Massachusetts Institute of Technology
6.857: Network and Computer Security
Professors Ronald L. Rivest and Yael Tauman Kalai

Handout 5
April 1, 2019
Due: April 22, 2019

Problem Set 4

This problem set is due on *Monday, April 22, 2019* 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.* When submitting the problem in Gradescope, ensure that **all your group members are listed on Gradescope**, and not in the PDF alone.

You are to work on this problem set (and problem set 5) in groups of your choosing of size three or four. If you need help finding a group, try posting on Piazza or email `6.857-tas@mit.edu`. You don't have to tell us your group members, just make sure you indicate them on Gradescope. 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 4-1. 6857Coin

Rumor has it that a new cryptocurrency has sprung up at MIT!

The new, experimental 6857coin is designed to use a verifiable delay function (VDF) as it's proof-of-work in order to discourage excessive parallelism in favor of more sequential work. A verifiable delay function is a function which is guaranteed to take some amount of time to compute, but cannot be efficiently parallelized, so ideally this would reduce incentive to set up large mining farms.

In a normal proof-of-work blockchain (e.g. Bitcoin), we usually try to compute some nonce so that

$$\text{Hash}(block||nonce)$$

is small; in particular, for difficulty d, the hash must be at most $\max(\text{Hash})/d$, so as to require d trial hashes on average.

In 6857coin, our verifiable delay function is of the form: take a hard-to-invert function f (like a hash function) and iterate it T times:

$$f(f(\dots f(block)\dots)) = f^T(block)$$

Then, we want $f^T(block)$ to be small, but T to be large. In particular, for a difficulty d, we require that $T > d$ and $f^T(block) < \max(f)/d$; this way, any miner must perform d computations of f initially to get a large enough T, and then d more computations on average to find a small enough f. Furthermore, this can't be parallelized, because each computation depends on the previous. Then, the startup cost of d trials should discourage people from buying additional hardware!

The "verifiable" aspect of this function is important so that everyone can effectively verify the miner's computed value of $f^T(block)$ is truthful; otherwise, they'd have to go through just as much computation to verify the value. In order to make it easy to verify, we have to choose specific number-theoretic functions. For 6857coin, we chose squaring modulo a semiprime.

The details of 6857coin and the particular VDF can be found at `http://6857coin.csail.mit.edu/`. It should be possible to complete this problem without having to dive deep into the VDF, though it may help you mine faster!

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

(a) Theory. In the first half of this problem, we'll ask the question: do VDF's work to discourage parallelism and encourage sequential computation? Are they still secure?

In a simplified model, assume that there are n miners (n is large) with identical hardware that can each compute our VDF function f or a blockchain's hash function.

1. In a normal blockchain at difficulty d, on average, how many hashes are computed in total by all miners per block successfully mined? How many hashes are computed by each miner in that time? What's each miner's probability of mining the block?
2. With our VDF at difficulty d, on average, how many times is f computed in total by all miners per block successfully mined? How many f computations are run by each miner in that time? How many actual "candidates" does each miner evaluate? What's each miner's probability of mining the block?
   Hint: Remember that we require $T > d$, so the first d evaluations aren't valid candidates for the proof-of-work.
3. Now, consider Melissa the Miner, who is trying to optimize her mining process. Melissa buys another identical computer, getting twice the parallel compute power. Now, on the traditional blockchain, how many hashes can Melissa evaluate in the time to mine a block (by anyone), and what is her new probability of mining the next block? On the VDF blockchain, how many candidates does she evaluate in the time to mine a block (by anyone), and what is her new probability of mining the next block on the VDF blockchain?
   (Note that n is very large, so you can make some reasonable approximations to simplify the calculations.)
4. Melissa hears that sequential computation is better, so instead of buying a second computer, she upgrades her computer to one that is twice as fast at hashing and computing f. Now, on the traditional blockchain, how many hashes can Melissa evaluate in the time to mine a block (by anyone), and what is her new probability of mining the next block? On the VDF blockchain, how many candidates does she evaluate in the time to mine a block (by anyone), and what is her new probability of mining the next block on the VDF blockchain?
5. Does the VDF help discourage paralellism? Does it reward sequential computation? Is it as secure as a traditional blockchain?

(b) Practice. Setting all that aside, let's mine some blocks!

1. To get started, visit `http://6857coin.csail.mit.edu/` and read the API for 6857coin. Then, look at the provided miner.py template and make the required modifications to begin mining. You will receive full credit for this part 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' kerberos usernames separated by commas in the block contents.
2. 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. You will receive full credit for this part if you ever append to the main chain and include in your writeup a description of your strategy for mining a block on the main chain. Remember to include your team's kerberos usernames 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.

Problem 4-2. Digital Signatures

Recall the El-Gamal signature scheme that we discussed in class.

For a generator g of a cyclic group G of prime order q, the signing key of the El Gamal signature scheme is $x \leftarrow \mathbb{Z}_q^*$ and the verification key is g^x. To sign a message m, we sample $k \leftarrow \mathbb{Z}_q^*$ and set $r = g^k$. We then set $e = h(m, r)$ for a hash function h that is modeled as a random oracle. The signature for m is the pair (r, s) where $s = k^{-1}\big(e + xr\big)$.

To verify a signature (r, s) for a message m and a verification key $y = g^x$, the verifier checks that $r^s = y^r \cdot g^{h(m,r)}$.

We will explore the security of this scheme for a weak hash function. Specifically, we will consider the hash function $h(x, y) = x$, which will result in $h(m, r) = m$ in all signatures.

(a) Argue that the El Gamal signature scheme is insecure with this identity hash function. Specifically, given an El Gamal verification key $y = g^x$, construct a signature (r, s) and a message m such that $r^s = y^r \cdot g^m$.

(b) Explain why using a random oracle $h(m, r)$ prevents the attack you gave in part (a).

Problem 4-3. Poor Randomness

Recall that in her guest lecture (Lecture 13), Nadia Heninger showed us that generating randomness in the real world is not an easy task. Many real-world implementations have insufficient randomness (potentially revealing some pattern or predictability to an adversary) when generating public and secret keys, which renders the underlying scheme to be insecure. In this problem we will explore some vulnerabilities exposed by poor randomness in schemes we have recently studied.

(a) Suppose RSA keys are generated, using true randomness, but $S = p + q$ is leaked to the adversary. Is the RSA scheme still secure? More specifically, given such a leakage, is the function $f_{n,e} : \mathbb{Z}_n \to \mathbb{Z}_n$ still one way where $f_{n,e}(x) = x^e \mod n$? (Note: this is the encryption function for RSA given in lecture.) Argue whether it is or is not.

(b) In the El-Gamal signature scheme and DSA, randomness is used in each and every signature generation. More specifically, we randomly select a per-message signature value k from $\mathbb{Z}_q^*$ (i.e. $1 < k < q$).

1. What happens if k is reused for different messages and an adversary has access to many signatures? Is security lost?
2. Now suppose that all but the last 15 bits are leaked, i.e. the first 145 bits of k ($|k| = 160$ bits) are leaked in each of several signatures (but we are using different k for each). Is our secret key x ($x \leftarrow \mathbb{Z}_q^*$ during the key generation process) secure?