CS255: Cryptography and Computer Security **Winter 2000**

iButton extra credit

Due: Friday, March 10, 2000.

1 Getting Started

See the getting started handout for details on downloading software, APIs, and how to setup the extra credit project.

1.1 Starter Code

The startup code is available in .zip format. Expand the .zip file to find a project file (CS255_iButton.prj) and directory (CS255_iButton). Place both of these on the Windows desktop.

Inside the directory, you will find code for the Host and the iButton. Provided in the directories called Source are the relevant .java files. You will need to fill in the functionality on both the Host and the iButton sides. Also provided in the directory CS255_iButton are the compiled version of the starter code in the directories called Classes.

Files you will need to modify:

CS255_iButton_Host.java

CS255_iButton_Applet.java

When you start the iButton-IDE begin work by opening the project file: CS255_iButton.prj.

If you place these files in to a different directory, you will also need to change the class paths found at the very end of the CS255_iButton_Host.java file.

In both of the source files, you will find comments directing you to implement the various instructions and explaining various features of the iButton architecture.

2 Objectives

The high level goals of using he iButton are two fold: (1) protect the user's private keys by storing them on the tamper resistant iButton, and (2) enable the user to use any (untrusted) desktop to connect to the chat server. The desktop should never learn any information about the user's private keys. The iButton enables the user to sit at an Internet Caffe, authenticate himself to the chat server, and pay for posting messages to the chat server.

The minimum goal of the extra credit project is to move all of the signature-related functionality to the iButton. In particular, all private keys used by the client for the purpose of signing should reside on the iButton, and never ever leave the iButton. When the desktop software needs the client to sign a message (e.g. the client Diffie-Hellman message) it sends the message to the iButton, the iButton signs it and returns the signature. In addition, at registration time, the signing key should be generated and stored on the iButton. The iButton gives the public key to the desktop software which sends it to the CA to be certified.

In addition to storing keys on the iButton, the iButton must be protected by a user PIN. To activate the button (and enable it to sign messages) the desktop software gives the iButton a

valid user PIN (supplied by the user). Without the PIN the iButton will not answer any requests issued by the desktop software. This way, someone who steals your iButton cannot immediately impersonate you. For the iButton, PINs are actually Strings and not strictly limited to numbers.

Although not required, moving the PayWord (hash chain) functionality to the iButton would complete the iButton project. The starter code does not provide support for this part of the project. You are responsible for designing, implementing, and adding the needed instructions by yourselves.

In designing and implementing every piece of functionality, take care to ensure that all sensitive information remains entirely on the iButton.

2.1 iButton PIN

In the original project 2, the user private key is protected by JCE's Password Based Encryption. In an analogous fashion, you should implement password based protection for the iButton. In the starter code we provide you with all of the instructions that you need to implement. If you feel like you need to add more instructions, then use the "Add Instructions" option under the IDE's project menu. If you go through the example code, PinCheck.prj, you will find several examples of setting, checking, and managing the PIN on the iButton.

The first time the iButton is used, the user should be forced to set her PIN. Feel free to provide the functionality for changing PINs in a secure manner (i.e. enter your old PIN, enter your new PIN).

2.2 Signatures

Everything dealing with the user's digital signatures (DSA) should be handled on the iButton. If you look at the OpenCard security API (opencard.opt.security.*) which is imported for you in the applet starter file, you will find cryptographic functions for DSA signatures – like those used in the original project 2. As you know, these cryptographic primitives are operations on 1024 bit numbers. Fortunately, the iButton you will use for this project is equipped with a cryptographic accelerator coprocessor.

Just as in the original project 2, if the user does not have signing/verifying keys (for example, if this was the first time they used the iButton) you will need to generate these keys on the iButton. You will need to add private data for holding the key pair on the iButton in the Applet code.

After you have the signing/verifying keys generated, you should implement the instructions for retrieving the verification public key from the iButton. This is so that other parties can verify signatures generated by the iButton.

Finally, you will also need to be able to pass in messages to the iButton for signing. As described in class, a message is first hashed, then signed. You will need to implement both the hashing and the signing functionality on the iButton side.

Take a careful look at the example RSASign.prj. You will find example code that does the hashing before passing the result to a function that does raw RSA signing. In a similar fashion, you should implement this using the DSA signature scheme. Note that you may NOT hard-code the DSA parameters, like the example does for the RSA exponent and modulus – you must generate these on the iButton.

2.3 PayWords, hash chains

As mentioned before, this is the optional part of the extra credit project, worth more points. As in the original project 2, the base of the hash chain, your random value x should also remain protected

on your iButton, until it is spent. You will need to implement iButton applet methods to handle the interactions with the Bank and the Chat Server. Note that in addition to adding private variables to the iButton applet for the base of the chain (for the coins), this also means that you must store the Bank's signature on the message on the iButton and be able to retrieve it for the ChatServer.

For this section you will need to again add all of the relevant instructions on the iButton and handle them in the Host.

3 Project Details and Specifics

3.1 Emulator vs. iButton

The iButton-IDE comes with a very useful iButton emulator, but heed this very important warning: You must compile with the correct supporting classes and database files depending on whether you are compiling for the emulator or for the actual iButton.

Make sure the following Options (under Compile) are set depending on what platform you are building for.

The emulator is emulating version 1.00.0004 of the firmware which uses:

button33.jar and jib33.jibdb.

The iButton is version 1.03.0005 of the firmware which requires:

button36.jar and jib36.jibdb.

All of these files already come with the IDE, just change the file in the given default path (.classes).

3.2 APDU Sender

Another useful tool for developing/debugging for the iButton is the APDU Sender. This tool allows you to directly perform several operations like Master Erase, Load Applet, Current Loaded Applets. In addition it allows you to send commands by directly using the CLA, INS, P1, P2, and DATA byte arrays.

To understand how to use the APDU Sender, try testing it with the example projects Tutorial.prj and EchoTutorial.prj.

4 Integration with Project 2

After you have your iButton and Host code written and tested, it is time to integrate them into your original project 2. The difficulties here involved are mainly in porting your project from jdk1.2.2 to the jdk1.2.1 installed to run with the iButton. To this end, you will also need to download and install the JCE1.2 from:

http://java.sun.com/products/jce

Most of the other functionality that was used in the original project should remain intact. The main difficulty will be in setting up the Java 2 environment to run on the Windows platform. Expect to spend time tinkering with your DOS environment to get your original working project working before integrating.

After removing the main() from the Host, a natural place to integrate the extra credit (i.e. instantiate a Host object) would be the ChatClient or ChatConsumer depending on your implementation. Instead of only being able to login using a name and password, you should make the needed changes to allow iButton logins to the Chatroom.

Be sure to deal with the removal of the iButton during the chat session.

Again, integrating the PayWords part of the project will be extra work and receive commensurate credit.

5 Help

Like in the original project, use the newsgroup as a primary place for asking questions and finding answers.

Priority in office hours will go to helping people finish required class work.

And as a last resort, e-mail the staff at cs255ta@cs.stanford.edu.

6 Submission

Accompanying your well-decomposed, well-commented solution must be a README with the names, leland usernames, and SUIDs of the people in your group.

Instructions for how to submit the extra credit will be posted on the newsgroup before the due date of March 10, 2000, 11:59pm.