Homework 8

1 Problem 1 (15 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 information that will allow you to recover $\phi(N)$. See if you can use the discrete log finder to generate such information.

2 Problem 2 (25 points)

Recall the S/key system discussed in class. The secret key is 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 ith 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 (note that we are slightly changing the notation in class, which had the original vk being labeled as y_n).

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 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 Tth 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^{30}), so this time-memory trade-off may be 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 ith 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 $O(n \log n)$.

3 Problem 3 (20 points)

Recall that a graph G is 3-colorable if each node of the graph can be painted one of three colors (say, red, green, and blue, which we will associate with the numbers 1,2,3) such that, for every edge in G, the endpoints of that edge are painted different colors. Notice that it is easy to verify that a 3-coloring is valid by just checking all of the edges and making sure the endpoints are different. Therefore, 3-coloring is in NP, where the witness for G being 3-colorable is just the 3-coloring.

Consider the following simple proof. Choose a random permutation σ on the colors $\{1, 2, 3\}$. Given a coloring C, let $\sigma(C)$ be the coloring obtained by applying σ to the color of each node. To prove that a graph is 3-colorable, using a witness C, the prover simply sends $\sigma(C)$ to the receiver. The verifier then checks that $\sigma(C)$ is a valid 3-coloring.

(a) This proof system fails in one of the three required properties for a zero knowledge proof system (completeness, soundness, or zero knowledge). Which one fails?

In order to get around the problem above, instead imagine the following physical scheme using locked boxes. The prover will start with one lockable box for each node in G; for each node, she will write down the color of that node in $\sigma(C)$ and put it in the corresponding box. The prover will lock all the boxes, keep the keys, but send the locked boxes to the verifier. Of course, now the verifier has no hope of checking that $\sigma(C)$ is a valid 3-coloring. Instead, the verifier chooses a random edge $e = (u, v) \in G$, and sends e to the prover. The prover then sends the keys for those two nodes back to the verifier. The verifier opens the two boxes, and checks that the colors inside are different

(b) Prove that, if G is not 3-colorable, that a malicious prover has a significant chance of being caught (though it will potentially be < 1). Thus this scheme has a weak form of soundness.

(c) Prove that the scheme has (malicious verifier) zero knowledge. Assume the simulator can build its own locked boxes

The soundness of this physical scheme can be boosted by repeating the scheme many times sequentially, using a new random σ for each iteration. Thus, we can obtain a physical ZK proof.

(d) What crypto object should we use to implement the locked boxes above, to get a purely digital scheme? Re-prove soundness and zero knowledge for this new protocol.

Recall that 3-coloring is an NP complete language. By composing the scheme above with NP reductions, we can obtain a zero knowledge proof for all of NP