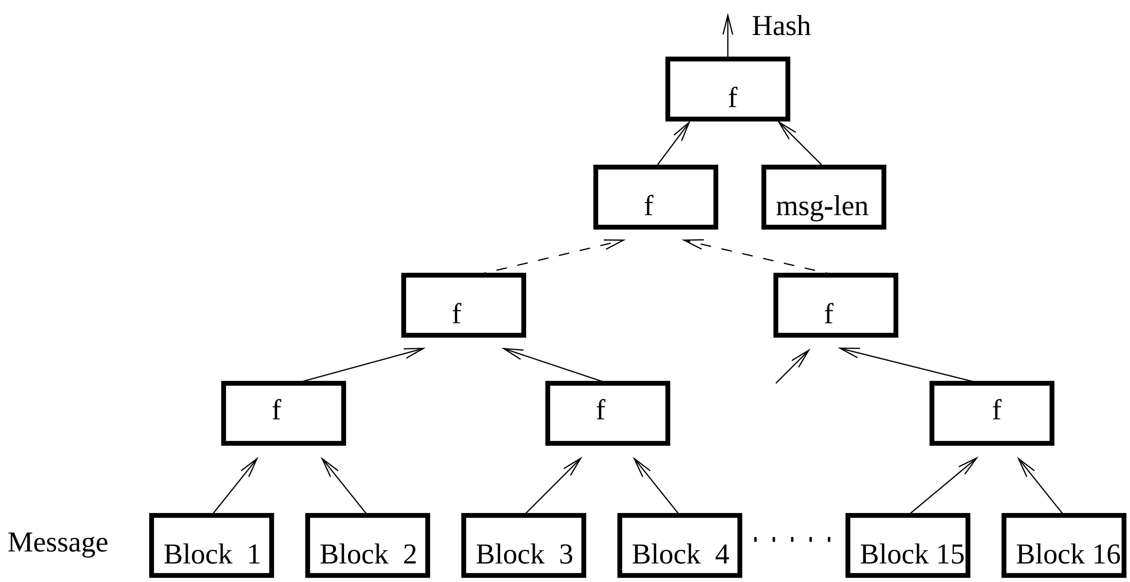
CS255: Cryptography and Computer Security **Winter 2005**

Assignment #2

Due: Wednesday, February 16th, 2005.

Problem 1 Merkle hash trees.

Merkle suggested a parallelizable method for constructing hash functions out of compression functions. Let f be a compression function that takes two 512 bit blocks and outputs one 512 bit block. To hash a message M one uses the following tree construction:



Prove that if one can find a collision for the resulting hash function then one can find collisions for the compression function.

Problem 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.

1. Show that defining $MAC_k(M) = h(k \parallel M)$ results in an insecure MAC. That is, show that given a valid msg/MAC pair (M, H) one can efficiently construct another valid msg/MAC pair (M', H') without knowing the key k.
2. 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.

Problem 3 Suppose Alice and Bob share a secret key k. A simple proposal for a MAC algorithm is as follows: given a message M do: (1) compute 128 different parity bits of M (i.e. compute the parity of 128 different subsets of the bits of M), and (2) AES encrypt the resulting 128-bit checksum using k. Naively, one could argue that this MAC is existentially unforgeable: without knowing k an attacker cannot create a valid message-MAC pair. Show that this proposal is flawed. Note that the algorithm for computing the 128-bit checksums is public, i.e. the only secret unknown to the attacker is the key k. Hint: show that an attacker can carry out an existential forgery given one valid message/MAC pair (where the message is a kilobyte long).

Problem 4 Let $x_1, \ldots, x_n$ be randomly sampled integers in the range $[1, B]$. The birthday paradox says that when $n = \lfloor 1.2\sqrt{B} \rfloor$ the probability that there is a collision (i.e. exists $i \neq j$ such that $x_i = x_j$) is a constant (greater than 1/2). How many samples $x_1, \ldots, x_n$ do we need until the probability that we get k collisions (i.e. exist $i_1, j_1, \ldots, i_k, j_k$ such that $x_{i_1} = x_{j_1}, \ldots, x_{i_k} = x_{j_k}$) is some non-zero constant? Justify your answer.
Hint: define the indicator random variable $I_{j,u}$ to be 1 if $x_j = x_u$ and zero otherwise. Then the expected number of collisions is $\sum_{j,u=1}^{n} E[I_{j,u}]$. When is this expectation greater than k?

Problem 5 Suppose user A is broadcasting packets to n recipients $B_1, \ldots, B_n$. Privacy is not important but integrity is. In other words, each of $B_1, \ldots, B_n$ should be assured that the packets he is receiving were sent by A. User A decides to use a MAC.

a. Suppose user A and $B_1, \ldots, B_n$ all share a secret key k. User A MAC's every packet she sends using k. Each user B_i can then verify the MAC. Using at most two sentences explain why this scheme is insecure, namely, show that user B_1 is not assured that packets he is receiving are from A.

b. Suppose user A has a set $S = \{k_1, \ldots, k_m\}$ of m secret keys. Each user B_i has some subset $S_i \subseteq S$ of the keys. When A transmits a packet she appends m MAC's to it by MACing the packet with each of her m keys. When user B_i receives a packet he accepts it as valid only if all MAC's corresponding to keys in S_i are valid. What property should the sets $S_1, \ldots, S_n$ satisfy so that the attack from part (a) does not apply? We are assuming all users $B_1, \ldots, B_n$ are sufficiently far apart so that they cannot collude.

c. Show that when $n = 6$ (i.e. six recipients) the broadcaster A need only append 4 MAC's to every packet to satisfy the condition of part (b). Describe the sets $S_1, \ldots, S_6 \subseteq \{k_1, \ldots, k_4\}$ you would use.

Problem 6 In this problem, we see why it is a really bad idea to choose a prime $p = 2^k + 1$ for discrete-log based protocols: the discrete logarithm can be efficiently computed for such p. Let g be a generator of $\mathbb{Z}_p^*$.

a. Show how one can compute the least significant bit of the discrete log. That is, given $y = g^x$ (with x unknown), show how to determine whether x is even or odd by computing $y^{(p-1)/2} \bmod p$.

b. If x is even, show how to compute the 2nd least significant bit of x.
Hint: consider $y^{(p-1)/4} \bmod p$.

c. Generalize part (b) and show how to compute all of x.

d. Briefly explain why your algorithm does not work for a random prime p.