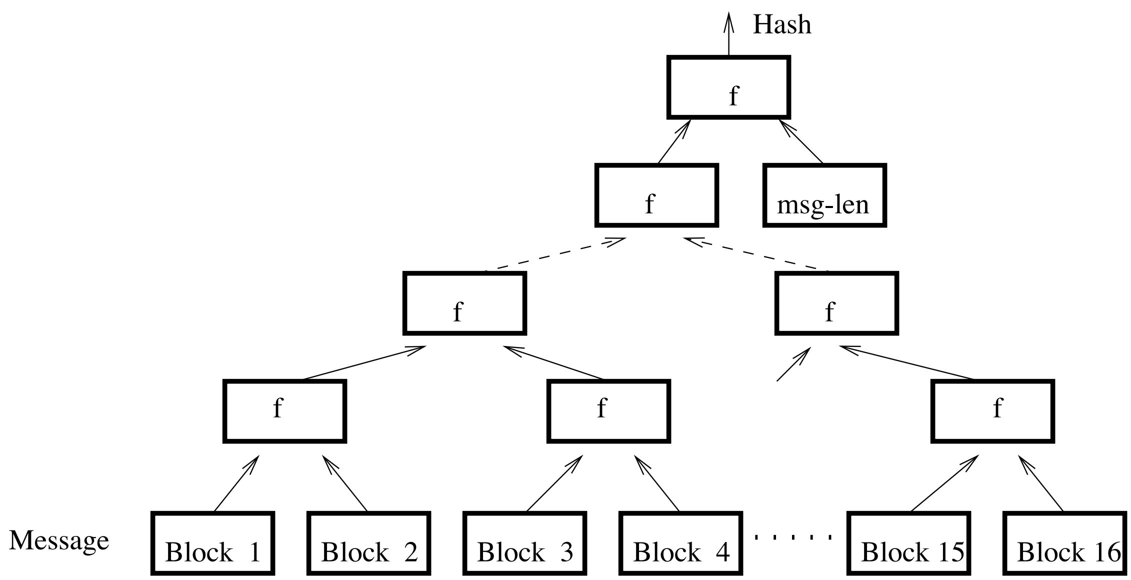
CS255: Cryptography and Computer Security	Winter 2011

Assignment #2

Due: Wednesday, Feb. 16, 2011.

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 the lecture we saw that Davies-Meyer is often used to convert an ideal block cipher into a collision resistant compression function. Let $E(k, m)$ be a block cipher where the message space is the same as the key space (e.g. 128-bit AES). Show that the following methods do not work:

$$f_1(x, y) = E(y, x) \oplus y \quad \text{and} \quad f_2(x, y) = E(x, x) \oplus y$$

That is, show an efficient algorithm for constructing collisions for f_1 and f_2. Recall that the block cipher E and the corresponding decryption algorithm D are both known to you.

Problem 3. Suppose user A is broadcasting packets to n recipients $B_1, \dots, B_n$. Privacy is not important but integrity is. In other words, each of $B_1, \dots, B_n$ 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 $B_1, \dots, B_n$ all share a secret key k. User A MACs 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 $S = \{k_1, \dots, k_m\}$ 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 MACs 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 $S_1, \dots, S_n$ satisfy so that the attack from part (a) does not apply? We are assuming all users $B_1, \dots, B_n$ are sufficiently far apart so that they cannot collude.

c. Show that when $n = 10$ (i.e. ten recipients) the broadcaster A need only append 5 MAC's to every packet to satisfy the condition of part (b). Describe the sets $S_1, \dots, S_{10} \subseteq \{k_1, \dots, k_5\}$ you would use.

Problem 4. Conference key setup.

Parties $A_1, \dots, A_n$ and B wish to generate a secret conference key. All parties should know the conference key, but an eavesdropper should not be able to obtain any information about the key. They decide to use the following variant of Diffie-Hellman: there is a public prime p and a public element $g \in \mathbb{Z}_p^*$ of order q for some large prime q dividing $p-1$. User B picks a secret random $b \in [1, q-1]$ and computes $y = g^b \in \mathbb{Z}_p^*$. Each party A_i picks a secret random $a_i \in [1, q-1]$ and computes $x_i = g^{a_i} \in \mathbb{Z}_p^*$. User A_i sends x_i to B. User B responds to party i by sending $z_i = x_i^b \in \mathbb{Z}_p^*$.

a. Show that party i given z_i (and a_i) can determine y.

b. Explain why (a hash of) y can be securely used as the conference key. Namely, explain why at the end of the protocol all parties $A_1, \dots, A_n$ and B know y and give a brief informal explanation why an eavesdropper cannot determine y.

c. Prove part (b). Namely, show that if there exists an efficient algorithm $\mathcal{A}$ that given the public values in the above protocol, outputs y, then there also exists an efficient algorithm $\mathcal{B}$ that breaks the Computational Diffie-Hellman assumption in the subgroup of $\mathbb{Z}_p^*$ generated by g. Use algorithm $\mathcal{A}$ as a subroutine in your algorithm $\mathcal{B}$. Note that algorithm $\mathcal{A}$ takes as input a triple (g, g^x, g^y) and outputs $g^{x/y}$ while algorithm $\mathcal{B}$ takes as input a triple (g, g^x, g^y) and outputs g^{xy}

Problem 5. Computing on ciphertexts. Let G be a group of prime order q with generator g.

a. Consider a variant of ElGamal encryption where the encryption of a message $m \in \mathbb{Z}_q$ using public key (G, g, h) is defined as $c \leftarrow (g^r, g^m h^r)$ where $r \xleftarrow{R} \mathbb{Z}_q$. Suppose $1 \le m \le B$. Write pseudo-code to decrypt the ciphertext c (i.e. recover the message m) using the secret key $x := \mathrm{Dlog}_g(h)$ with one exponentiation and $O(B)$ additional group operations.

b. Let c_1, c_2 be encryptions of message m_1, m_2 respectively. Show that there is a simple algorithm $\mathcal{A}$ that takes the public key (G, g, h) and the two ciphertexts c_1 and c_2 as input, and outputs a random encryption of $m_1 + m_2$. The output ciphertext should be distributed as if the message $m_1 + m_2$ was encrypted with fresh randomness. Note that $\mathcal{A}$ does not know either m_1 or m_2.

c. Suppose n people wish to compute the average of their salaries. Let x_i be the salary of person number i, where x_i is an integer in $[1, B]$ for all i. Our goal is to compute $A := (x_1 + \ldots + x_n)/n$ without revealing any other information about individual salaries. Note that A need not be an integer.

Design an n step protocol where in step i (for $i = 1, \ldots, n-1$) user number i sends a message to user number $i+1$. In step n user number n sends a message to user 1. User 1 then publishes A for all n people to see.

You may assume user 1 does not collude with any other user. All user 1 sees is the message he sends to user 2 and the message he receives from user n. Some remaining users may share information with one another to try and learn more information about individual salaries (information beyond what is revealed by A). **Hint:** User 1 generates a public/private ElGamal key. The remaining users use your mechanism from part (b).

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.
Hint: let $b \in \{0, 1\}$ be the LSB of x obtained using part (a). Try setting $y_1 \leftarrow y/g^b$ and observe that y_1 is an even power of g. Then use part (b) to deduce the second least significant bit of x. Show how to iterate this procedure to recover all of x.

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