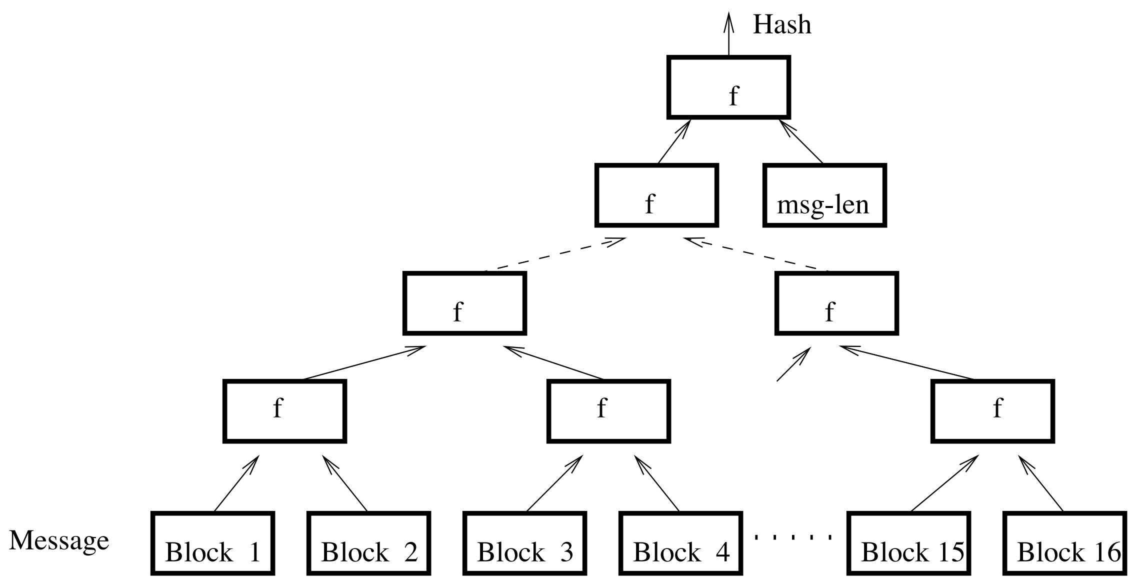
CS255: Cryptography and Computer Security Winter 2012

Assignment #2

Due: 5pm Friday, Feb. 17, 2012.

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:



For similicity, let's assume that the number of blocks in m is always a power of 2.

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

b. Show that if the msg-len block is eliminated (e.g. the contents of that block is always set to 0) then the construction is not collision resistant.

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:

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

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 we implement the CBC-MAC using a PRP (E, D). Show that for any key k the function $H(m) := \text{rawCBC}_E(k, m)$ is not collision resistant. That is, for an arbitrary key k show how to construct distinct m and m' such that $\text{rawCBC}_E(k, m) = \text{rawCBC}_E(k, m')$. Note that here k is public.

Problem 4. 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 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 $S = \{k_1, \ldots, k_m\}$ 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 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 $S_1, \ldots, S_n$ 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 $n = 10$ (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 $S_1, \ldots, S_{10} \subseteq \{k_1, \ldots, k_5\}$ you would use.

Problem 5. CBC padding attack. Recall that when using CBC mode, TLS pads messages to a multiple of the block length by appending a t byte pad for a suitable value of t and all bytes of the pad are set to $t - 1$. For example, if a 2 byte padded is needed, TLS appends $(1, 1)$ to the plaintext prior to CBC encryption. The recipient, after decrypting the CBC chain, checks that the pad has the correct format and if not rejects the ciphertext. A bug in older versions of OpenSSL let the attacker learn if ciphertext rejection happened due to a bad pad.

Now, suppose an attacker intercepts a target ciphertext c_{full}. The attacker deletes the last block of c_{full} thereby deleting any padding blocks. Let c be the resulting truncated ciphertext and let m be the result of decrypting this c using CBC decryption. Your goal is to show that this OpenSSL bug can let the attacker test if the last of byte of m is equal to some byte g of the attacker's choosing. Using c, construct a ciphertext c' that has the following property: when c' is sent to the server, the decryption of c' will end with a valid pad if the last byte of m is equal to g and will end with an invalid pad (w.h.p) otherwise. By sending c' to the server, the attacker can therefore learn if m ends with g.

note: In principle, the attacker can repeat this experiment for all 256 values of g until a match is found. He then learns the last byte of m. However, TLS tears down the connection and renegotiates a new key when a pad error occurs and therefore this typically cannot be applied to TLS. Nevertheless, when using IMAP over TLS, the IMAP server repeatedly sends the user's password to the IMAP server giving the attacker a perfect opportunity to mount a repeated attack and learn the user's password one byte at a time.

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.

Problem 7. Conference key setup.
Parties $A_1, \ldots, A_n$ 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 $A_1, \ldots, A_n$ 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}