COS 433: Cryptography | Princeton University
Homework 9 | Due: May 7, 2017, 11:59pm

# Homework 9

## 1 Problem 1 (50 points)

Recall the S/key system discussed in class. The secret key is $\mathsf{sk}$, a random input, and the verification key is $\mathsf{vk} = y_0 := H^n(\mathsf{sk})$, where $H^n$ denotes iterating the hash function $n$ times. At the beginning of the $i$th round of identification, Bob is storing the value $y_{i-1} := H^{n-(i-1)}(\mathsf{sk})$. Alice sends the message $y_i$ to Bob. Bob verifies that $H(y_i) = y_{i-1}$. Then Bob updates his state to $y_i$.

One problem with this scheme is that Alice's running time is on average $O(n)$ per identification, since she needs to compute $y_i$ from $\mathsf{sk}$. One possibility is to pre-compute all $y_i$ values at the beginning, and store them all. However, this now requires $O(n)$ space for Alice. Alice can also interpolate between these two by storing only every $T$th hash; to compute the next message, she will at maximum need to compute $T$ hashes. In any of these settings, we have $ST \geq n$, where $S$ is the storage requirement, and $T$ is the number of hashes needed in each identification. Since $n$ bounds the number of one-time passwords, typically $n$ is quite large (e.g. $2^{20}$), so this time-memory trade-off is undesirable.

**Show how Alice can maintain a state consisting of $O(\log n)$ values, and only require a total of $O(n \log n)$ hashes for all $n$ identification rounds. This gives an amortized cost of $O(\log n)$ hashes per iteration.**

Hint: The problem naturally corresponds to a certain pebbling game. There are $n$ positions, numbers 1 through $n$, corresponding to the $n$ messages Alice will send, $y_1, ..., y_n$ (where $y_n = \mathsf{sk}$). Some $k$ positions have pebbles, corresponding to the hashes stored by Alice. At the beginning of round $i$, the $i$th position should have a pebble on it (so Alice can send $y_i$ to Bob). The pebble at $i$ is removed (since Alice can forget $y_i$ afterward). Then, you can make a sequence of pebble moves. Given a pebble at position $j$, one possible move is to place a pebble at position $j-1$ (corresponding to computing $y_{j-1} = H(y_j)$). You can also remove any pebble arbitrarily (by forgetting a hash); removal does not count as a move, only placing. The restriction is that the total number of pebbles in play never exceeds $k$, and you want to minimize the number of pebbles moves during each round. At the end round $i$, you should be ready for the next iteration, meaning you have a pebble at $i+1$.

Suppose $n = 2^k$. Suppose your maximum number of pebbles is $k+1$. In round $i$, after removing the pebble at $i$, let $j$ be the lowest pebble above $i$. You will start from $j$, and place pebbles at $j-1, j-2, ...$ until you reach $i+1$. You may not have enough pebbles to leave pebbles at each of $j-1, j-2, ...$, so instead, you will remove most of the pebbles once you've placed the next pebble. However, you will strategically leave some pebbles behind to make your life easier on future iterations. The number of moves you will need in this round will be $j-(i+1)$.

**What pebbling strategy ensures that no more than $k+1$ pebbles are ever in play, and that the amortized number of steps is $O(n \log n)$?**

As a further hint, it is possible to set things up so that roughly half the rounds will require no moves, a quarter will require one move, an eighth will require three, a sixteenth will require seven, etc (so roughly $1/2^{\ell}$ fraction will require $2^{\ell-1}-1$ moves, for $i = 1, ..., k$). Summing up all the moves gives the desired $O(n \log n)$.

## 2 Bonus: Problem 2 (10 points)

Show how to modify the pebbling strategy above so that it takes *worst case* $O(\log n)$ number of pebble moves per iteration (As opposed to amortized), while preserving the total storage of $O(\log n)$ pebbles.

## 3 Problem 3 (25 Points)

Let $p$ a prime, and $\mathbb{G}$ a group of order $p$. In class (lecture 15) we saw a random self reduction for discrete log: given a pair $(g, h = g^a)$, you can come up with a new pair $(g, h' = g^{a'})$ such that (1) $a'$ is random and independent of $a$, and (2) given the discrete log of the new pair, $a'$, you can find the discrete log of the old pair, $a$. To do so, let $h' = h^r$ for a random $r$. Then $a' = ar$, so it is random and independent of $a$. Also, given $a'$, you can recover $a$ by dividing by $r \pmod p$.

Show such a random self reduction for DDH. That is, you are given a tuple $(g, u = g^a, v = g^b, w = g^c)$ where $a$ and $b$ are random (in $\mathbb{Z}_p$), and $c$ is either $ab \bmod p$ or also random. In the random $c$ case, we call this a random tuple. In the $c = ab \bmod p$ case, we call this a DDH tuple. Show how to devise a new tuple $(g, u', v', w')$ such that

- If $(g, u, v, w)$ is a DDH tuple, then so is $(g, u', v', w')$. Moreover, the components of $(g, u', v', w')$ are independent of $(g, u, v, w)$. For example, even if $u = g^2, v = g^3, w = g^6$ (so there is no entropy and the discrete logs are fixed), $(g, u', v', w')$ should be a random DDH tuple.
- If $(g, u, v, w)$ is *not* a DDH tuple, so that $c \neq ab \mod p$, then $(g, u', v', w')$ is a random tuple. Moreover, the components of $(g, u', v', w')$ are independent of $(g, u, v, w)$. For example, even if $u = g^2, v = g^3, w = g^5$, $(g, u', v', w')$ should be a (at least statistically close to a) uniformly random tuple.

The transformation from $(g, u, v, w)$ to $(g, u', v', w')$ must be efficient: you cannot compute discrete logs as part of the transformation.

Note that the following simple transformations will not work:

- $(g, u^r, v, w^r)$. This is a DDH tuple if $(g, u, v, w)$ was a DDH tuple, and isn't a DDH tuple if $(g, u, v, w)$ isn't. However, it is not independent of the original tuple. For example, the third component is identical.
- $(g, u^r, v^s, w^{rs})$. This is a DDH tuple if and only if $(g, u, v, w)$ is, but still isn't independent of the original tuple. For example, if $(g, u, v, w)$ is random, we have that $c/ab = c'/a'b' \mod q$. Therefore, there is a relationship between the new elements and the old elements. Instead, we expect if $(g, u, v, w)$ is random, that all of the terms in the two tuples $(g, u, v, w), (g, u', v', w')$ are random and independent (except of course the $g$ terms).

## 4 Problem 4 (25 points)

Here, you will show that computing discrete logs mod a composite integer $N = pq$ is as hard as factoring $N$. In other words, you are given an algorithm $A$ such that given $g, h \in \mathbb{Z}_N^*$, $A$ efficiently computes an integer $x$ such that $g^x \bmod N = h$. (Note that in general $\mathbb{Z}_N^*$ is not cyclic, so the discrete log is not guaranteed to exist. The algorithm for discrete logs is only guaranteed to work when the discrete log exists). Show that given $A$, you can factor $N$.

Hint: recall from your previous homeworks that recovering a multiple of $\phi(N)$ is sufficient to recover the factors of $N$. Explain how to use a discrete log algorithm to compute a multiple of $\phi(N)$.