**CS255: Cryptography and Computer Security** **Winter 2006**

# Programming Project #2

Due: Monday, March 13th, 2006, 11:59 pm

## 1 Overview

For programming project 2 you will implement an instant messaging system with certain security features. Both passive and active attackers should not be able to learn or change the contents of any messages. All public keys should be signed by a certificate authority. All messages should be encrypted and signed. If a message containing a banned word is received, the sender should be removed from the system.

The project will involve various aspects of public key cryptography: key generation, certificates, revocation lists, encryption and decryption, and signing and verification. The project 2 code is independent from that of project 1, though some of the code you have written before could be useful. This project is larger in scope as compared to project 1, so please start early.

### 1.1 Requirements

For this project, you will need to do the following :

- Build public key infrastructure.
- Timestamp each message.
- Encrypt each message using the recipient's public key.
- Sign each message using the sender's private key.
- Implement revocation lists. Users are revoked when their messages contain certain words.
- Extra credit: hashed banned words and redactive signatures.

### 1.2 Quickstart

The instant messaging system is functional but completely insecure.

1. Type **make**.
2. Type **java IM/Setup**. This generates the CA's public and private keys. They are stored in the files **ca.pub** and **ca.priv**.
3. Type **java IM/CA**. This starts the CA.
4. Open another terminal. Type **java IM/Register alice**. This generates a key pair for the user alice, and also a certificate (stored in the files **alice.pub**, **alice.priv**).

5. Type **java IM/Register bob**.
6. Type **java IM/Receive bob**. This program waits for messages sent to the username bob.
7. Open another terminal. Type **java IM/Send alice bob 'Hello there'**. This sends a message from alice to bob.
8. Type **java IM/Send alice bob 'Tibet'**. This sends a message from alice to bob, but since it is a banned word, the CA is notified and "alice" is added to the revocation list.
9. Type **java IM/Send alice bob 'Hello'**. This sends a message from alice to bob, but since Alice is now banned so Bob does not receive the message.
10. The revocation list can be reset by running **java IM/ClearRL**. Once this is done Alice can send and receive messages again.

Note: To make it easy for the developer, all the programs and keys have been placed in the same location, and all connection attempts are made to "localhost". In reality, the Setup, CA, and Register programs and CA keys would be on a highly secured system, while each user would only possess the Send and Receive programs, along with their own private key, the public keys of their friends and the CA public key. Everyone should have the revocation list, though in this project we will not consider how it is distributed after being updated by the CA.

# 2 Description

The system consists of a Certificate Authority (CA) which issues certificates for public keys and also revokes them, and the users of the instant messaging system, who send each other instant messages. If a user receives a message containing a banned word, the CA is automatically notified and the sender is revoked.

## 2.1 Certificate Authority

To set up the system, first run **Setup.java** to generate CA private and public keys, and store them to disk. Use 1024-bit DSA keys.

Keys and certificates must be generated for each user of the system. The program **Register.java** does this. It generates a private and public key for a given username, and also a certificate for the public key.

Use 1024-bit RSA keys. In this system, we only want to tie a username to a certain public key, so the certificate can be the CA's signature on the message **(username, public key)**. The certificate should be written to the same file as the public key.

In real life, this step is quite complicated as a user would first need to prove that they are who they say they are before receiving a certificate. Furthermore, certificates usually have more than just some sort of identifier and a public key. They can contain expiry dates, subjects, serial numbers, etc. We are ignoring these details.

The CA also maintains an online presence, so that whenever a banned word is reported, it can instantly update the revocation list. Because of the way we have placed the files for this project, every program reads the same revocation list so once the CA updates the list, it is instantaneously updated for all the users.

In real life, there are issues with sending out updated revocation lists to every user, but again we are ignoring this detail.

### 2.2 Users

When a user wishes to be online in the instant messaging system, they run **Receive** *username*. Once this is done they can receive messages from other users.

To send a message, the user runs **Send** *sender receiver message*. If the receiver's public has been revoked, or is not certified, then the message does not get sent.

Each message should encrypted using the receiver's public key, and signed using the sender's private key. When a message is received, if the sender's public key has been revoked or is not certified then the message is ignored. It is also ignored if the signature does not verify.

Furthermore, each message should be timestamped to minimize problems caused by replay attacks. The current time and date should be attached to each message, and when received, displayed in international date and time notation, that is **YYYY-MM-DD HH:MM:SS**. Print this within parentheses and before the rest of the message, e.g.

**(2006-02-26 12:34:56) alice: Hello there**.

The encryption should be RSA encryption as defined in PKCS #1.

The order of signing and encryption is important. See the section below on this.

Lastly, if the message contains certain banned words, the CA is sent the message (in the clear) along with the sender's signature to prove that the sender did indeed transmit a banned word. The sender will then be revoked.

Note revocation lists are read on every message sent and received. This is often not the case in real life, as revocation lists could potentially be quite large and this approach is not practical. (This means there is a time window when messages will be accepted as genuine when in fact the key has been revoked.)

## 3 Files

The file **config** contains the port numbers used by the CA and Receive programs respectively. They can be any unused ports on the system you are developing on. You may need to change these numbers if other users on your system are using those ports.

The file **revocationlist** contains the initially empty revocation list. It is a Java array of strings (that must be read with Java object deserialization routines). As users are revoked, their usernames are appended to this file. Eventually this list should be signed by the CA.

The file **banned** contains the banned words. It is a Java array of strings (that must be read with Java object deserialization routines). It is currently set to the so-called "Three T's" that Google voluntarily censors in China.

## 4 Source Code

We briefly describe the provided Java files.

### 4.1 ClearRL

This program clears the revocation list. It also serves as a simple example of how to serialize Java objects and write them to a file.

### 4.2 PrintRL

This program prints the current revocation list. It also serves as a simple example of how to deserialize Java objects and read them from a file.

### 4.3 GenerateBanned

This prorgram generates the default list of banned words. It is also a simple example of Java object serialization. You will need to change this program for extra credit.

### 4.4 Message

This class defines the format of an instant message. It is presently three strings: one for the sender, one for the receiver and one for the message, all in cleartext. You will need to redefine this so that the message is timestamped, encrypted and signed.

The **Send** and **Receive** programs serialize and deserialize objects of this class when communicating.

### 4.5 Squeal

This class defines the format of a revocation request sent from **Receiver** to **CA** when a banned message is received.

The **Receive** and **CA** programs serialize and deserialize objects of this class when communicating.

### 4.6 Setup

This program is run once to setup the system. It generates a public key and private key for the CA. At the moment it creates the files "ca.pub" and "ca.priv" but does not write anything useful to them.

### 4.7 CA

In this system, Certificate Authority maintains a permanent online presence. If it receives proof that a certain user has sent a banned word in a message, it immediately updates the revocation list.

### 4.8 Register

This program generates public key, private key and certificate for a given username. For this system, the certificate only needs to tie the public key to the username, so a signature on the message **(username, public key)** is adequate.

At the moment the relevant files are created, but nothing useful is written to them.

Note: in real life, a user would generate a public key and private key themselves. When applying for an account, they would give their public key to the CA, who gives them a certificate back. We are ignoring these details in this project.

### 4.9 Receive

Once a user runs Receive, they are able to receive messages sent to them. Receive should use the user's private key to decrypt an incoming message, and the sender's public key to verify the signature. It also needs to check the sender has been certified and has not been revoked. Lastly, if an incoming message contains a banned word, the CA is sent the message along with the sender's signature so the sender can be revoked.

### 4.10 Send

This program is used to send a message to a user running the Receive program. It should check if the receiver's public key is valid (i.e. certified and not revoked). Then it should encrypt the message using the receiver's public key, and also sign it using the sender's private key.

## 5 Combining Encryption and Signing

Don Davis has a very accessible article ("Defective Sign & Encrypt in S/MIME, PKCS#7, MOSS, PEM, PGP and XML") on the subtleties of combining signing and encryption. We quote a couple of examples here. (Google for "don davis defective sign encrypt" to find it.)

Suppose Alice wishes to send a encrypted signed message to Bob, and sends the following:

$$E_B(\text{“I love you”}, Sign_A(\text{“I love you”}))$$

i.e. she signs, then encrypts the message. But after decryption, Bob can reencrypt and send

$$E_C(\text{“I love you”}, Sign_A(\text{“I love you”}))$$

to Charlie, who will be under the impression that Alice sent it to him.

Or suppose instead she tries encrypting first and then signing:

$$E_B(\text{“my idea”}), Sign_A(E_B(\text{“my idea”}))$$

But Charlie can intercept this and replace Alice's signature with his own before passing it on to Bob:

$$E_B(\text{“my idea”}), Sign_C(E_B(\text{“my idea”}))$$

Bob now thinks this message originated from Charlie.

One way to solve to these problems is to include the intended recipient of the message before signing and then encrypting. When Alice sends a message to Bob, she should transmit the following:

$$E_B(Sign_A(\text{“bob”}, \text{“message”}), \text{“message”})$$

Messages in our system should be encrypted and signed as above. (The timestamp is considered to be part of the message.)

Incidently, it is also possible to fix these problems when encryption precedes signing by including the sender's username:

$$Sign_A(\text{“message”}, E_B(\text{“alice”}, \text{“message”}))$$

but then Bob has to send more to the CA to report banned words.

## 6 Extra credit: Hashed Banned Words and Redactive Signatures

Firstly, having the taboo words in cleartext may not be desirable, as banning something can draw attention to it. For extra credit, store a list of hashes of banned words instead. This will require you to break each message into words, and to hash each word individually to find banned words.

Next, implement redactive signatures to send along with the normal signatures. Suppose Alice has sent a message to Bob containing a banned word. Bob wants to prove to the CA that she has said a banned word, but does not want to reveal the rest of the message. This cannot be done with normal signatures: the signature is on the whole message, and Bob would need to give the whole message to the CA so that the CA can verify it.

A redactive signature is a signature that can be later manipulated so that it becomes a signature on a single word. In other words, if Alice sends Bob a redactive signature on each message, Bob can extract the Alice's signature on a given word, and thus needs only to send that single banned word to the CA along with Alice's signature on that word.

We can use RSA to do this. Suppose Alice's private key consists of two large primes $p, q$ and her public key is $N = pq$ along with a random $a \in \mathbb{Z}_N$.

Creating a redactive signature on a message consisting of the words $w_1, ..., w_n$ works as follows. Alice computes their hashes $h_1 = H(w_1), ..., h_n = H(w_n)$, where $H : \{0,1\}^* \to \mathbb{Z}_N$, and computes $d = (h_1...h_n)^{-1}$ modulo $\phi(N)$. She sends the redactive signature $a^d$ to Bob.

To verify, Bob also computes the hashes on each word, raises $a^d$ to the power of $h_1...h_n$ (modulo $N$) and checks if the result is $a$.

Suppose Bob wishes to prove to a third party that Alice had used the word $w_i$ in her message. Then Bob raises $a^d$ to the power of $h_1...h_{i-1}h_{i+1}...h_n$ modulo $N$ (i.e the product of all the hashes except $h_i$) and sends this quantity $s$ to the CA. To verify, the CA checks if $s^{h_i} = a$ modulo $N$.

Strictly speaking, as presented, this scheme is not a signature because the order of words in the original message is not taken into account. This is fine since we are using a normal signature scheme on top of this scheme. Note we could preserve the order if desired by defining $h_i = H(i||w_i)$, but if we were to use this as our redactive signature, Bob would also need to tell the CA which word in Alice's message was banned, which gives a little more information away.

## 7 Notes

We recommend using Java object serialization to read or write keys, ciphertexts, etc. to disk or when sending over a network. Serializable java objects can also be signed. (Look up how to use ObjectInputStream, ObjectOutputStream. Some examples are in the starter code.)

Key generation and signatures should be quite straightforward. In fact, for these aspects of the project it is almost possible to simply cut-and-paste some of the code from the these links:

- key generation: http://javaalmanac.com/egs/java.security/GenKeyPair.html

- signing objects: http://javaalmanac.com/egs/java.security/SignObj.html

# 8 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

# 9 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 *pp2* directory and type /usr/class/cs255/bin/submit.