Cam Olssen - Cybr372 Assignment 2

Part 1:

Navigate to src/part1.
Run echoserver.java and echoclient.java with the following commands.
java echoserver.java **keysize**
java echoclient.java **keysize**
keysize must be a valid key length in bits for RSA encryption – 1024, 2048 or 4096 specifically.

Input the public key generated by the client as the destination public key for the server and vice versa.

EchoServer.java output for Part 1:

```
Public key:
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAixRoMf0r5dIWiwGDWW2hszEHNW9lPGcwCL1SYKf6Ew254JnX84esvP+YPvdx0BDHqJVXXLW72vD6z6xpWO65hn4K3AX8b2/aIa4S1vl0X9JJh7qXob
Destination public key: MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAmKgN6GbKtW1qO8FBBOqEcg8TV1z6G/Ek/De0887gicTUUMtGv4Lmmkpap/ObBt4jBC/4Q0FHtV95m+nKjU0Qj5Ogun
Received: CYBR372
Checking signature.
Signature matches!
Sent: 42 5C A9 A3 67 0B B5 C9 F3 D0 97 C9 53 27 D2 78 64 CF 44 91 A3 05 23 34 66 48 6C 4F F5 8E 3C 11 12 26 7C 30 9C 2B 57 67 EE 4E A1 BC 41 97 92 F5 FA BB 6F
```

Prints the received message from client - “CYBR372” – as plaintext and the response ciphertext. The response for part 1 is just the received message re-encrypted and re-signed.

EchoClient.java output for Part 1:

```
Public Key:
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAmKgN6GbKtW1qO8FBBOqEcg8TV1z6G/Ek/De0887gicTUUMtGv4Lmmkpap/ObBt4jBC/4Q0FHtV95m+nKjU0Qj5Ogun6ap2S9p8PXjrDFRU1If37cIrt
Destination public key:
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAixRoMf0r5dIWiwGDWW2hszEHNW9lPGcwCL1SYKf6Ew254JnX84esvP+YPvdx0BDHqJVXXLW72vD6z6xpWO65hn4K3AX8b2/aIa4S1vl0X9JJh7qXob
Sent 09 1B 31 E5 D2 15 C1 CE FB 25 93 43 4E 10 CC 3D 5B E5 CB 2A 9A 2C C5 61 30 BD DF 55 A1 98 64 08 03 AC 87 3A B9 BE F4 27 F4 14 A7 EA F0 34 EE 43 28 41 E3 2
Received: CYBR372
Checking signature
Signature matches!
```

Prints the sent ciphertext and the received message from server – “CYBR372” and verifies the signature.

Part 2:

Navigate to src/part2.
Run echoserver.java and echoclient.java with the following commands.
java echoserver.java
java echoclient.java

When prompted, enter **badpassword** as the password for both.

Keystore (cybr372.jks) was generated with the following commands.
keytool -genkeypair -alias server -keyalg rsa -storepass badpassword -keystore cybr372.jks -storetype PKCS12 -dname " CN=ROOT"

keytool -genkeypair -alias client -keyalg rsa -storepass badpassword -keystore cybr372.jks -storetype PKCS12 -dname " CN=ROOT"

EchoServer.java output for Part 2:

```
EchoServer ×   EchoClient ×
C:\Users\camol\.jdks\corretto-1.8.0_362\bin\java.exe ...
Input password: badpassword
Received: CYBR372
Checking signature
Signature matches!
Sent: 5E 9A 6A BD C3 F8 68 F2 7C 3E 5B 77 AD 18 83 C9 4D C7 79 87 76 41 E6 69 AB C9 95 FD DB 3E 93 2D 9E 98 E1 66 26 C7 4F CB 6E 70 47 19 B4 0F 0E F0 76 6A FC

Process finished with exit code 0
```

EchoClient.java output for Part 2:

```
EchoServer ×   EchoClient ×
C:\Users\camol\.jdks\corretto-1.8.0_362\bin\java.exe ...
Input password: badpassword
Sent 3A A9 0E 4D 6B 92 2E DB B0 EE 5D 48 B5 99 E8 0E D4 AF F0 E7 41 76 0B 57 0A 08 DE 9C F8 8F 84 9A 39 4A 96 91 D2 4D 78 DF 78 BB D6 38 51 AE BA 98 8A E0 3D C
Received: CYBR372
Checking signature
Signature matches!

Process finished with exit code 0
```

Design Choices:

I added methods to the Util class to get public and private keys from cybr372.jks. Now, instead of using the keys generated in the program, the client/server calls the required method to access the necessary public or private keys for encryption/decryption. This would be inadvisable in practice, as both sides of the conversation being able to access each other's private keys defeats the point of even having an encryption system.