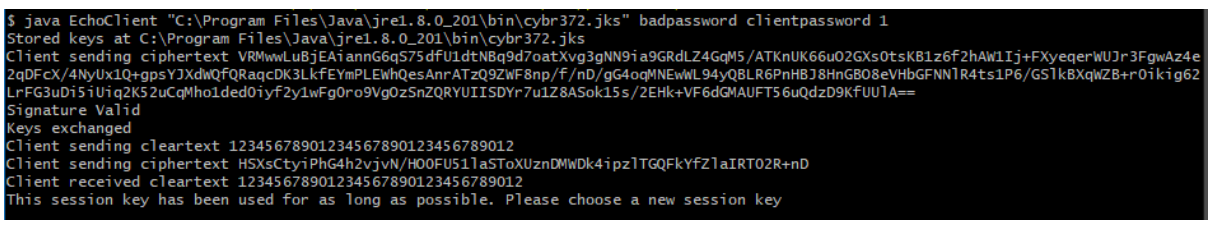
The server has found the message numbers don’t match and it has received a message with a higher message count than what it was expecting. This means that the messages are out of order and warns the user of this.

Since both the server and the client check that they get messages in the correct order that means that they can detect replay attacks of previous messages.



The program has detected that it has already seen before so it warns the user that it has been this message before.

Session length can be specified by the client. If the message count reaches that session length, then it tells the user that they have to pick a new session key.



In this case, a session length of one was selected. So the client was able to send one message before the client had to pick a new session key.

Running Part 3:

Client Arguments: (Key Store location) (Key Store Password) (Client password) (Session Length)

Server Arguments: (Key Store location) (Key Store Password) (Server password)

Key Store Password: badpassword

Client Password: clientpassword

Server Password: serverpassword

Design Decisions

The Client generates the master key before sending it to the Server and both programs derive the keys to use for Part 3. This allows both programs to use the keys for encryption and decryption.

When ensure that the messages from length 1-32 can be handled, I have padded the shorted messages so that every message is 32 characters long. I have done this by adding spaces on the end. This ensures that every message is the same length and it means that it is less likely to be errors and therefore the programs can handle messages from 1 – 32 characters long.

When using a tag for AES/GCM, I have made the tag of some of the information for each message. This includes the message count and the length of the message. This ensures that both the client and the server know what the tag as well as it being different each time. It means that if an attacker was to learn what the tag was, it wouldn't matter as much as it would be different the next time. This is also partly because it has also been hashed each time as well so it is different each time.