CS 380S - Theory and Practice of Secure Systems
Fall 2009

Homework #3

Due: 2pm CST (in class), November 19, 2009

NO LATE SUBMISSIONS WILL BE ACCEPTED

YOUR NAME: ______________________________

Collaboration policy

No collaboration is permitted on this assignment. Any cheating (*e.g.*, submitting another person's work as your own, or permitting your work to be copied) will automatically result in a failing grade. The Computer Sciences department code of conduct can be found at `http://www.cs.utexas.edu/academics/conduct/`

Homework #3 (30 points)

Problem 1

Consider the following variant of RSA encryption. Recall that an RSA public key is a pair (N, e). To encrypt some message m, first generate a fresh random value r of the same length as m. Use r as if it were a one-time pad to encrypt m (*i.e.*, let $s = m \oplus r$), and then encrypt r using plain RSA (*i.e.*, let $t = r^e \mod N$). The ciphertext is the (s, t) pair.

Problem 1a (2 points)

How does decryption work in this scheme?

Problem 1b (4 points)

Is this encryption scheme semantically secure? Explain.

Problem 2

Recall the oblivious transfer protocol between the Sender (S) and the Chooser (C) based on the hard-core predicate of a one-way trapdoor permutation. The Sender chooses a one-way trapdoor permutation F (let T be the trapdoor, and H the hard-core predicate of F). Let $b_{0,1}$ be the Sender's input bits, and let c be the bit indicating the Chooser's choice.

The protocol proceeds as follows:

$$\begin{array}{llll}
\text{S} & \rightarrow & \text{C} & F \\
\text{S} & \leftarrow & \text{C} & y_0, y_1 \qquad\qquad \text{where } y_c = F(x_c) \text{ for a random } x_c;\ y_{\overline{c}} \text{ is random} \\
\text{S} & \rightarrow & \text{C} & m_0 = b_0 \oplus H(T(y_0)),\ m_1 = b_1 \oplus H(T(y_1))
\end{array}$$

The Chooser computes b_c as $m_c \oplus H(x_c) = (b_c \oplus H(T(y_c))) \oplus H(x_c) = (b_c \oplus H(T(F(x_c)))) \oplus H(x_c) = (b_c \oplus H(x_c)) \oplus H(x_c) = b_c$.

Problem 2a (3 points)

Suppose the Sender is *malicious* rather than semi-honest. Is the above protocol secure? If not, explain precisely what a malicious Sender can do to make his view of the real-world protocol unsimulatable in the ideal world.

Problem 2b (3 points)

Suppose the Chooser is *malicious* rather than semi-honest. Is the above protocol secure? If not, explain precisely what a malicious Chooser can do to make his view of the real-world protocol unsimulatable in the ideal world.

Problem 3

Recall Schnorr's ID protocol, which is an honest-verifier zero-knowledge proof of knowledge of the discrete logarithm s of $t \mod p$, where p is a large prime, and g is a generator of an order-q subgroup. Let P be the prover, V the verifier.

P	$\rightarrow$	V	$x = g^r \mod p$	where r is a random value between 1 and $q-1$
P	$\leftarrow$	V	c	where c is a random k-bit value
P	$\rightarrow$	V	$y = sc + r \mod q$	

Verifier accepts the proof if $x \cdot t^c = g^y$.

Problem 3a (4 points)

Suppose Larry and Moe execute the above protocol with Larry acting as the prover and Moe acting as the verifier. Now Moe wants to convince Curly that he knows the discrete logarithm of t. He records the transcript of his conversation with Larry and gives it to Curly.

Should Curly be convinced that Moe knows the discrete logarithm of t? Explain.

Problem 3b (4 points)

Suppose that instead of sending his messages directly to Moe (the verifier), Larry (the prover) signs his messages and gives them to Curly, who forwards them to Moe. Similarly, Moe signs his messages and gives them to Curly, who forwards them to Larry.

Assume that they are using an unforgeable digital signature scheme, *i.e.*, it is not feasible for Curly to forge Larry's (respectively, Moe's) signature on a message that Larry (respectively, Moe) did not sign. Also assume that Larry knows Moe's public signature verification key, and vice versa.

Should Moe be convinced that Larry knows the discrete logarithm of t? Explain.

Should Curly be convinced that Larry knows the discrete logarithm of t? Explain.

Problem 4 (4 points)

Let (N, e) be Larry's RSA public key, and d the corresponding private key. Both Moe and Curly know a ciphertext c encrypted under Larry's public key. Moe claims that he knows the corresponding plaintext m (recall that with RSA, $m = c^d \mod N$), and that he can prove this to Curly without revealing m.

He does it via the following protocol:

- Moe generates a random number $r \mod N$, encrypts it under Larry's public key to obtain $x = r^e \mod N$, and sends x to Curly.
- Curly flips a fair coin.

  Coin comes up heads: Curly asks Moe to send him the plaintext of x, *i.e.*, $x^d \mod N$. Moe sends him $y = r$. Curly verifies the answer by checking $x = y^e \mod N$.

  Coin comes up tails: Curly asks Moe to sends him the plaintext of $c \cdot x$, *i.e.*, $(c \cdot x)^d \mod N$. Moe sends Curly $y = m \cdot r$. Curly verifies the answer by checking $x \cdot c = y^e \mod N$.

Prove that this protocol is zero-knowledge even if Curly is malicious. Hint: it is sufficient to write a simulator that with probability $\frac{1}{2}$ creates a transcript which is indistinguishable from the transcript of Moe and Curly's conversation.

Problem 5

Let N be a product of two large primes, and let $y = x^2 \mod N$ for some x (assume that y and N are relatively prime). Both Larry and Moe know y and N. Furthermore, Larry claims that he knows x.

Larry proves this to Moe as follows:

- Larry generates a random x_0 and sets $x_1 = xx_0^{-1} \mod N$. He sends Moe $y_0 = x_0^2 \mod N$ and $y_1 = x_1^2 \mod N$.
- Moe verifies that $y_0 y_1 = y \mod N$.
- Moe generates a random bit i and asks Larry for x_i. He then verifies that $x_i^2 = y_i \mod N$. If verification is successful, we say that Larry has *passed* the protocol.

This protocol is repeated until Moe is convinced (*i.e.*, Larry successfully passes every instance of the protocol).

Problem 5a (3 points)

If Larry does not know x, can he find y_0 and y_1 such that (i) $y_0 y_1 = y \mod N$ and (ii) he knows x_0 and x_1 such that $y_0 = x_0^2 \mod N$ and $y_1 = x_1^2 \mod N$? Why or why not?

Problem 5b (3 points)

Suppose Moe's bit i always comes out as 1, and Larry knows this. Can Larry exploit this to pass the protocol every time? (Larry could try to always set $x_1 = 1$, but let's assume that Moe checks for this.)