CS255: Cryptography and Computer Security **Winter 2009**

Assignment #3

Due: Wednesday, Mar. 4, 2009. In class.

Problem 1 Let's explore why in the RSA public key system each person has to be assigned a different modulus $N = pq$. Suppose we try to use the same modulus $N = pq$ for everyone. Each person is assigned a public exponent e_i and a private exponent d_i such that $e_i \cdot d_i = 1 \bmod \varphi(N)$. At first this appears to work fine: to encrypt a message to Bob, Alice computes $C = M^{e_{bob}}$ and sends C to Bob. An eavesdropper Eve, not knowing d_{bob} appears to be unable to decrypt C. Let's show that using e_{eve} and d_{eve} Eve can very easily decrypt C.

a. Show that given e_{eve} and d_{eve} Eve can obtain a multiple of $\varphi(N)$.

b. Show that given an integer K which is a multiple of $\varphi(N)$ Eve can factor the modulus N. Deduce that Eve can decrypt any RSA ciphertext encrypted using the modulus N intended for Alice or Bob.
Hint: Consider the sequence $g^K, g^{K/2}, g^{K/4}, \ldots g^{K/\tau(K)} \bmod N$ where g is random in $\mathbb{Z}_N$ and $\tau(N)$ is the largest power of 2 dividing K. Use the the left most element in this sequence which is not equal to $\pm 1 \bmod N$.

Problem 2. Time-space tradeoff. Let $f : X \to X$ be a one-way permutation. Show that one can build a table T of size B bytes ($B \ll |X|$) that enables an attacker to invert f in time $O(|X|/B)$. More precisely, construct an $O(|X|/B)$-time deterministic algorithm $\mathcal{A}$ that takes as input the table T and a $y \in X$, and outputs an $x \in X$ satisfying $f(x) = y$. This result suggests that the more memory the attacker has, the easier it becomes to invert functions.
Hint: Pick a random point $z \in X$ and compute the sequence

$$z_0 := z, \;\; z_1 := f(z), \;\; z_2 := f(f(z)), \;\; z_3 := f(f(f(z))), \;\; \ldots$$

Since f is a permutation, this sequence must come back to z at some point (i.e. there exists some $j > 0$ such that $z_j = z$). We call the resulting sequence $(z_0, z_1, \ldots, z_j)$ an f-cycle. Let $t := \lceil |X|/B \rceil$. Try storing $(z_0, z_t, z_{2t}, z_{3t}, \ldots)$ in memory. Use this table (or perhaps, several such tables) to invert an input $y \in X$ in time $O(t)$.

Problem 3 Commitment schemes. A commitment scheme enables Alice to commit a value x to Bob. The scheme is *secure* if the commitment does not reveal to Bob any information about the committed value x. At a later time Alice may *open* the commitment and convince Bob that the committed value is x. The commitment is *binding* if Alice cannot convince Bob that the committed value is some $x' \neq x$. Here is an example commitment scheme:

Public values: (1) a 1024 bit prime p, and (2) two elements g and h of $\mathbb{Z}_p^*$ of prime order q.

Commitment: To commit to an integer $x \in [0, q-1]$ Alice does the following: (1) she picks a random $r \in [0, q-1]$, (2) she computes $b = g^x \cdot h^r \bmod p$, and (3) she sends b to Bob as her commitment to x.

Open: To open the commitment Alice sends (x, r) to Bob. Bob verifies that $b = g^x \cdot h^r \bmod p$.

Show that this scheme is secure and binding.

a. To prove security show that b does not reveal any information to Bob about x. In other words, show that given b, the committed value can be any integer x' in $[0, q-1]$.
Hint: show that for any x' there exists a unique $r' \in [0, q-1]$ so that $b = g^{x'} h^{r'}$.

b. To prove the binding property show that if Alice can open the commitment as (x', r') where $x \neq x'$ then Alice can compute the discrete log of h base g. In other words, show that if Alice can find an (x', r') such that $b = g^{x'} h^{r'} \bmod p$ then she can find the discrete log of h base g. Recall that Alice also knows the (x, r) used to create b.

Problem 4 Access control and file sharing using RSA. In this problem $N = pq$ is some RSA modulus. All arithmetic operations are done modulo N.

a. Suppose we have a file system containing n files. Let $e_1, \ldots, e_n$ be relatively prime integers that are also relatively prime to $\varphi(N)$, i.e. $\gcd(e_i, e_j) = \gcd(e_i, \varphi(N)) = 1$ for all $i \neq j$. The integers $e_1, \ldots, e_n$ are public. Let $R \in \mathbb{Z}_N^*$ and suppose each file F_i is encrypted using the key $key_i = R^{1/e_i}$.

Now, let $S \subseteq \{1, \ldots, n\}$ and set $b = \prod_{i \in S} e_i$. Suppose user u is given $K_u = R^{1/b}$. Show that user u can decrypt any file $i \in S$. That is, show how user u using K_u can compute any key key_i for $i \in S$.

This way, each user u_j can be given a key K_{u_j} enabling it to access those files to which it has access permission.

b. Next we need to show that using K_u user u cannot compute any key key_i for $i \notin S$. To do so we first consider a simpler problem. Let d_1, d_2 be two integers relatively prime to $\varphi(N)$ and relatively prime to each other. Suppose there is an efficient algorithm $\mathcal{A}$ such that $\mathcal{A}(R, R^{1/d_1}) = R^{1/d_2}$ for all $R \in \mathbb{Z}_N^*$. In other words, given the d_1'th root of $R \in \mathbb{Z}_N^*$ algorithm $\mathcal{A}$ is able to compute the d_2'th root of R. Show that there is an efficient algorithm $\mathcal{B}$ to compute d_2'th roots in $\mathbb{Z}_N^*$. That is, $\mathcal{B}(X) = X^{1/d_2}$ for all $X \in \mathbb{Z}_N^*$. Algorithm $\mathcal{B}$ uses $\mathcal{A}$ as a subroutine.

c. Show using part (b) that user u cannot obtain the key key_i for any $i \notin S$ assuming that computing e'th roots modulo N is hard for any e such that $\gcd(e, \varphi(N)) = 1$. (the contra-positive of this statement should follow from (b) directly).

Problem 5 In class we briefly noted that a one-time signature scheme can be converted into a many-time signature scheme. Let's explore how to do it. The signer in our many-time scheme will maintain internal state and update it every time he signs a message. Let (G, S, V) be a one-time signature scheme (i.e. a scheme secure as long as a public/secret pair is used to sign at most one message). To build a signature scheme for signing 2^n messages (say $n = 32$) visualize a complete binary tree with 2^n leaves. Every node in this tree stores a different public/secret key pair for the one-time system. The public key for our many-time scheme is the public key stored at the root of the tree. To sign message number i the signer uses the ith leaf in the tree (for $1 \leq i \leq 2^n$). Let $u_0, \ldots, u_n$ be the n nodes on the path from the root to the ith leaf (u_0 is the root of the tree, u_n is the leaf). To sign the message m, first use the secret key in the leaf node u_n to sign m. Let s_n be the resulting signature. Then for $i = 0, \ldots, n-1$ use the secret key in node u_i to sign the pair of public keys stored in its two children. Let $(s_0, \ldots, s_{n-1})$ be the resulting n one-time signatures. For $1 \leq i \leq n$ let pk_i be the public key stored is node u_i and let pk'_i be the public key stored in the sibling of node u_i. The many-time scheme outputs $(i, (s_0, pk_1, pk'_1), \ldots, (s_{n-1}, pk_n, pk'_n), s_n)$ as the signature on m.

a. Write (short) pseudo-code to implement the signing and verification algorithms for the many-time scheme. Your signing code should maintain state containing at most $2n$ one-time public/private key pairs at any given time. Your verification code should be stateless.

b. Briefly explain why your implementation is secure. In other words, explain why your signing code never uses a one-time public-key to sign two distinct messages.

c. What is the size of the resulting signatures when using the Lamport one-time signature scheme discussed in class? How many applications of the one-way function are needed (on average) to generate a signature?