CS255: Cryptography and Computer Security **Winter 2008**

Programming Project #1

Due: Friday, February 8th, 2008.

1 Overview

For the first programming assignment you will be implementing a password manager, similar (but inferior in many ways) to your operating system's keychain software. The password manager must be able to operate securely either locally or over a network. It must be protected against both an adversary who takes over the network, and to a lesser degree against one who takes over the server.

2 Features

Any password manager's essential function is a secure map from the names of resources to their passwords, protected by a highly secret *master password*. For simplicity, we will assume that these are all strings. In particular, there are no additional data such as user name, site URL or notes. The software is divided up into a client, which manages the user interface, and a remote server, which stores the database of passwords. For simplicity, our server only supports a single user.

- The user must be able to add, remove and change (resource name, password) pairs. These pairs should be stored on the server, and should persist across restarts of both the client and server.
- The user must be able to change the master password at any time, without re-encrypting all of his stored passwords.
- The client software should not need to save any state between sessions. That way, it can be used on multiple machines at the same time without synchronizing state between the clients.

3 Threat Model

The password manager should be secure against the following attackers:

- Attackers who simply try to log into the server should only be able to mount an *online* attack against the user's master password. That is, they shouldn't be able to retrieve anything useful for an offline dictionary attack.
- Active network attackers should not be able to read any of the user's passwords. Nor should they be able to tamper with either the client or the server: they should't be able to convince the client to accept the wrong password (even an old one for the same resource), or to convince the server to change or delete a password. Of course, a network attacker can deny service; you do not need to make any effort to prevent this.

- Attackers who break into the server by some external means should not be able to read any of the passwords stored there, though they might tamper with them or delete them.

You are *not* required to protect the secrecy of the user's resource names or operations; that is, an eavesdropper may be able to determine whether the user is adding a new password, modifying one, or looking one up, and for what site. Nor are you required to protect the *length* of the stored passwords.

4 Cryptographic Requirements

4.0.1 Counter Mode (CTR)

You should use AES encryption in counter mode to protect the secrecy of the stored passwords (and, if you're doing the extra credit, the resource names).

Counter mode encryption generates a pseudorandom sequence by encrypting successive values of a counter. Formally, encryption of a message $(m_0, m_1, \ldots, m_n)$ is $(IV, E(k, IV) \oplus m_0, E(k, IV+1) \oplus m_1, \ldots, E(k, IV+n) \oplus m_n)$. As in other modes of encryption, new IV should be chosen randomly each time. Unlike most other modes of operation, counter mode does not require padding: the ciphertext length can be truncated to the length of the actual message without losing information.

4.1 Integrity Check using MACs

You will need to protect the integrity of messages on the network in order to prevent an attacker from modifying them while in transit. To prevent replay attacks without storing persistent state on the client, you should use a different MAC key in each session. You will still need a unique nonce on each message, but because of the per-session MAC, they can be unique-per-session instead of globally unique.

5 Components

5.1 Map

The primary functionality of the password manager is a secure, persistent, networked map from strings to strings. This functionality is developed in layers: **FileMap** provides a persistent map; **NetworkedMap** (along with **NetworkedMapServer** and **NetworkedMapServerThread**) exports it over the network; **EncryptedMap** provides secrecy and authentication; and **StringMap** translates to and from Strings using the UTF-8 character set.

You don't need to implement any of these maps: the only security-related one is EncryptedMap, and it's fairly trivial.

The maps included in this package do not quite conform to the Java map specifications in that they treat **byte** arrays as immutable objects. It is important to realize that because arrays are actually mutable, two arrays with the same elements are not considered equal by the Java standard libraries. As a result, **FileMap** and the like behave differently from, say, a **HashMap<byte[],byte[]>**.

5.2 Aes and Hmac

Aes and Hmac provide convenience classes over the Java cryptographic library. Hmac wraps the system implementation of HMAC/SHA1, and Aes uses HMAC and the system implementation of AES to implement authenticated AES counter-mode encryption.

You need to implement the Aes class; the Hmac class is provided for you.

5.3 BlobIO

BlobIO handles input and output using arrays of bytes (*blobs*), and, for convenience, arrays of arrays of bytes. Its instance FileBlobIO implements atomic file operations using temporary files and renaming semanatics. Its instance IOBlobIO uses the standard input/output libraries to send over pipes and network sockets.

You need to implement the SecureBlobIO instance, which will provide a channel whose integrity is protected from network attackers.

5.4 Client

The Client class implements the password manager's GUI. The current client is fairly limited; for instance, it cannot connect to any server other than localhost. You're welcome to improve this class, but it's not really the point of the project.

5.5 NetworkedMapServer

This class implements the network server, saving files in a directory called net_test. You don't need to modify it.

5.6 Test

The test class will conduct a simple series of tests over a virtualized network. It won't involve the GUI, and it can be built and run even if you don't have SWT installed.

6 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.

6.1 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:

tar xvzf pp1.tar.gz

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

6.2 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
pwman/Aes.java	The implementation of AES modes
pwman/SecureBlobIO.java	Cryptographic network protocol
pwman/Hmac.java	Wrapper class around HMAC/SHA1
pwman/BlobIO.java	Binary Input/Output module
pwman/Client.java	GUI client
pwman/NetworkedMapServer.java	Main server
pwman/Test.java	Test harness
pwman/EncryptedMap.java	Encrypted implementation of binary map
pwman/NetworkedMap.java	Network map protocol

6.3 Running the code

To build the project, enter the *pp1* directory and type *make*. To test the system, type *make run-test*. To use the client and server, type *make all run-server &* followed by *make run-client*. The client will only compile and run on a machine which has the SWT graphics library installed; this can be obtained on Ubuntu by typing *sudo apt-get install libswt3.2-gtk-java*. To erase created class files along with cores, emacs temporary files and the test and net_test directories, type *make clean*.

Note: Your solution will be tested on the elaine machines. So, please test your code on one of the elaines before submitting.

6.4 Crypto Libraries and Documentation

Java's 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 1.5. 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/1.5.0/docs/api
- JCE Reference Guide
  http://java.sun.com/j2se/1.5.0/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.Mac
- javax.crypto.Cipher
- javax.crypto.SecretKeyFactory
- java.security.SecureRandom

7 Miscellaneous

7.1 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.
- You can also email the staff at cs255ta@cs.stanford.edu

7.2 Deliverables

In addition to your well-commented solution to the assignment, you should submit a README containing the names, leland usernames and SUIDs of the people in your group and 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:
make clean
/usr/class/cs255/bin/submit.

7.3 Extra credit

The following options are available for extra credit.

- For minor extra credit, change EncryptedMap so that the keys to the map (the resource names) are kept secret as well as the values (their passwords).
- For major extra credit, change NetworkedMap to contact multiple servers using secret sharing. This will protect the password manager against a single failed or malicious server. If you decide to do this, don't worry about implementing a distributed transaction manager (which would be required for synchronization). Just assume that only one client will be making a request at any given time.

If you attempt the extra credit, be sure to explain its design in your README file.