IN2935/TEL2810: Introduction to Computer Security
Due, Friday Dec 3, 2003
Programming Assignment (Score 300)

The programming assignment has three parts: *Authentication*, *Signature*, and *Encryption*. It is strongly suggested that you do them in that order, especially if you do not have previous Java experience. You need to understand the basic concepts of classes and how to use some java classes (from Java Cryptographic Extension). The objective of this exercise is to use Java features to write security code modules that can be used in applications.

Caution: *If you are new to Java the exercise may appear long and tough but it is actually easy once you understand the working of classes that need to be used. I will try to discuss some issues in the class.*

Before beginning the assignment, you should familiarize yourself with using Java on a SIS Unix machine (e.g., paradox.sis.pitt.edu). If you prefer, you can also use Java on your own PC but you will need to download and install Java Software Development Kit yourself.

Java is an object-oriented programming language. This means that you create classes of objects. In Java, the name of a source file must match the name of the public class defined and implemented in that file. So a class named ProtectedServer must be defined and implemented inside a file named ProtectedServer.java.

All Java source code files end with .java extension. To compile a source file name ProtectedServer.java, type the following at the command prompt:

```
javac ProtectedServer.java
```

If you have multiple source files in your project, you can compile all of them at once by typing:

```
javac *.java
```

The compilation produces files with .class extension. The number of files produced is the same as the number of classes that you have. The entry point of a Java program is the *main* function, which is defined in one of the source files. So if your *main* function is defined inside the ProtectedServer class in ProtectedServer.java file, you can execute your program by typing:

```
java ProtectedServer
```

Note that you only specify the class name, without the .class extension.

For example, the first part of your assignment asks you to implement double-strength password authentication. To run this part of the assignment, you should type in your first command line window: "java ProtectedServer" to start the server. Then on the second command line window, type: "java ProtectedClient" to start the client. You must start the server first.

You will most likely need to consult Java API documentation. You can download and install the documentation yourself, or you can access them from this URL:

http://java.sun.com/j2se/1.4.2/docs/api/index.html

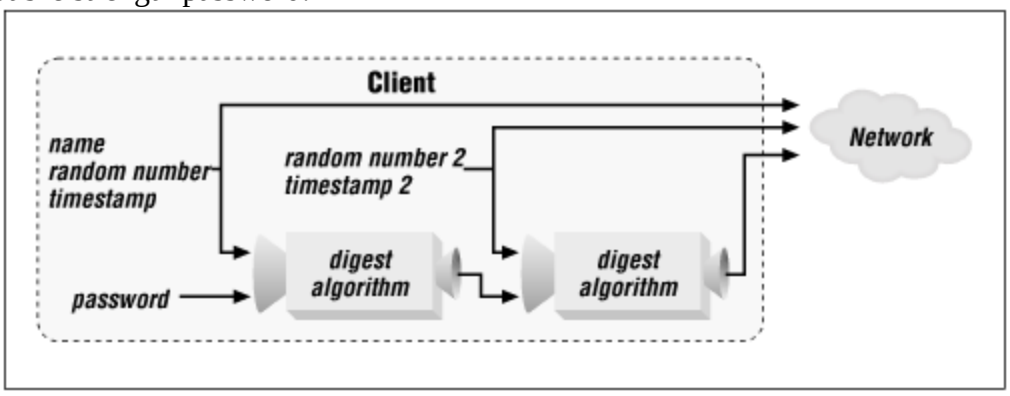
Java books that you can references that you

> *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 (100)

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 paradox.sis.pitt.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 (100)

In this part of the assignment, you are to implement the El Gamal Signature scheme described in the textbook in section 10.6.2.2.

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 El Gamal 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 (100)

In the last part of the assignment, the client program *CipherClient* should (1) generates 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, writing to or reading from a file, and perform encryption/decryption of an object. Most of the needed information should be available at:
http://java.sun.com/products/jce/doc/guide/API_users_guide.html

Submission: Submit all source files to the GSA via e-mail.