CS 470
Introduction to Applied Cryptography
Instructor: Ali Aydın Selçuk
Department of Computer Engineering, Bilkent University

Project #1: Secure Chat System
Proposal April 24, 2013 17:00
Checkpoint May 3, 2013 17:00
Implementation May 16, 2013 17:00

1 Description

In this project, you will be given a base code for a chat program and you will add authentication and privacy functionality to it. You will be free to choose, and to a degree, the design of your chat system and the way you provide authentication.

Your ultimate goal is that clients receive each others message in a properly encrypted and authenticated form. For example your system may work in one of the following ways:

- You can have chat rooms, among which clients choose the one they want to enter at the time of connecting the chat server.
- You can build a system where clients communicate one-to-one by establishing a session key beforehand.
  - If one-to-one communication is preferred, you are free to use a chat server, or clients can send messages directly to each other.

You can provide authentication by using either passwords or certificates/signatures. Depending on the way you choose, there may be three types of entities in the chat system.

- Certificates/signatures:
  - Chat server, denoted by S.
  - Chat clients, denoted by $C_1,\dots,C_n$.
  - Trusted certificate authority, denoted by CA.
- Password authentication
  - Chat server, denoted by S.
  - Chat clients, denoted by $C_1,\dots,C_n$.
  - Key distribution center (denoted by KDC) consisting of
    - Authentication server, denoted by AS.
    - Ticket granting server, denoted by TGS.

2 Deliverables

1. **Your protocols explained on a paper (nicely handwritten or typed).** First design the protocols that you will implement. Draw your protocols in the standard way we always use (Arrows between communicating parties). For readibility please use the standard notation used in the lecture slides ($[\dots]_{pri.key}$ for signature, $\{\dots\}_{pub.key}$ for public key encryption, $K_{sym}\{\dots\}$ for symmetric encryption)[1]. Do not forget to state which entities start with which keys or username-password pairs at the beginning clearly.

2. **Your complete source code together with the additional files such as the *keystores*.** Then implement these protocols by completing the base code given. You may add any classes you need. Make sure that your code is well commented and readable. The implementation given to you as a starting point already allows the clients chat with each other freely and insecurely through the chat server. Right now, there is a chat server which just forwards every message it receives to all clients, directly. You need to fix this gossipy system. Keystore files will be explained below.

3 Protocols

Depending on the way of authentication you choose, the protocols you need to design are summarized below.

- Certificate/Signature Authentication
  - A registration protocol between a client and the CA: client obtains its certificate
  - A login protocol between a client and the chat server
  - A messaging protocol between a client and the chat server
- Password Authentication
  - An authentication protocol between client and authentication server: client authenticates itself
  - A login protocol between client and ticket granting server
  - A messaging protocol between client and chat server

Important: Note that you need design these protocols using the ideas you learned in order to prevent attacks you saw such as dictionary attacks, etc. In your protocols and implementation, you can assume that the public keys of the servers are present at the client side. Do not forget to employ MAC in the messaging protocol.

4 Implementation

Like all software systems, your chat system first needs a careful design. Fortunately, it will not be a huge one, and most of it is already given. The system must work as follows:

- You will first run the servers with pre-defined port numbers (assuming that we will be running our codes usually on the same computer, we need to differentiate entity addresses using different ports instead of IPs). For obvious consistency and coordination reasons, please use the following port numbers for respective servers:

[1]See `http://cs.bilkent.edu.tr/~mustafa.battal/cs470/slides/cs470.Auth.ppt`

  - Chat server: 5555
  - Certificate authority: 6666
  - Authentication server: 7777
  - TGS: 7778
  - Clients: If you design a system where clients directly communicate, you can use ports 5001, 5002,... for clients listening ports

- Preparing keystores using **keytool** beforehand:
  - A keystore is a password encrypted file which contains public and private keys that an entity must store (its own private key, any others public keys, any secret keys currently used, etc.).
  - In your system, each entity should maintain a keystore file from the beginning.
  - Before running the system, public-private key pairs for all entites can be generated using keytool which is a command-line tool for generating keys easily. You can search the Internet to learn how you can use it.

- Chat rooms. If your system will use chat rooms and clients will talk to these rooms instead of communicating one-to-one:
  - Obviously, there must be at least two chat rooms. (Two is enough.)
  - During the first communication between client and server, client needs to obtain K_{room}, the symmetric key belonging to the room that it chooses.

5 Submission and Deadlines

Please be aware of the following

- You can work in groups of two.
- First of all, make sure that your code is well-commented. (Remember that quality is more important than quantity.)
- Prepare a `README` file that includes:
  - Your info, including your names and student IDs.
  - A concise explanation about how to run your code.
  - A list of all the usernames and passwords of the clients.
  - A list of keystore file names and their passwords.
- Send your java and keystore files, your `README`, and any other file that may be needed, in a compressed file to `mustafa.battal@cs.bilkent.edu.tr`

Deadlines are as follows:

1. **April 24, 17:00 -** Proposal due date. Briefly argue why your design will prevent dictionary attacks, etc.
2. **May 3, 17:00 -** Implement your designs except the messages sent are not final, but they are in simple format such as "$K_{sym}\{message\}$". Send this middle stage code via email. There might be a demo.
3. **May 16, 17:00 -** Send your code and other files via email. We will also schedule a demo time for each project group.

6 Starter Code

We have a starter code, which performs basic communication via sockets and threads. First, try to compile the code as it is, and make sure that you can run the chat server, CA or KDC servers depending on which you will use, and a few clients. Connect to the server and see that you can chat without any cryptographic protocols (which is precisely what you want to change). You can work in your favourite environment, as long as you provide clear and concise instructions how to run your code.

Here is a brief explanation of files in starter code:

- **ChatClient.java:** Class for the client entity. Obtains certificate from CA and connects to the chat server.
- **ChatServer.java:** Accepts chat requests (via secure connection) from clients.
- **ChatServerThread.java:** Receive messages from the clients and post messages to the appropriate chat room.
- **ChatClientThread.java:** Receive posted messages from the server.
- **ClientRecord.java:** Stores client information.
- **CertificateAuthority.java:** Starts up the CA.
- **CertificateAuthorityThread.java:** Accepts connections from the clients, verify their username-passwords, and issue certificates to them.
- **X509CertificateGenerator.java:** A class with a static method for creating X.509 certificates. May be a cleaner way of creating certificates.
- **CertificateAuthorityActivity.java:** Displays the activity of the client. Useful for printing debug output.
- **CertificateAuthorityPanel.java:** GUI class for the CA.
- **ChatLoginPanel.java:** GUI class for the login screen.
- **ChatRoomPanel.java:** GUI class for the chat room screen.

Note that you do not need to modify the GUI classes at all. In addition to these classes, you may need to provide the following classes, which will be completely independent from the framework above and run beforehand for setup:

- **KeystoreMaker.java:** This program will read a file of client (username, password, privilege) tuples and generates an encrypted file using a key generated from the CA password. You can simply see this as preparing the environment as if CA had registered the passwords of the clients beforehand, so that they can make password authentication.
- **IssueChatServerCertificate.java:** This program will sign the public key of the chat server with the private key of the CA and stores this signed certificate into the keystore of the chat server.

7 Useful Links

Do not confine yourself to these links only (you can find more on the Internet by simple searches), but here are some material that may be useful:

- Java API:
  `http://docs.oracle.com/javase/6/docs/api/`
- Keytool:
  `http://docs.oracle.com/javase/6/docs/technotes/tools/solaris/keytool.html`
- JSSE Reference Guide:
  `http://docs.oracle.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html`
- java.security.cert:
  `http://docs.oracle.com/javase/6/docs/api/java/security/cert/package-summary.html`

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

- java.security.SecureRandom:
  `http://docs.oracle.com/javase/6/docs/api/java/security/SecureRandom.html`
- java.security.KeyStore:
  `http://docs.oracle.com/javase/6/docs/api/java/security/KeyStore.html`
- java.net.ServerSocket:
  `http://docs.oracle.com/javase/6/docs/api/java/net/ServerSocket.html`
- java.net.Socket:
  `http://docs.oracle.com/javase/6/docs/api/java/net/Socket.html`