**CSCE 5550**

**Lab:** 07

**Submitted by:** Md Khorrom Khan and Anand Kumar Bapatla

**UNT ID:** 11241118

# **Final Project**

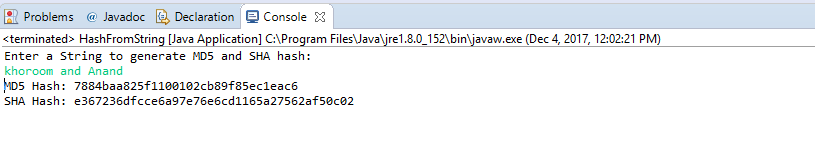
Implementation is done in java and execution is done in Eclipse Oxygen

**1. Message Digest**

This an implementation of MessageDigest class to generate hash from string. Program allows user to input a string in the terminal, uses MD5 and SHA scheme to hash the string given by user.

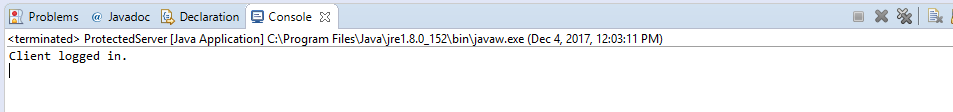
Steps to execute.

1. Execute HashFromString.java
2. Input the string which needs to be hashed
3. Hash will be displayed in console both for MD5 and SHA schemes as shown below.



**2(A). Authentication**

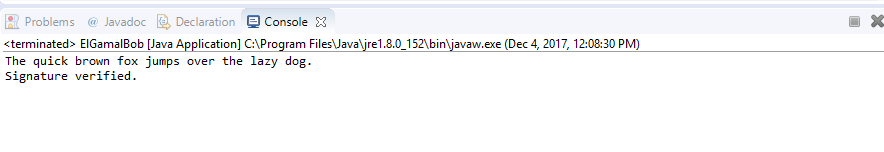
This implementation implements double strength password login using MessageDigest. It has three classes protection.java, ProtectedClient.java and ProtectedServer.java.

1. First Execute the ProtectedServer.java.
2. Execute ProtectedClient.java
3. The result will be shown in the console as below.

**2(B). Signature**

In this section we implemented ElGamal signature scheme. This implementation has two classes ElGamalAlice.java and ElGamalBob.java. Among these Bob receives the encrypted message sent from Alice and verifies the signature.

1. First execute ElGamalBob.java
2. Execute ElGamalAlice.java.
3. The signature will be verified, and the result will be shown in the console as below:

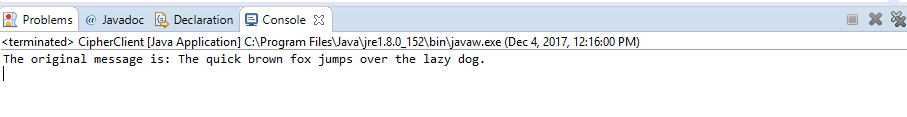


**2(C). Encryption**

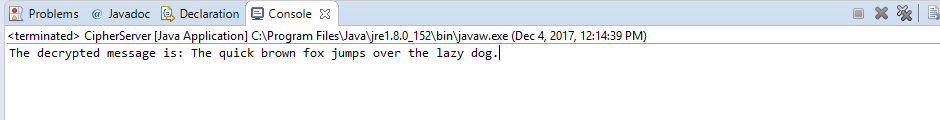
This implements Encryption system where a message is encrypted using DES key and sent to server. Server decrypt the actual message from the ciphertext using that DES key. There are two files CipherClient.java and CipherServer.java.

1. First execute CipherServer.java
2. Execute CipherClient.java
3. The decrypted message will be displayed in the server output console.

**Client output console:**



**Server output console:**

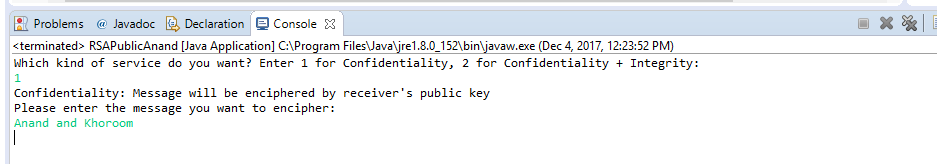


**2(D). Public-Key system**

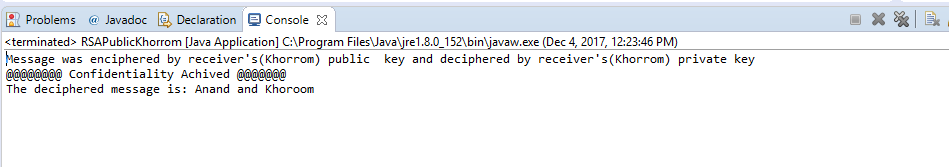
This implementation demonstrates the RSA public -key system in exchanging messages to achieve confidentiality or confidentiality and integrity. There are two classes RSAPublicAnand.java and RSAPublicKhoroom,java.

1. Execute RSAPublicKhoroom.java.
2. Execute RSAPublicAnand.java
3. It will ask for either confidentiality or both confidentiality and integrity. Choose either one of it by giving input as 1 or 2.
4. Next, we need to input the text message which needs to be communicated from Anand to Khoroom.
5. The result will be shown by displaying deciphered text.

**RSAPublicAnand output:**



**RSAPublicKhoroom output**

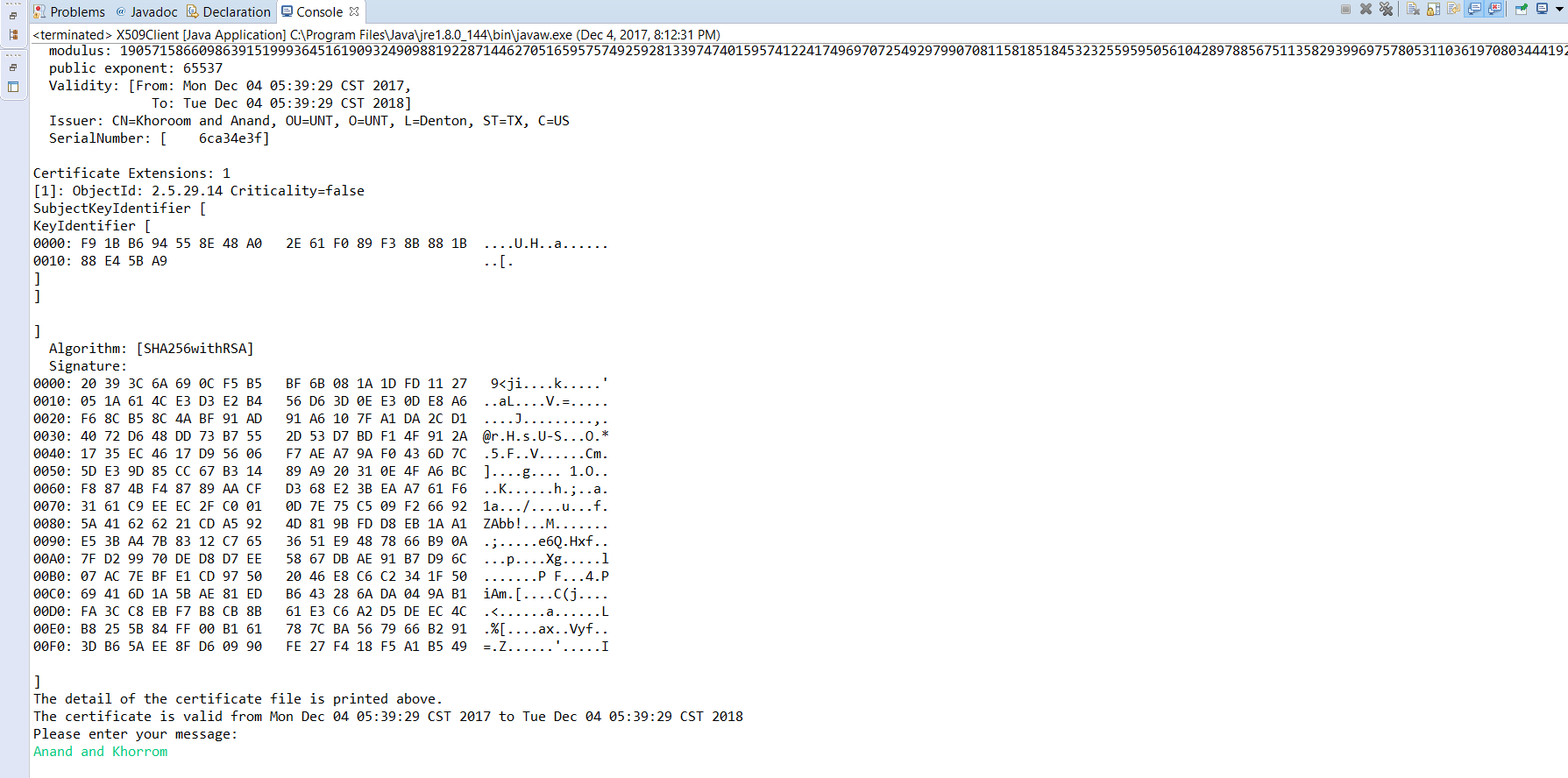


**2(E). X509 Certificates:**

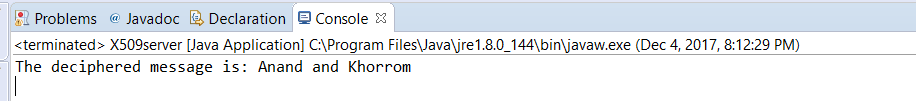
In this section, use of X.509 certificate in exchanging messages to achieve confidentiality is demonstrated. This has two classes, X509Client.java and X509Server.java and two files X509Project.cer and X509Project.jks.

1. First execute X509Server.java
2. Execute X509Client.java.
3. The client will verify the certificate expiration date and signature and then take a message input. It will be encrypted using server’s public key and sent to server.
4. The server will decrypt using server’s private key and displays the plain text message which was sent from client.

**X509Client console output:**



**X509Server console output**



**Question: What are the limitations of using self-signed certificates? What are they useful for?**

**Ans:**

**Limitations:**

Self-signed certificates are not verified by trusted 3rd party (i.e. Certificate Authority). They are risky for clients. Someone can setup a man in the middle attack by using their own self signed certificate to pose as the server. Since there is no chain of trust level and it’s not verified by CA, client doesn’t know if he is communicating to a legitimate server or not.

Self-signed certificates are not trusted by other applications/operating system. This may lead to authentication error. Self-signed certificates are not trusted by most of the web browsers, and it will be flagged as potentially risky. Error pop up will be shown to encourage users to not trusting the certificate which results in business loss.

Lifetime for Self-signed certificate is only one year.Advanced features of the server-side applications required Advanced PKI (Public Key Infrastructure) support which is not available for self-signed certificates.

It’s difficult to maintain self-signed certificated as it changes frequently and then it needs to be distributed to all the client. So, it’s not efficient for large number of clients.

**Usefulness of Self-Signed Certificates:**

Self-signed certificate provides same level of encryption as CA verified certificates without any cost. Self-signed certificate are great for test servers. Situation where privacy is a requirement of less concern self-signed certificate can be used. It’s easy and quicker to generate and server don’t need to deal with a 3rd party for its security. System which has small number of clients can use self-signed certificate.