CSCE 5550 Introduction to Computer Security

Final Project

Total Points: 150

Due Date: **Dec 3, 2015**

1. Message Digest [25 Points]

Write a program to demonstrate the use of hashing using **MD5** and **SHA** scheme and the **MessageDigest** class.

Your program should allow a user to enter a string; the program then hashes it and output the result.

You will need to consult Java API documentation to learn how to use java classes. You can download and install the documentation yourself, or you can access them from this URL: http://www.oracle.com/technetwork/java/api-141528.html

2. Various Crypto Techniques [125 Points]

The objective of this exercise is to use Java features to write some security mechanisms that can be used in applications. You will be using the classes from Java Cryptographic Extension to implement them.
You will need to consult Java API documentation. You can download and install the documentation yourself, or you can access them from this URL:

http://www.oracle.com/technetwork/java/api-141528.html

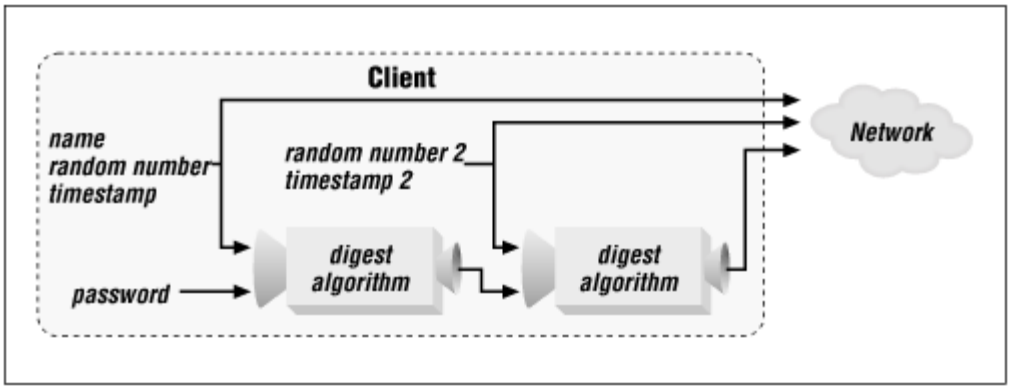
Java books that you can reference

> *Inside Java 2 Platform Security*, 2nd Edition, L. Gong, G. Ellision, M. Dageforde
> *Java Security*, Scott Oaks, O'Reilly

For each part of the assignment, skeleton Java code has been provided. These skeletons will NOT compile. You will need to make modifications on them before they can be successfully compiled and run.

A) Authentication

For the first part of the assignment, you should use the skeleton Java code to implement double-strength password login using message digest. The following diagram illustrates the double strength password.



Note that you need to generate 2 random numbers and 2 timestamps.

There are three classes defined:

- *Protection*, which provides three functions *makeBytes*, *makeDigest* (version 1), and *makeDigest* (version 2).
    - *makeBytes* takes in a long integer and a double, then converts them into a single byte array. *makeBytes* has already been implemented for you.
    - *makeDigest* (version 1) takes in a byte array, a timestamp, and a random number, then generates a digest using SHA. This function has already been implemented for you.
    - *makeDigest* (version 2) takes in a user name, a password, a timestamp, and a random number, then generates a digest using SHA. You need to implement this function. You may have to consult *MessageDigest* API in the documentation.

- *ProtectedClient*, which implements the client. There are two functions: *main* and *sendAuthentication*.
    - *main* is the starting point of the client program and has already been implemented for you. Make sure the host variable is set to the correct server address (it is currently set to cse.unt.edu).
    - *sendAuthentication* is the function that you need to implement. It takes in user name, password, and an output stream as the function inputs. In this function, you should implement double-strength password authentication and send to the server by writing to the variable 'out'. Consult *DataOutputStream* API on how to write different data types to 'out'.

- *ProtectedServer*, which implements the server. There are three functions: *main*, *lookupPassword*, and *authenticate*.
    - *main* is the starting point of the server program and has already been implemented for you. It creates a server process that waits for an incoming connection. Once a connection is established, *authenticate* is called to authenticate the user. If the user successfully authenticate, your program should print out "Client logged in."

- *lookupPassword*, which simply returns the password of the user stored on the server.
- *authenticate* is the function which you need to implement to authenticate the user trying to log in. Consult *DataInputStream* API on how to read data from the 'in' stream. The function should return either *true* or *false* depending on whether the user is authenticated.

B) Signature

In this part of the assignment, you are to implement the ElGamal Signature scheme described below.

The ElGamal Signature scheme is similar to the Diffie-Hellman scheme. It relies on the difficulty of solving the discrete logarithm problem. Choose a prime p and two random numbers g and d both less than p. Compute $y = g^d \bmod p$. The public key is the triplet (y, g, p); the private key is d.

Suppose Alice wants to send Bob a signed contract m. She chooses a number k that is relatively prime to $p - 1$ and has not been used before. She computes $a = m^k \bmod p$ and then uses the Extended Euclidean Algorithm to find b such that

$m = (da + kb) \bmod p - 1$

The pair (a, b) is the signature.

To verify the signature, check that $y^a a^b \bmod p = g^m \bmod p$

EXAMPLE: Alice and Bob decide to use the El Gamal digital signature scheme. Alice chooses $p = 29$, $g = 3$, and $d_{Alice} = 6$, yielding $y = 4$. She wants to send Bob the signed contract X (23). She chooses $k = 5$, which is relatively prime to 28. She computes

$a = 3^5 \bmod 29 = 11$

and then uses the Extended Euclidean Algorithm to solve for b:

$23 = (6 \times 11 + 5 \times b) \bmod 28$

This yields $b = 25$. She sends Bob the message $m = 23$ and the signature (11, 25).

Bob obtains the message and wants to verify the signature. He computes

$y^a a^b \bmod p = 4^{11}11^{25} \bmod 29 = 8$

and

$g^m \bmod p = 3^{23} \bmod 29 = 8$

Because the two match, Alice signed the message.

If someone learns *k*, the corresponding message *m*, and the signature (*a*, *b*), then she can use the Extended Euclidean Algorithm to recover *x*, Alice's public key.

EXAMPLE: Bob happens to learn that Alice signed the last message using $k = 5$. He immediately solves the following equation for *d*:

$m = (da + kb) \bmod p - 1 \rightarrow 23 = (11d + 5 \times 25) \bmod 28$

which yields $d = 6$. This is Alice's private key.

There are two classes in this assignment, *ElGamalAlice* and *ElGamalBob*, corresponding to the sender (Alice) and the receiver (Bob). The *main* functions for both the classes have been written for you. Your assignment is to write various functions that implement ElGamal key generation and signature creation algorithms (for Alice), and signature verification algorithm (for Bob). The functions you have to implement are indicated in the source files.

C) Encryption

In this part of the assignment, the client program *CipherClient* should (1) generate a DES key and stores the key in a file, (2) encrypts the given String object using that key and sends the encrypted object over the socket to the server. The server program *CipherServer* then uses the key that was previously generated by the client to decrypt the incoming object. The server obtains the key simply by reading it from the same file that the client previously generated. The server should then print out the decrypted message. For this part of the assignment, you will need to consult external sources and documentations on how to generate a DES key, write to or read from a file, and perform encryption/decryption of an object.

Most of the needed information should be available at:
http://www.oracle.com/technetwork/java/api-141528.html

D) Public-Key System

In this part of the assignment, you will need to demonstrate the use of RSA Public-key system to exchange messages that achieve confidentiality and integrity/authentication as well as the combination of both. You can assume a simple way of sharing public keys between the two communicating parties. You can reuse the basic code provided for above for this assignment.

E) X.509 certificates

In this part of the assignment, you will need to demonstrate the use of X.509 certificate to exchange messages that achieve confidentiality. The client loads the server's certificate and proceeds to verify its authenticity and validity. This includes checking the expiration

date and verifying if the certificate was signed using the private key that corresponds to the specified public key. Also, you need to print the content of the certificate you received. If the certificate received is valid, proceed to exchange confidential messages between the two parties. You can reuse the code previously provided for part C.

You will need to create an X.509 certificate for the Server, using **RSA** as the key generation algorithm. This certificate can be self-signed and it should have **your name**. You can use **keytool** for this purpose. **keytool** is installed as part of the standard Java SDK/Runtime library, and is located in its *bin* subfolder. Refer to the keytool's documentation for further instructions.
http://download.oracle.com/javase/1.5.0/docs/tooldocs/windows/keytool.html

Finally, **answer these questions** in the report.
What are the limitations of using self-signed certificates? What are they useful for?

Project Submission: Submit a **single ZIP file** with your *UNT email ID* as its filename via the Blackboard system. The package should contain all your source files and a *readme* file including the answers to the above questions. You also need to demonstrate your programs to the TA in person, where you may be asked to explain about different parts of your code. The date for demo will be announced later.