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

Handout 3
Feburary 26, 2018
Due: March 12, 2018

Problem Set 2

This problem set is due on *Monday, March 12, 2018* 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_pset4_problem1.pdf*).

You are to work on this problem set in groups. Like Problem Set 1, we will randomly assign the groups for this problem set. From the next problem set onwards, you will work 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 2-1. Range Change

In the course, we have considered hash functions that output *bitstrings* in some fixed range $D = \{0,1\}^\kappa$.

In this problem, we are interested in changing the **range** of the hash function to another range S. For example, the output domain S, of the hash function, might be the set of all permutations on k symbols, or the set of all sequences of six English words (from some fixed dictionary).

We imagine doing this by using some fixed public deterministic function f that maps D to S.

That is, we transform an original hash function $h : \{0,1\}^\star \to D$ to a modified hash function $g : \{0,1\}^\star \to S$ as follows:

$$g(x) = f(h(x)) \ .$$

State necessary and sufficient conditions (on f and perhaps S) for g to be collision-resistant, assuming that h is collision-resistant and argue why that is.

A condition is sufficient if for any hash function h, and any function f satisfying the condition, $f(h(\cdot\ ))$ is collision resistant. And a condition is necessary if for every function f that violates the condition, there exists a collision resistant hash function h such that $f(h)$ is not collision resistant.

Problem 2-2. Side-Channel Attack on Simon

An attacker can use "side-channel" information to obtain a secret Simon key.

We set up a server to simulate this type of side-channel attack. Our server encrypts random plaintexts using 68-round Simon128/128 with a secret key k (both blocksize and key are 128-bits).

Simon is a block cipher with a balanced Feistel structure, which means it is composed of rounds where the output of one round is fed into the input of the next round. If you look at the Simon Wikipedia entry

(`https://en.wikipedia.org/wiki/Simon_(cipher)`), you will see on the right a diagram of one round of the Simon cipher. When executing the cipher, our code also produces some side-channel information that you will use to recover the key k used in the cipher.

The side-channel information is computed as follows: after each round of the cipher, the total number of ones in the ciphertext CT1 and CT2 are counted. The one-count for a single round is added to the one-count of all other rounds, and when the cipher is complete the side-channel information you will receive is the cumulative one-count of all 68 rounds of the cipher.

If you query `http://6857simon.csail.mit.edu/?num=1000` you will receive num pairs of the form (128 bits of plaintext, total one-count). In this case, $num = 1000$, but feel free to specify any $num \in [1...10000]$ (it might take a bit to load the pairs for a large num). Also, feel free to query the server multiple times if you need more pairs, because the plaintexts are randomly generated so you will get new data.

The server returns the pairs in json format. The side-channel one-count is an integer between 0 and $68 \cdot 128 = 8704$. (There are 68 rounds and the state is 128 bits).

We have provided our server implementation in the files `server.py` and `simon.py`. The `simon.py` code contains the side-channel leak. Feel free to play with it on your local machine.

(a) Let X be a random variable that is the total number of heads in t independent coin-flips of a fair coin, and let Y another independent random variable that counts the number of heads in t coin flips, and then adds one. Suppose t is known.

Let Z is either X or Y , but you don't know which. How many draws of Z do you expect to need, in order to determine with reasonable reliability whether Z is X or Y?

(b) Describe an algorithm for recovering the Simon key k given the Simon side-channel information described above. (Hint: try using the result of part (a) to recover each bit of the low 64 bits of the first round key, then recover the low 64 bits of the second round key.)

(c) Recover the secret key k. Submit any code you used.

(d) How many pairs did your attack require? (Note that even when your attack seems to recover almost all the key bits correctly, it may still get one or two key bits wrong, which will result in an incorrect decryption of a ciphertext. Please report the number of plaintexts for which your algorithm has a good chance of outputting all key bits correctly.)

Problem 2-3. Differential Privacy

Consider an election with k candidates and n voters.

Each voter gets to 1 vote, which she can divide between the candidates as she pleases. That is, she submits a vector v of k nonnegative real numbers that add to 1.

We want to output the overall distribution of votes in an epsilon differentially-private manner, for a given epsilon.

Input:

- $v_1, ..., v_n$. These are length-k vectors describing each person's vote.
- ϵ. A positive real number.

Output:

- (An approximation to) the sum of the v_i . This sum is a length-k vector.

Proposed Algorithm:

- Compute $s = \sum_{i=1}^{n} v_i$.
- Sample Z, a vector of k i.i.d. Laplace random variables with spread parameter b. (That is, set $\mu = 0$ and $b = 2/\epsilon$ in the notation of `https://en.wikipedia.org/wiki/Laplace_distribution`.)
- Release $Y = s + Z$.

(a) Show that the L_1 norm of $s(v) - s(v')$ is at most 2, where v and v' are two datasets which differ only in the vote cast by one voter.

(b) Let g and g' be the probability density functions of the distributions on Y obtained by running the algorithm above on inputs v and v' respectively.

Show that for every possible output vector Y in R^k, the ratio $g(Y)/g'(Y)$ is bounded above by $\exp(2/b)$.

(c) Argue that it suffices to set $b = 2/\epsilon$ to satisfy ϵ-differential privacy.