Project 2 :: StealthNet
Security Engineering 2010

Matt Barrie
mattb@ee.usyd.edu.au
School of Electrical and Information Engineering
University of Sydney

April 15, 2010

Due: To be marked in labs, week of 14th May 2010

1 Introduction

In Project 1, you were tasked by some grey organisations to build StealthNet - a secure communications application for underground messaging and file transfers. Project 2 extends upon this in implementing a number of new security features, including a digital payment protocol with the FirstVirtual bank so that the aforementioned grey organisations can start making money by trading secrets.

Again, your team of two crack cryptographers have been retained to complete this project.

These new features include:

- Securing the key exchange protocol to be resistant to man-in-the-middle and session hijacking attacks.
- Implementation of routines to securely store sensitive information (e.g. private keys) to a file.
- Implementation of a payment protocol.
- Implementation of any remaining security features that were not implemented in Project 1 (e.g. replay resistance).

2 Securing the Key Exchange Protocol

You may have realised during Project 1 that key exchange protocols such as Diffie-Hellman are vulnerable to an active adversary through a **man-in-the-middle** attack. To prevent this, you should implement a system using public key cryptography. In particular, you should design a system using digital signatures to protect the Diffie-Hellman exchange which mutally authenticates the Client and the Server, and Clients to each other.

It is important in addition to preventing man-in-the-middle, your system is also resistant to both **replay** and **session hijacking** attacks.

Upon starting the Client, the following procedure for dealing with public keys occurs:

- The Client checks for the presence of already generated public and private keys in an encrypted file (using Password Based Encryption).
- If the keys are present, the Client uses these.
- If the keys are not present, the Client generates a public/private key pair and stores them in an encrypted file. A copy of the public key is sent to the Server.

The Server should act as a certification authority. It stores public keys for all Clients. A Client can request public keys for any user from the Server. You may assume that the Server's public key is well known to all Clients (either published in the Underground Times, or distributed with the StealthNet software). The procedure for dealing with the encrypted file is detailed in the next section (below).

3 File System

There are now secrets in the system which we wish to protect. These secrets will have to be stored on the file system somewhere between sessions. For example, we will need to protect the private (signing) keys used in the key exchange above.

To do so, we will make use of the JCE's built in capability for Password Based Encryption.
You must implement routines to securely read and write data:

- Ensure the confidentiality of each file by using a symmetric cypher with a a password.
- Ensure the integrity of any file you create with a MAC.
- Make bulk cracking of files difficult by using a salt in addition to increase the complexity of password cracking on a file.

4 Payment Protocol

The currency of choice for underground grey organisations is CryptoCredits. CrytoCredits are a form of digital money implemented with hash chains (otherwise known as hash stalks).

A hash chain is simply a series of numbers generated by repeatedly hashing a generator (or seed), e.g. $h(h(h(h(x))))$ is a hash chain of length 5 (where x is a random number). Say Alice generates a hash chain of length 100 and sends the top value, $T = h^{100}(x)$ to the server. You can think of this as a stack of 100 digital coins (in this case CryptoCredits). If Alice wants to spend the first coin in the stack, she sends $C = h^{99}(x)$. The server then verifies that $h(C) = T$. If this holds true, then the server knows the coin is valid. Say Alice wants to send another coin to the server, she sends $C = h^{98}(x)$. The server then verifies that $h(h(C))) = T$ for the coin to be valid, and so on.

Naturally, we do not want Alice to have a license to print money, so we add the presence of a Bank (a nameless Swiss bank that guarantees anonymity and does not ask questions) into the system. The Bank is responsible for authorising 'legal' tender. Note that clients **must** authenticate to the bank in the same manner as the Server (i.e. Diffie-Hellman secured against man-in-the-middle). You may choose to implement the bank within in the StealthNet server itself. The way in which the Bank authorises 'legal' tender is as follows:

- Alice generates a hash stalk of length n.
- Alice authenticates to the Bank using your secured Diffie-Hellman protocol.
- Alice sends the bank her identity, the length n of the hash chain, and the top coin of the hash chain, $h^n(x)$.
- The Bank verifies that Alice has enough money in her account (you may omit this if you wish).
- The Bank signs the tuple ("Alice", n, $h^n(x)$) and returns the signature to Alice.
- Alice can now start spending money from the hash chain.

There some subtleties that your system must be aware of:

- Alice must not be able to spend the same coin twice.
- Alice must not be able to spend more coins than there are bound by the signature for the hash chain.
- Once a coin has been spent once, it cannot be re-spent (why?).

Client buying information

- The StealthNet Server checks to see if there are sufficient CryptoCredits in the Client's StealthNet account (note: separate from the Client's Swiss bank account) to purchase the secret. If so, it deducts the amount from the account and facilitates the transfer of the sercret.
- If the Client does not have enough money in the StealthNet account to fully pay for the purchase, it sends a request for the amount extra required to the Client. If the Client has enough coins on it already (e.g. a partially used hash chain), it sends them to the Server. The Server verifies the coins, adds them to the Client's StealthNet account, then subtracts the full amount of the purchase and sends the information.
- If the Client does not have enough coins on it, it needs to withdraw some money from the Bank. First the client may send any partially used hash chain it may have as partial payment. Then it will need to generate a new hash chain to pay for the rest. To do this, the Client generates a new hash chain (with random x), and sends that to FirstVirtual for signing. The Bank checks the Client's FirstVirtual account to see if the Client has enough money (you may omit this step if you wish) and returns a signed hash stalk. The Client then sends coins to the StealthNet Server for payment. The StealthNet Server verifies the coins, adds them to the Client's StealthNet account, then subtracts the full amount of the purchase and sends the information.
- Once the transaction has completed, the StealthNet Server credits the seller's StealthNet account.

Note: as in Project 1, **all** messages between the Client and the Server must be protected by both a **MAC and a symmetric cypher in CBC mode**. You are free to use any MAC or cipher, provided that your choice is justified in the design document. Likewise, you are free to improve on the security mechanisms provided here, provided that an explanation of those improvements is given in the design document with appropriate assumptions and justifications for those decisions detailed.
If you have time, you may (optionally) implement a withdrawal protocol to allow Clients to get their money out of StealthNet.

5 Implementation

You will be implementing StealthNet in Java using Java 2 SDK which contains the Java Crypto Extensions (JCE) for implementing the security routines.

6 Documentation

You are to provide a two page design document outlining the security you implemented within your system and justification of your design choices. Make sure that your code is commented and in neat order.

7 Help

We will be continually monitoring the StealthNet forum (on the course website) for questions regarding the project. This is the first place you should look.

Thanks to all students that answered questions regarding Project 1 in the forums.

Tutors should be your next point of contact for assistance. As a last resort, you should email elec5616@ee.usyd.edu.au for help.

8 Marking

Your project is to be marked during your scheduled lab time. You must have your project marked during labs to receive credit. We will not be marking projects received by email, nor those provided on floppy disk, tape, punch cards, print outs or any other relic of the 20th century.