

CAS CS 538. Problem Set 8

Due 5pm Thursday, December 2, 2010, by email to reyzin@bu.edu or in my mailbox in room MCS 138.

Problem 1. (20 points) Show how to compute the root of an n -leaf Merkle tree in $\log n$ space. More precisely, given x_1, x_2, \dots, 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.

Here is some algebra review that may be useful in the next two problems. If G is a group, then *order of G* is simply another way of saying “the number of elements in G ,” or “ $|G|$.” If $g \in G$, the *order of g* is the smallest integer $r > 0$ such that $g^r = 1$ in G . It is a theorem from algebra that the order of g always divides the order of G . If the order of g is equal to the order of G , then one can show easily that the powers of g cover the entire group G , and hence g is a generator of G . If a group has a generator, it is called *cyclic*.

Note that if G has prime order q , and $g \in G$ has order r , the $r = 1$ or $r = q$ (because $r|q$ and q is prime). If $r = 1$, then $g = g^1 = g^r = 1$. Else, the order of g is the same as the order of G , and so g is a generator of G . Thus, in a group of prime order every element but the element 1 is a generator, so it is always cyclic. In particular, if $p = 2q + 1$, and p, q are primes, then $|QR_p| = (p - 1)/2 = q$ is prime, and hence QR_p is cyclic, with every element but 1 as its generator.

Now, we will augment collision-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, $D_{p,g,h} = M_{p,g,h} \times R_{p,g,h}$, where $M_{p,g,h} = R_{p,g,h} = \{1, 2, \dots, q\}$). 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 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, given extra information t_i , collisions are easy to find in a very strong sense (given one input and half of another, you can find the remaining half).

Below you will show that some hash families have trapdoors. 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 it.

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 computation 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 2. (20 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.)

Problem 3. (60 points) Let $p_1 = 2q_1 + 1$ and $p_2 = 2q_2 + 1$ be two distinct safe primes, and $n = p_1p_2$ be of length k bits.

(a) (10 points) Show by CRT that QR_n is cyclic (the easiest way to do this is to try to construct a generator of QR_n and show that it is indeed a generator by showing that its order is q_1q_2).

(b) (10 points) Show how to efficiently factor n given q_1q_2 . In fact, you can factor n given any multiple of q_1q_2 , but we won't show it here.

(c) (20 points) Let g be a generator of QR_n , and let $M_{n,g} = R_{n,g} = \{1, 2, \dots, q_1q_2\}$. Let $H_{n,g}(m, r) = g^{m2^k+r} \bmod n$ (note that $m2^k+r$ is just concatenation of m and r as bit strings). Show that this is a collision-resistant hash family under the assumption that such n are hard to factor. Specifically, show how, given (m_1, r_1) and (m_2, r_2) that collide, one can factor n . You may use without proof the fact that n can be efficiently factored given any non-zero multiple of q_1q_2 .

(d) (20 points) Show also that this is a trapdoor hash family (again, find what trapdoor information Gen should output and what the algorithm T will do). Analyze the running time of T to notice how it makes the process described in the paragraph above problem 2 very efficient.