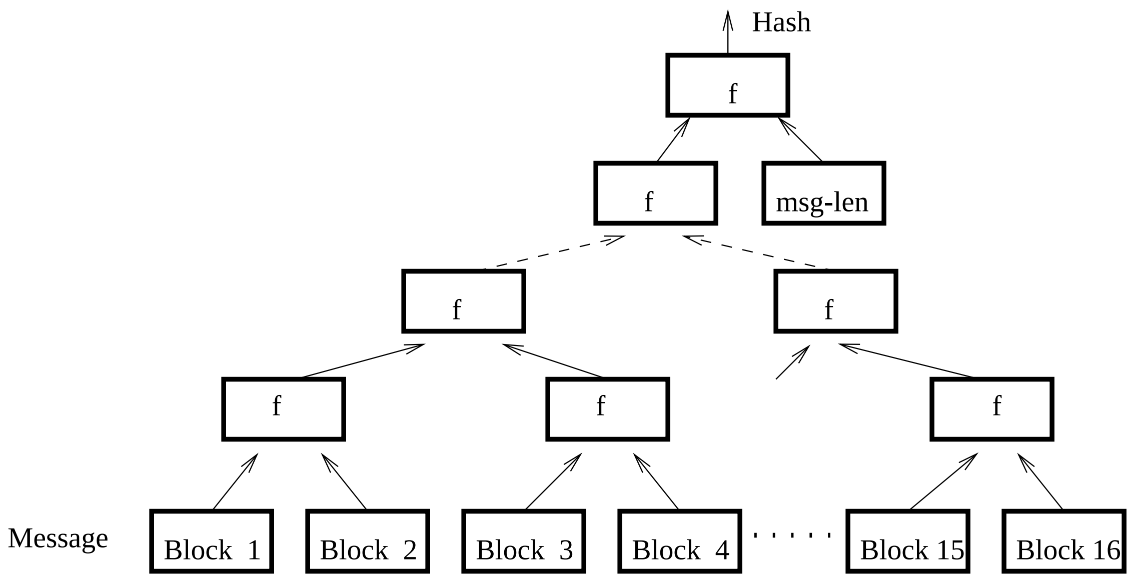
CS255: Cryptography and Computer Security — Winter 2001

Assignment #2

Due: Friday, February 22nd, 2002.

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 $h : \{0,1\}^* \to \{0,1\}^b$ 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 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 without knowing k an attacker cannot compute the MAC of a message M. Show that this proposal is flawed. Note that the algorithm for computing the 128-bit checksums is public.
Hint: show that an attacker can carry out an existential forgery given one valid message/MAC pair.

Problem 4 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.

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.

The fact that $p = 2^k + 1$ is inappropriate for crypto is unfortunate since arithmetic modulo such p can be done very fast.