Massachusetts Institute of Technology
6.857: Network and Computer Security
Professor Ron Rivest

Handout 3
February 24, 2012
Due: March 10, 2012

Problem Set 2

This problem set is due on *Monday, March 10* at **11:59 PM**. Please note that no late submissions will be accepted. Please submit your problem set, in PDF format, *by email* to `6.857-hw@mit.edu`. Each problem should be in a separate PDF. Have **one and only one group member** submit the finished problem set. 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 appear on a separate page. 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 will 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 the homework submission website.

Problem 2-1. Rainbow Tables

Ben Bitdiddle is learning to code, and he has decided to build a website! To authenticate users, he created a database with all of his users' usernames and the SHA-256 hash of their passwords. Unfortunately for him, Eve has stolen the contents of this database.

In this problem, we will explore some of the space/time tradeoffs which Eve can use to invert cryptographic hashes of passwords. Suppose Eve knows that all the passwords come from a set S, and our hash function produces d-bit strings. With no precomputation, if $|S|$ is much smaller than 2^d, then inverting the hash of a password will take Eve about $|S|/2$ tries in expectation, and therefore time that is $\Omega(|S|)$.

If Eve had infinite time to prepare for breaking the password hashes, she could construct a lookup table with the hash of every possible password in it. Inverting a password hash would just be an $O(1)$ table lookup, but Eve would then be required to store a table of size $\Omega(|S|)$.

(a) One early approach to achieving a finer-grained tradeoff between table size and lookup time is hash chains. Just as our cryptographic hash function h is a pseudorandom function mapping S to $\{0,1\}^d$, it is typically possible to construct a pseudorandom function f mapping $\{0,1\}^d$ to S. In particular, this is possible if S is *efficiently enumerable*. This means that there is an ordering on S such that the ith element of S can be efficiently computed.

If S is the set of all 10 character alphanumeric passwords, construct a pseudorandom function mapping $\{0,1\}^{256}$ to S.

(b) The next idea is that f and h are alternately applied k times on a random password to yield a chain of k passwords and password hashes. Show that if we store a single entry in a lookup table mapping the last hash of this chain to the starting password, then in $O(k)$ time, we can invert any of the hashes which occurred in the chain.

(c) A natural next idea is to store multiple (hash, password) pairs in the lookup table, all constructed in the same way. Show that if all the corresponding chains are of length k, then we can efficiently invert the hash of any password which occurs in any of the chains. If you create $|S|/k$ chains of length k as described above, do you expect the chains to contain close to $|S|$ distinct passwords?

(d) Rainbow tables use a different and independent pseudorandom function f_i mapping $\{0,1\}^d$ to S for each step of the chain. Why does this increase the number of distinct hashes you expect the chains to contain?

(e) How can Ben Bitdiddle modify his password storage scheme to make it infeasible for a precomputed rainbow table to significantly weaken his users' passwords?

Problem 2-2. Password Cracker

You're working for the NSA and have been assigned the task of breaking into the most top secret government agencies. To do so, you must crack as many passwords that have been hashed using the MD5 algorithm as you can. The hashes have been included in a separate file (*hashes.txt*) and have been divided into weak, moderate and strong categories. Try to break as many of them as you can!

Please submit three python lists, one for each category, of cracked passwords. Leave passwords that you are unable to break as an empty string. For example:

Weak:

1. 1f3870be274f6c49b3e31a0c6728957f
2. fe01d67a002dfa0f3ac084298142eccd

Answer:

Weak:

['apple', '']

Please use the Problem 2-2 python template provided.

Problem 2-3. Scalability of alternate currencies.

All information about transactions in Bitcoin are stored into blocks that are joined together to form the block-chain.

After receiving a new block each node in the network verifies (a) the proof-of-work property; and (b) that all ECDSA signatures corresponding to the transactions are valid. If both checks pass the block is propagated further, otherwise it is rejected.

Currently neither (a) nor (b) pose a CPU bottleneck, because cryptographic operations (SHA256, RIPEMD-160 hash computations and ECDSA signature verification) involved in (a) and (b) are very fast.

Some proposed extensions to BitCoin (e.g. ZeroCoin and ZeroCash) would make BitCoin more anonymous, but all make transaction verification in part (b) more expensive. E.g. in ZeroCoin the fast ECDSA signature verification would be replaced by a much slower cryptographic checks.

Suppose that Ben Bitdiddle has invented yet another public-ledger based currency BenCoin, that works just like Bitcoin, but with slower transaction verification times.

(a) The transaction verification time for BenCoin is terrible: 1 second per transaction on a typical computer. What effects does this have on block propagation?

(b) Do dishonest minorities have easier time attacking BenCoin than they would have attacking Bitcoin? Why?

(c) Alyssa suggests to propagate block immediately after proof-of-work checks pass and then discard it (after sending it off to others) if transaction checks fail. Would this suggestion improve the scalability of BenCoin?