Problem Set 2

This problem set is due on *Monday, March 7, 2016* at **11:59 PM**. Please note our late submission penalty policy in the course information handout. Please submit your problem set, in PDF format, on Gradescope. *Each problem should be in a separate PDF.* Have **one and only one group member** submit the finished problem writeups. Please title each PDF with the Kerberos of your group members as well as the problem set number and problem number (i.e. *kerberos1_kerberos2_kerberos3_pset1_problem1.pdf*).

You are to work on this problem set with your assigned group of three or four people. Please see the course website for a listing of groups for this problem set. If you have not been assigned a group, please email `6.857-tas@mit.edu`. Be sure that all group members can explain the solutions. See Handout 1 (*Course Information*) for our policy on collaboration.

Homework must be submitted electronically! Each problem answer must be provided as a separate pdf. Mark the top of each page with your group member names, the course number (6.857), the problem set number and question, and the date. We have provided templates for LaTeX and Microsoft Word on the course website (see the *Resources* page).

Grading: All problems are worth 10 points.

With the authors' permission, we may distribute our favorite solution to each problem as the "official" solution—this is your chance to become famous! If you do not wish for your homework to be used as an official solution, or if you wish that it only be used anonymously, please note this in your profile on your homework submission.

Our department is collecting statistics on how much time students are spending on psets, etc. For each problem, please give your estimate of the number of person-hours your team spent on that problem.

Problem 1-1. Floyd's Two Finger Algorithm

Ben Bitdiddle is given a random oracle $H : \mathbb{Z}_N \to \mathbb{Z}_N$. H responds to every unique query with (truly) random response chosen uniformly from $\mathbb{Z}_N$. If a query is repeated, H responds the same way every time that query is submitted. In this problem, the running time and space of a probabilistic algorithm refer to the expected running time and space over the random choices made by the oracle and the algorithm.

(a) Ben Bitdiddle wants to find a collision $x \neq x'$ such that $H(x) = H(x')$. A natural approach, which takes $O(N)$ time and $O(\sqrt{N})$ space, is to query H on $\{0, \ldots, O(\sqrt{N})\}$, then compare each pair of them to see if there is any collision.

Argue that this algorithm finds a collision with constant probability.

(The time complexity can be improved to $O(\sqrt{N})$ using a hash table, but that is not germane to this problem.)

(b) Ben Bitdiddle went to the recitation and learned Floyd's Two Finger Algorithm. The algorithm keeps two pointers p and q, and picks a random x from $\mathbb{Z}_N$ as the starting point.

At the first stage, p, q are initialized to x; in each step, $p = H(p)$ and $q = H(H(q))$; repeat until $p = q$.

At the second stage, p is set to x, and q remains at the meeting point; in each step, $p = H(p)$ and $q = H(q)$; repeat until $H(p) = H(q)$.

Finally, the collision is p and q.

Present an argument that Floyd's Two Finger Algorithm always terminates with $p \neq q$ such that $H(p) = H(q)$ if x lives in a tail of some cycle.

(c) Analyze the running time of Floyd's Two Finger Algorithm in terms of the length of the tail and the length of the cycle. Argue that the running time is $O(\sqrt{N})$.

(d) Instead of finding collisions, Ben Bitdiddle wants to find x, x' such that $H(x) + H(x') = 0 \pmod{N}$. Describe an algorithm that takes $O(\sqrt{N})$ time and $O(1)$ space.

Problem 1-2. 857coin

Rumor has it that a new cryptocurrency has sprung up at MIT. Lighter than litecoin, more wow than dogecoin, 857coin brings a memory-intensive three-collision proof of work to the table.

In 857coin, blocks operate just as in bitcoin, except that blocks now have 3 *unique* nonces: n_1, n_2, n_3. To compute a proof of work for a difficulty d, we compute the hash h_i of the block with respect to each individual nonce n_i for $i = 1, 2, 3$, and then check that $h_1 \equiv h_2 \equiv h_3 \bmod 2^d$.

Note: this problem is somewhat of an experiment for us, and we reserve the right to tweak it with reasonable warning as events unfold.

Also note: we will not be bringing up the server until sometime around 5pm on Tuesday, February 23rd.

(a) To get started, visit `http://6857coin.csail.mit.edu:8080/` and read the API for 857coin. Then, look at the provided miner.py template and make the required modifications to begin mining. You will receive full credit for part (a) after successfully mining a block that appends to any tree rooted at the genesis block. To receive credit for your team, include your team members' usernames separated by commas in the block contents.

(b) Now see where you can optimize your miner even further. The slower your miner is in comparison to other miners, the longer it will take to add to the main (longest) chain. Take a look at the three-collision algorithms paper posted at `https://courses.csail.mit.edu/6.857/2016/studentsOnly` for inspiration. You will receive full credit for part (b) if you ever append to the main chain. Remember to include your team in the block contents! Also note that the earlier you start, the slower your competition will be! Feel free to get creative by using different languages or hardware. Partial credit will be awarded for mining blocks of the following difficulties:

- 4/5pts for difficulty 38 or greater
- 3/5pts for difficulty 37
- 2/5pts for difficulty 36
- 1/5pts for difficulty 35

Please submit your code, along with a brief explanation of your strategy, to Gradescope.

Problem 1-3. Merkle Trees

Merkle Trees were covered in lecture as an application of hash functions to authenticate a collection of files. Merkle trees are binary (or more generally, k-ary) trees where each non-leaf node is the hash of the concatenation of its children, and each leaf node is the hash of one file from the collection.

It was stated that we can use Merkle Trees to authenticate any collection of files as long as the hash function we choose is collision resistant and the number of files in the collection is known.

(a) Prove that if the number of files that we are trying to authenticate is not known in advance, then Merkle trees are not secure for authentication purposes. That is, as a mapping from collections of files to hashes at the root, the mapping is not collision-resistant.

(b) Suppose we want to authenticate a variable number of files. Propose a simple modification that would allow Merkle trees to be used securely for authenticating such a collection of files. (Note that the number of files in the collection also needs to be authenticated.)

(c) Suppose we make the Merkle tree k-ary, for some integer $k \geq 2$. Suppose that the tree has a large number n of leaves, each representing one file. For what k do we minimize the length of the proof needed to prove that a file is a member of the collection authenticated by the (modified) Merkle tree? (In lecture, Professor Rivest said that the answer was likely to be $k = 2$ or $k = 3$. Is one of these right?)