Cryptography & Network Security I Project SKB Protocol

Samuel Rhody, Kevin O'Connor, and Benjamin Kaiser

November 21, 2012

1 Library

The library we used for our implementation is Crypto++. Documentation is available at http://www.cryptopp.com/docs/ref/. For a list of the specific files we included, see util.cpp:13-18 or below:

```
#include ”includes/cryptopp/sha.h”
#include ”includes/cryptopp/hex.h”
#include ”includes/cryptopp/aes.h”
#include ”includes/cryptopp/ccm.h”
#include ”includes/cryptopp/gcm.h”
#include ”includes/cryptopp/osrng.h”
```

2 Compiling from Source and Launching

The makefile contains all necessary commands to compile from source, although compilation will require a working install of Crypto++. Executables for the Bank, ATM, and Proxy have all been provided as well.

To launch the interface, first run the Bank and Proxy files. Bank takes a single argument: a port number to connect to the proxy. Proxy takes two arguments: the port the ATM will connect to and the port the Bank will connect to. ATM takes two arguments: the port it will connect to the Proxy through and an ATM number, 1 through 50. These numbers represent individual physical ATMs. If you would like to connect multiple ATMs, you must use different ATM numbers.

A usage example follows (execute each command in a separate terminal window):

```
./bank 4444
./proxy 6666 4444
./atm 6666 1
```

3 Executing Transactions

The ATM is equipped with the following transactions: login, logout, balance, withdraw, transfer. The Bank is equipped with the following transactions: balance, deposit.

3.1 ATM Commands

The login command accepts one argument: the username of the account to be logged in to. It searches for a matching .card file, then prompts the user for the PIN, which is six digits. The PIN is masked with asterixes during input. The accounts and PINs are as follows:

Username	PIN	Initial Balance
alice	123456	$100.00
bob	234567	$50.00
eve	345678	$0.00

If the login fails for any reason (invalid user or invalid PIN), the user will still be allowed to proceed, but at the end of their transaction will get a ”transaction denied” error. This is to avoid leaking any information about the accounts.

If a login is attempted with a valid user but invalid PIN three times on the same user, the user's account will be locked by the bank. This can only be cleared by resetting the bank server - something we assume realistic adversaries would not be able to do.

The logout command terminates the ATM process immediately.

The balance command takes no arguments and returns the current balance of the logged-in user. The user is then logged out.

The withdraw command takes one argument: the amount to be withdrawn. If the argument is a valid amount, the account balance is deducted by that amount and the new balance is printed. If the argument is invalid, the user gets a ”transaction denied” error. The user is logged out at the end of this command.

The transfer command takes two arguments: the account to be transferred to and the amount to be transfered. If either argument is invalid (arg1 is not a valid account or arg2 is an invalid amount), the user gets a ”transaction denied” error. The user is logged out at the end of this command.

3.2 Bank Commands

The balance command takes one argument: the username of the account in question. It returns the current balance of that user.

The deposit command takes two arguments: the username of the account to deposit to and the amount to deposit. If both arguments are valid, the amount is deposited into the account.

4 Communications & Packet Handling

Communications between the ATM and Bank are shown in the table below. Nonces are referred to as N_b for the Bank and N_a for the ATM. The message is referred to as M. Login information is referred to as L. Hashing is referred to as $H()$. Return messages are referred to as r.

Step	ATM	Direction	Bank
1	$getNonce()$	→	Generate N_b & create session
2	Store N_b	←	$E(N_b)$
3	$E_x(L\|N_a\|H(N_a+N_b))$	→	$D_x(M)$, $validateNonce()$, $validateLogin(L)$
4	$D_x(L\|N_b)$, $validateNonce()$ == "I Got That"	←	$GenerateN_b$, $E_x(M\|N_b\|H(N_a+N_b))$
5	Generate N_a, command = $E_x(M\|N_a\|H(N_a+N_b))$	→	$D_x(M)$, $validateNonce()$, $executeCommand()$, return r
6	$D_x(L\|N_b)$ and print r	←	Generate N_b, $E_x(r\|N_b\|H(N_a+N_b))$, kill session

Table 3.1: Communications between ATM and Bank

During this whole transaction process, both the Bank and the ATM are keeping track of their state. The states are as follows:

Num	State
0	ATM requests. initial nonce from the bank and the two handshake.
1	Bank sends back a nonce
2	ATM sends login information
3	Bank sends an ACK
4	ATM sends a valid command for the bank
5	Bank replies with the response to command or the generic error message

Table 3.2: List of states that exist during transactions

After the fifth state, both the ATM and the Bank go back to state 0. At any point, the Bank can send a kill message to the ATM that will terminate the transaction and send both back to state 0.

5 Padding, Encryption, and Delays

All packets sent are 984 bytes. The maximum that could be sent is 1024 bytes, but because AES is a block cipher, 984 is the largest encrypted block less than 1024. Users input commands no larger than 204 bytes which are then compiled into messages and padded to a length no greater than 460 bytes. These packets are encrypted by AES, then sent after a random time delay no greater than one second (to thwart timing attacks).