Problem Set 5

Part A: Due on Friday, April 16, 2010. Part B: Due on Monday, April 26, 2010.

1 A Variant of the Feige-Fiat-Shamir Authentication Protocol

This problem is to implement a simple variant of the one-round Feige-Fiat-Shamir authentication protocol described in Lecture 18. The assignment is split into two parts. Part A is to generate the public and private keys for use by the protocol. Part B is to implement the protocol itself as an internet protocol using TCP sockets.

In our variant, Alice chooses a security parameter N. She generates a random modulus $n = pq$ that is the product of two distinct odd primes p and q, each of length $\lfloor N/2 \rfloor$. She then chooses at random a secret $s \in \mathbf{Z}_n$ and computes $v = s^2 \bmod n$. She makes the pair (n, v) public and keeps the pair (n, s) private. One round of the variant authentication protocol is shown in Figure 1.

	Alice		Bob
1.	Choose random $r \in \mathbf{Z}_n$.		
	Compute $x = r^2 \bmod n$.	$\xrightarrow{x}$	
2.		$\xleftarrow{b}$	Choose random $b \in \{0, 1\}$.
3.	Compute $y = rs^b \bmod n$.	$\xrightarrow{y}$	Check $y^2 \equiv xv^b \pmod n$.

Figure 1: One round of the variant FFS protocol.

Note that we have simplified the protocol of Lecture 18 in two ways: First, we have eliminated the need to compute inverses mod n. Second, we do not restrict the secret to $\mathbf{Z}_n^*$. Obviously it is desirable in practice to avoid secrets in $\mathbf{Z}_n - \mathbf{Z}_n^*$, but the protocol still "works" even with such secrets (in the sense that the real Alice will be accepted by Bob), and the probability of generating such a secret is too small to have much effect on the overall security.

2 Part A: Key Generator Assignment

Write a computer program `ffs-keygen` to generate public and private key files for use by the protocol. `ffs-keygen` takes two command line arguments, `N` and `user`, where `N` is the security parameter and `user` is the name of the user for whom the key is being generated. The program generates a public key pair (n, v) and a private key pair (n, s) as described above and writes its output to two files, `user.pub` and `user.prv`. File `user.pub` contains `user`, n, and v, one per line. File `user.prv` contains `user`, n, and s, one per line. The user name is a string, and the numbers are represented as decimal integers. Your program should handle values of $N \leq 4096$ and user names at most 31 characters long.

3 Part B: Protocol Implementation Assignment

Write two computer programs, `ffs-server` and `ffs-client`.

`ffs-server` takes two or more command line arguments: a repetition count `t` and one or more public key files. It reads the key files and builds a table of authorized users. It then opens a welcome socket, prints out the host name and port number of that socket, and waits for incoming connections. When contacted by a client, it identifies itself to the client and gets a user name from the client. It then runs t rounds of the protocol of Figure 1, trying to authenticate that user. You may assume that $t \leq 100$.

If all t rounds succeed, it informs the user of successful authentication and logs the outcome to standard output. If the check in line 3 of any round fails, it informs the user of unsuccessful authentication and logs the outcome to standard output, printing the round number which failed. If a bad or unexpected message is encountered, it informs the user of an error, logs the error to standard output, and closes the connection. Other error conditions, such as a broken network connection, should also be logged to standard output as appropriate. The server then goes back and waits for another incoming connection.

The server as just described can handle only one connection at a time. In real life, one would want a server that can handle multiple simultaneous authentication attempts by different users, but we do not require that enhancement for this assignment.

`ffs-client` takes three command line arguments: `host`, `port`, and a private key file. The client reads the key file and attempts to connect to the server at the specified host and port. If successful, it reads and prints the greeting message from the server. Next, it sends the user name contained in the private key file to the server and waits for a response. It then runs round after round of the authentication protocol of Figure 1 until the server either authenticates or rejects the client, or until an error is encountered. Whichever event finally occurs should be printed out, after which the client should exit.

4 Low-level Protocol

Client and server should communicate by alternate exchange of messages. Each message consists of a single line of text, terminated by a newline character. A message begins with a message type code, possibly followed by whitespace-delimited arguments. The allowed message types are given in Figure 2.

A sample successful run of the protocol is shown in Figure 3.

5 Programming Hints

Your program should be written in C or C++ and should use one of the big number libraries discussed in Lecture 8. You may use any of the provided functions in solving this problem. In particular, you do not need to implement your own primality testing function or random number generator if the versions provided by the package are adequate for this problem. I also do not require that the random number generator be cryptographically strong.

For those of you who are unfamiliar with sockets, I will provide working C code for a simple client and server that communicate via sockets.

Sender	Type	Parameters	Meaning
Server	`HELLO`	hostname, port	Initial greeting
Server	`UNKNOWN`		User is unknown to server
Server	`START`		Start a round of the protocol
Server	`QUERY`	b	The query bit b from line 2
Server	`AUTHENTICATED`		User was successfully authenticated
Server	`REJECTED`		User was rejected
Client	`USER`	username	My user name
Client	`QR`	x	The quadratic residue x from line 1
Client	`RESPONSE`	y	The response bit y from line 3
Either	`ERROR`	description	Error/illegal data received from peer

Figure 2: Message types.

Who	Action or message
Client:	Opens connection to server
Server:	HELLO frog.zoo.cs.yale.edu 2345
Client:	USER mike
Server:	START
Client:	QR 392285843992034...
Server:	QUERY 1
Client:	RESPONSE 49996843002020...
Server:	START
Client:	QR 969594645830992...
Server:	QUERY 0
Client:	RESPONSE 654923923432...
Server:	AUTHENTICATED
Server:	Closes connection and accepts new client
Client:	Receives message; closes connection and exits

Figure 3: A successful run of the protocol for $t = 2$.

6 Submission

You solution should be submitted two parts. The key generation assignment should be submitted as problem "5a". The protocol implementation should be submitted as problem "5b". Remember that the two parts have different due dates.