CS255: Cryptography and Computer Security **Winter 1998**

Assignment #3

Due: Friday, February 27th, 1998.

1. Let's explore some of the connections between our security notions for hash functions.

   (a) Assuming that factoring is hard, exhibit a hash function that is preimage resistant but not second preimage resistant.

   (b) Give an example of a hash function that is second preimage resistant but not collision resistant.
   Hint: think of a function that always outputs zero on some tiny fraction of the input space and computes SHA-1(x) everywhere else.

   (c) Show that if a hash function $h : \{0,1\}^{2k} \rightarrow \{0,1\}^k$ is collision resistant then it is preimage resistant.

   N.B.: In our security definitions we must specify the distribution of the "hard" problem instances. For preimage resistance (given y, it's hard to find x such that $h(x) = y$), y is chosen by choosing z uniformly from h's domain and computing $y = h(z)$. By "hard" we mean that a computationally bounded entity will find a preimage with negligible probability, given this distribution on y. Similarly, for second preimage resistance (Given x, find x' such that $h(x) = h(x')$), x is chosen uniformly from h's domain. There are other definitional subtleties, particularly for collision resistance, but they need not be considered to solve the above problems.

2. In this problem we explore the different ways of constructing a MAC out of a non-keyed hash function. Let $h : \{0,1\}^* \rightarrow \{0,1\}^b$ be a hash function constructed by iterating a collision resistant compression function using the Merkle-Damgård construction.

   (a) Show that defining $MAC_k(M) = h(k \parallel M)$ results in an insecure MAC. That is, show that given a valid text/MAC pair (M, H) one can efficiently construct another valid text/MAC pair (M', H') without knowing the key k.

   (b) Recall that in the Merkle-Damgård iterated construction one uses a fixed Initial Value IV as the initial chaining variable. Show that setting the IV to be the secret key k results in an insecure MAC.

   (c) Consider the MAC defined by $MAC_k(M) = h(M \parallel k)$. Show that in expected time $O(2^{b/2})$ it is possible to construct two messages M and M' such that given $MAC_k(M)$ it is possible to construct $MAC_k(M')$ without knowing the key k.

   (d) Give a short high level argument to show why the envelope method for constructing a MAC out of a hash function produces a secure MAC.

3. Recall that in the ElGamal signature scheme, a signature is of the form (a, b) where $b \in Z_q$ and a is an integer. In lecture, we glossed over the fact that a is required to be less than p. Show that without this restriction, one can forge signatures for any message. [Hint: ElGamal says to find (a, b) such that $y^a a^b = g^M$. First show how to find (b, c, d) such that $y^d c^b = g^M$.]

4. Signature schemes often include hashing of the input message as a first step. Typically, the hashed message is much shorter than the length of the input to the signature mechanism. For instance, the hashed message could be 160 bits long, while the signature mechanism takes a 1024 bit message. The question is how to convert the 160 bits hash value $H = h(M)$ to the 1024 bits input to the signature mechanism. There are two alternatives:

   - Append a *fixed* 864 bit string to the hashed message to make it into a 1024 bit string. Then feed the result into the signature mechanism.
   - Append a *random* 864 bits string to the hashed message to make it into a 1024 bits string. Then feed the result into the signature mechanism.

   Which of these alternatives results in the more secure signature mechanism? Justify your answer based on the amount of work the adversary has to do in order to generate a forged (message,signature) pair.

   We note that recently Bellare and Rogoway suggested a third approach for expanding the 160 bit hash to a 1024 bit value. Under certain assumptions, their construction can be shown to be secure against signature forgery.