**CS255: Cryptography and Computer Security** **Winter 2005**

# Programming Project #1

Due: Monday, February 7th, 2005.

## 1 Overview

For the first programming assignment you will be adding security to a financial services system. The broker has some financial information that he would like to disseminate widely yet securely. For example this might be a video broadcast providing investing tips to the broker's clientele: only those clients who have an account with the broker should be able to recover the plaintext. The server provides account management on behalf of the broker.

The required security features are :

- Secure storage (password-protected) on the server of passwords corresponding to each client.
- Encryption of all communication with a block cipher in CBC mode.
- Integrity check for all communication using Message Authentication Codes (MACs).
- Authentication of the clients by the server.
- Resistance to replay attacks (on the server) by eavesdroppers.

We will examine each of these features in detail in the following section.

## 2 Security Features

### 2.1 Secure storage of client passwords

The authority server maintains a list of clients who may gain access to all hot tips. A client is identified by a username. The server has a mapping from the username to the client password. This information is pre-generated (before running the system) and should be read in by the server during initialization. The server stores the *client username* to *client password* mapping in an encrypted file. The key to this file is generated using an admin password (held by the server alone).

When a client connects to the server, the *client username* is passed as part of the client's message. The server looks up *client username* and gets the password for that client. The server will then be able to encrypt/decrypt client messages and generate/verify integrity of same using keys derived from that password. Effectively, the *client password* becomes the shared secret between the client and the server. Note that the client's password is never sent on the network.

### 2.2 Message Encryption

Each message transmitted by any party (client, server, broker) must be encrypted using a block cipher. You may use any standard block cipher you like, but all the messages must be encrypted using CBC mode. The cipher keys and MAC keys should be generated from the appropriate shared *password*: between the broker and server or between the client and server. The CBC IV is generated at random for each message and sent along with the ciphertext.

### 2.3 Integrity Check using MACs

Every message going over the network should have a MAC to enable detection of a malicious attacker tampering with the messages en route. Again the key for the MAC you decide to use should be derived from the appropriate shared *password*.

### 2.4 Authentication

The client will authenticate himself to the server via providing his username, the encryption of some random value under the block key generated from the client's password, and that random value. Of course the client will need to provide integrity over this authentication token as well.

### 2.5 Resistance to Replay Attacks

Even after you secure all the transmitted messages with encryption and MACs, there is still a replay attack possible. An eavesdropper can capture a message en route to the server. He/she can then repeatedly send the message — which is still a valid encrypted and MAC'd message — flooding the intended recipient. Your solution should prevent the attacker from successfully replaying a message to the server and, if the server receives a replayed message, he should revoke the user (permanently) under whose key the message was encrypted and MAC'd.

## 3 Components

### 3.1 BrokerGUI

The BrokerGUI takes a password, hereafter called "sharedPwd," which he shares with the AuthorityServer. This program also takes the name of the file containing the information to be encrypted; again, this could be a *.JPEG file or something as simple as a file containing a String (e.g. "Buy low! Sell high!"). For the purposes of this assignment, you may assume that the file contains text. The Broker also takes the name of the output file to which to write the enciphered and MAC'd data.

- The BrokerGUI will generate a block key, K-BC, and a MAC key, K-MAC, from the password he shares with the AuthorityServer.
- The BrokerGUI will also generate new, random key material, K, which he will use to create a new block cipher key, K-temp, and a new MAC key, K-MAC-temp.
- Then the broker will encrypt and MAC the contents of the input file:

$$E[\ \mathbf{K\text{-}temp},\ \text{file\_contents}\ ]\ \|$$
$$MAC[\ \mathbf{K\text{-}MAC\text{-}temp},\ E[\ \mathbf{K\text{-}temp},\ \text{file\_contents}\ ]\ ]$$

- Then the broker will encrypt and MAC the new, random key material, K:

$$E[\ \textbf{K-BC}, \text{K}\ ]\ \|\ \ MAC[\ \textbf{K-MAC}, E[\ \textbf{K-BC}, \text{K}\ ]\ ]$$

- Then the broker will write all of this to the given output file and exit.

## 3.2 AuthorityServer

The AuthorityServer acts on behalf of the broker in an account management role. The Authority-Server takes the same sharedPwd as above as well as an admin password (which he uses to encrypt and MAC the username/password file). The program also takes a port to listen on. Since the AuthorityServer will need to store the usernames and passwords securely, he'll need to take in a plaintext file containing that information (it is OK to hardcode this file name).

## 3.3 BrokerClient

The BrokerClient takes as an argument the ciphertext file written by the BrokerGUI (conceptually, this file is publicly available). Other arguments are: the host the server is running on (e.g. "elaine39"), the port number the server is listening on, and the client's username and password. The BrokerClient will then read in the ciphertext file provided by the broker and, with that, generate a message for the server proving his (the client's) identity.

- The client's password will be used to generate a block cipher key, K-BC-user1, as well as a MAC key, K-MAC-user1.
- The client will supply to the server (as obtained from the broker):

$$E[\ \textbf{K-BC}, \text{K}\ ]\ \|\ \ MAC[\ \textbf{K-MAC}, E[\ \textbf{K-BC}, \text{K}\ ]\ ]$$

- The client will also supply to the server an authentication token:

$$\text{R}\ \|\ \ \text{user1}\ \|\ \ MAC[\ \textbf{K-MAC-user1}, \text{R}\ \|\ \ \text{user1}\ ]$$

Where R is a new, random number generated by the client and "user1" is that user's username. Then the client transmits all of this information to the server. The server in response will decrypt the key material encrypted by the Broker, confirm its integrity, check the client's username, decrypt the user's R, check the integrity of that message then, once the server is satisfied that the client is who he says he is, the server will return the encryption of the new, random key material (K) under the client's block cipher key as well as integrity over this:

$$E[\ \textbf{K-BC-user1}, \text{K}\ ]\ \|\ \ MAC[\ \textbf{K-MAC-user1}, E[\ \textbf{K-BC-user1}, \text{K}\ ]\ ]$$

Then the client will be able to decrypt that key material, generate from it the block cipher and MAC keys (K-temp and K-MAC-temp), and, finally, decrypt and verify the broker's original ciphertext file.

## 4 Implementation

You will be using the JCE (Java Cryptographic Extensions) while programming for this assignment. You should spend some time getting familiar with the provided framework.

### 4.1 Getting the code

Download the *pp1.tar.gz* file linked on site to a directory in your account. Untar and unzip using the following command:

```
gunzip <  pp1.tar.gz |  tar xvf -
```

This should create the source tree for the project under the *pp1/* directory.

### 4.2 Description of the code

Here is a brief description of the files we provide. The files you need to change are in **bold**

| | |
|---|---|
| Makefile | Makefile for the project |
| **hotTip/BrokerGUI.java** | The broker program (a GUI) |
| **hotTip/BrokerClient.java** | The Client program (a GUI) |
| hotTip/AuthorityServer.java | The Server program (a GUI) |
| **hotTip/AuthorityServerThread.java** | The thread spawned to communicate with the client socket. |

## 4.3 Running the code

To build the project, enter the *pp1* directory and type *make*. To run the system, follow these steps:

1. Generate a plaintext file containing the broker's hot tip. Generate a plaintext file containing the usernames and passwords of authorized clients.

2. Run the broker; you will be asked for the sharedPwd, the plaintext file containing the hot tip, and the ciphertext file name to write the output to:

```
elaine21:~/pp1> java hotTip/BrokerGUI &
```

3. Pick a unique port number for your group to use. We recommend using the last four digits of your phone number. Note this wont work if your last four digits are less than 1024 or your roommate is taking the class.

4. Start the server; you will be asked for the admin password, the sharedPwd, and the port to listen on:

```
elaine21:~/pp1> java hotTip/AuthorityServer &
```

5. Start one or more clients. Note that the client does not necessarily need to be running on the same machine as the server. You will, however, be asked for the host and port on which the server is running as well as the broker-generated ciphertext file name and the client's username and password.

```
elaine21:~/pp1> java hotTip/BrokerClient &
```

## 4.4 Crypto Libraries and Documentation

Javas security and cryptography classes are divided into two main packages:
java.security.* and javax.crypto.*. They have been integrated into Java 2 Platform Standard Edition v 1.4. Classes for cryptographic hashing and digital signatures (not required for project 1) can be found in security, whereas ciphers and MACs are located in the JCE.

The following are some links to useful documentation :

- Java API
  *http://java.sun.com/j2se/1.4.1/docs/api*
- JCE Reference Guide
  *http://java.sun.com/j2se/1.4/docs/guide/security/jce/JCERefGuide.html*
- Java Tutorial
  *http://java.sun.com/docs/books/tutorial/*
- Chapter 6 from Java Cryptography by Jonathon Knudsen
  *http://www.oreilly.com/catalog/javacrypt/chapter/ch06.html*

Some classes/interfaces you may want to take a look at:

- javax.crypto.KeyGenerator
- javax.crypto.SecretKey
- javax.crypto.IvParameterSpec
- javax.crypto.Mac
- javax.crypto.Cipher
- javax.crypto.CipherInputStream
- javax.crypto.CipherOutputStream
- javax.crypto.SecretKeyFactory

- java.security.MessageDigest
- java.security.SecureRandom

- java.math.BigInteger

### 4.5 A Word About Networking

The AuthorityServer program runs for the duration: listening on the port specified. Every time a new client connects, the AuthorityServer spawns a new thread to communicate with that client. So multiple clients may be interacting with the server at once. The bulk of the work should be done in the AuthorityServerThread program. The client and server communicate over Java sockets. The code sets up these connections and examples are provided therein of writing to and reading from these sockets.

## 5 Miscellaneous

### 5.1 Questions

- We strongly encourage you to use the class newsgroup (su.class.cs255) as your first line of defense for the programming projects. TAs will be monitoring the newsgroup daily and, who knows, maybe someone else has already answered your question.
- As a last resort, you can email the staff at cs255ta@cs.stanford.edu

### 5.2 Deliverables

In addition to your well-decomposed, well-commented solution to the assignment, you should submit a README containing the names, leland usernames and SUIDs of the people in your group as well as a description of the design choices you made in implementing each of the required security features.

When you are ready to submit, make sure you are in your *pp1* directory and type /usr/class/cs255/bin/submit.