CAS CS 538. Problem Set 6

Due in class Tuesday, October 23, 2012, *before* the start of lecture

Problem 1. (40 points) In this problem you will study whether two one-way functions can be composed to produce a new one-way function.

(a) (10 points) Suppose $f(x)$ is a one-way function. Let $g_1(x)$ be the function that outputs $f(x)$ followed by $|f(x)|$ zeroes. Prove by a reduction that g_1 is one way.

(b) (20 points) Suppose $f(x)$ is a one-way function. Let $g_2(x)$ be the following function: if the last $\lceil |x|/2 \rceil$ bits of x are 0, then output $f(0)$. Else output $f(x)$. Show that g_2 is a one-way function.

(Hint: Suppose an inverter A for g_2 exists. Let p_x be the probability that this inverter is correct on input x. We know that the proability of successful inversion over all inputs of size k is equal to

$$\sum_{x \in \{0,1\}^k} p_x / 2^k$$

and is not negligible in k. Construct an invereter B for f from A. Let q_x be the probability that B is correct on input x. The probability of B's success on inputs of length k is

$$\sum_{x \in \{0,1\}^k} q_x / 2^k.$$

Express q_x in terms of p_x, and show why the probability of B's success is not negligible.)

(c) (10 points) Suppose $g_1(x)$ and $g_2(x)$ are one-way functions. Is $g_3(x) = g_2(g_1(x))$ necessarily one-way? Prove your answer. (Hint: use previous two parts.)

Problem 2. (30 points) Show how to compute the root of an n-leaf Merkle tree in $\log n$ space. More precisely, given $x_1, x_2, \ldots, x_n$ as values to be placed in the leaves of the tree, and a hash function H, describe how to compute the root r of the tree while never storing more than $\lceil \log n \rceil$ hash values.

For the next problem, we will augment collsion-resistant hashing with a useful property. Consider a collision-resistant hash family whose domain D_i can always be written as $M_i \times R_i$ (for example, for the DL-based family studied in class, $i = (p, g, h)$ and $D_{p,g,h} = M_{p,g,h} \times R_{p,g,h}$, where $M_{p,g,h} = R_{p,g,h} = \{1, 2, \ldots, q\}$). We will call this family a *trapdoor hash family* if it has two additional properties: the algorithm Gen also outputs a *trapdoor key* t_i in addition to the hash function index i, and there exists an algorithm T that, on input $(1^k, i, t_i, m_1, r_1, m_2)$, will output r_2 such that $H_i(m_1, r_1) = H_i(m_2, r_2)$ (here, $m_1, m_2 \in M_i$ and $r_1, r_2 \in R_i$) in polynomial time. In other words, although collisions are hard to find given i, they are very easy to find given given extra information t_i. In fact, collision are so easy to find given t_i that, given one input and half of another, you can find the remaining half of the second input so that the two inputs collide.

Below you will show that the hash function family studied in class has a trapdoor. This means that you may have to trust the person who picked the hash function to not know the trapdoor—because the function is not collision-resistant to anyone who knows the trapdoor.

However, trapdoor hash families can be quite useful. For example, if the hash function is chosen by the signer at the same time as she generates her public-secret key pair for the signature scheme (in which case the signer puts i in her public key), then we don't mind that she knows the trapdoor, since it is in her interests not to reveal it—else, others would be able to find collisions and, therefore, forge signatures on her behalf. Moreover, this enables the signer to perform much of the signature computaiton ahead of time, before she even knows what message she will be signing, in the following interesting twist of the hash-and-sign paradigm (due to Adi Shamir and Yael Tauman).

Take any signature scheme and modify it as follows. Add the hash key i to your public key, and the trapdoor t_i to your secret key. Before you even know what message you are signing, take a random message m' and value r', and sign the hash $h = H_i(m', r')$ to get σ'. Then, when the time comes to sign some message m, simply run T to find r such that the hash of (m, r) is h, and output (σ', r) as your signature. Thus, you can precompute most of the signature before you even know the message, and then do only the very quick computation of T once the message is known. This is useful, e.g., when a server has idle cycles to burn some times, and is overloaded at other times.

Problem 3. (30 points) Show that the DL-based family studied in class is actually a trapdoor hash family. In other words, demonstrate how to modify Gen, what $t_{p,g,h}$ will be, and construct T that uses $t_{p,g,h}$. (Hint: Gen should no longer be simply selecting g, h blindly at random.)