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

Handout 5
April 3, 2017
Due: April 17, 2017

Problem Set 4

This problem set is due on *Monday, April 17, 2017* 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 with 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. Side-Channel Attack on Speck

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

We set up a server to simulate this type of side-channel attack. Our server encrypts random plaintexts using 32-round Speck128/128 with a secret key k. The server "leaks" the total number of ones of the XOR outputs during each encryption (counting only the outputs of the XOR's after applying round functions on the current states, not including the key scheduling part).

If you query `http://6857speck.csail.mit.edu:4000?num=1000` you will receive num (high 64 bits of plaintext, low 64 bits of plaintext, XOR-output one-count) triples. In this case, $num = 1000$, but feel free to specify any $num \in [1 \dots 10000]$ (it might take a bit to load the triples for a large num). Also, feel free to query the server multiple times if you need more triples, because the plaintexts are randomly generated so you will get new data.

The server returns the triples in json format. The XOR-output one-count is an integer between 0 and 32*128 = 4096. (There are 32-rounds and the state is 128 bits)

We have provided our server implementation in the files server.py and speck.py. 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 Speck key k given the Speck 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 triples 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 4-2. Elliptic Curves In last weeks recitation we covered calculations on elliptic curves. You can look at the lecture notes on elliptic curves where the operations on elliptic curves are defined. You can use any programming language environment to perform your computations. Specifically you may find SageMath useful (http://www.sagemath.org).

Consider the following elliptic curve parameters:

- The elliptic curve equation is $y^2 = x^3 + 3x + 4 \pmod{p}$
- The prime number is $p = 2017$

Let $G = (240, 864)$ be a generator for the elliptic curve group mentioned above.

(a) Compute P=100G

(b) Let $P = (165, 122)$ and $Q = (995, 1228)$. Compute $P + Q$.

(c) Using the same curve, and the point G, Alice and Bob would like to share a key using the (Elliptic Curve) Diffie-Hellman key exchange protocol. Alice picks $s_A = 17$ and Bob picks $s_B = 38$. What messages do they send each other, and what is their shared secret? Show your work and be sure to verify that Alice computes the same secret as Bob.

Problem 4-3. Pederson Commitments A quadratic residue (QR) mod p is any element of the form $x^2 \mod p$.

(a) Write an algorithm that given an element y in Z_p^* outputs 1 if y is a QR $\mod p$, and outputs 0 otherwise.

(b) Recall that in Pederson's commitment, g is a generator of a prime ordered group (denote this prime by q), $h = g^a$ is a random element in the group generated by g, and a commitment an element $x \in Z_q$ is $com(x; r) = g^x \cdot h^r$.

i What if we implement Pederson's commitment when g is a generator of Z_p^*. Is the hiding property maintained? Explain why or why not.

ii Let p be a safe prime, i.e. $p = 2q + 1$ where q is prime. If we wish to implement Pederson's commitment over Z_p^*, which element g should we use? Describe an algorithm to generate such an element g, and briefly explaiin or show why it works.