Massachusetts Institute of Technology | Handout 2
6.857: Network and Computer Security | Feburary 11, 2019
Professors Ronald L. Rivest and Yael Tauman Kalai | **Due:** Feburary 25, 2019

Problem Set 1

This problem set is due on *Monday, Feburary 25, 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.* 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 in groups. For problem sets 1, 2, and 3, we will randomly assign the groups for the problem set. After problem set 3, you are to work on the following problem sets 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 1-1. Security Policy for Delivery Robots

In January 2019, Amazon started testing autonomous package delivery with little robot vehicles. See `http://fortune.com/2019/01/24/amazon-delivery-robots-seattle/` or `https://www.dailymail.co.uk/sciencetech/article-5354457/` for more information. (Feel free to find or use other relevant sources.)

Write a set of basic functionalities that these sidewalk robots should have, and then describe an ideal security policy for these functionalities. Who can control these robots, and what happens if a robot is being attacked? How fast can the robot move, and what happens if the sidewalk is blocked? What other situations might arise and how should the robot and other agents react?

The policies you come up with should address each of the security goals discussed in class, though focus on the one(s) that are most relevant for robots. Given the time constraints and the complexity of the problem, we expect your solutions to be less than comprehensive. That being said, keep in mind that an adversarial party may try to interact with the robot and try to cause some undesirable result.

(This problem is a bit open-ended, but should give you excellent practice in writing a security policy. We have included sample solutions from similar questions in previous years on the course website.)

Problem 1-2. One-Time Pad

The country of Bitcoinistan encrypts messages between its embassies using a "one-time pad" scheme. Here each 8-bit message byte m_i is encrypted by exclusive-oring it with an 8-bit pad byte p_i to obtain an 8-bit ciphertext byte c_i:

$$c_i = m_i \oplus p_i$$

where $\oplus$ is the exclusive-or operator on eight-bit values.

A n-byte message $M = (m_1, \ldots, m_n)$ is encrypted using an n-byte pad $P = (p_1, \ldots, p_n)$ to obtain an n-byte ciphertext $C = (c_1, \ldots, c_n)$. Message bytes are encrypted using the standard UTF-8 encoding: `https://en.wikipedia.org/wiki/UTF-8` (which is equivalent to ASCII for the usual characters).

This approach would be perfectly secure if each n-byte message M was encrypted with a unique n-byte pad P chosen uniformly at random from the set of all 256^n possible pads. We show in lecture that OTP is informationally-theoretically secure.

However, Bitcoinistan spends most of its resources mining bitcoins, and doesn't have enough resources left over to generate pads properly. Instead of generating one eight-bit *byte* p_i per message byte m_i, it generates only *one bit* r_i per message byte, and defines the eight-bit pad byte p_i as

$$p_i = reverse(p_{i-1}) + (r_i * \text{0x5A})$$

where $reverse(x)$ denotes the reversal of an eight-byte value x (so $reverse(00110101) = 10101100$), where $+$ and $*$ denotes addition and multiplication modulo 256, and where 0x5A is the eight-bit byte constant `01011010`. Also, $p_0 = 0$ by definition. They call this scheme the "Cheap One-Time Pad" (COTP).

COTP saves Bitcoinistan a factor of eight in random-bit generation costs.

Note that given pad byte p_{i-1} there are only two possibilities for pad byte p_i, since r_i is just a bit (either 0 or 1).

The goal of this problem is to show that COTP is breakable, by means of an example.

The staff has prepared the encryption of a English paragraph using COTP and posted both the ciphertext and the encryption code on the class website (see `cipher.txt` and `cotp.py` under the problem set handout). This message is 504 characters long, and contains upper and lower-case letters, commas, spaces, and periods only. As stated above, all characters are represented as 8-bit bytes with the usual US-ASCII encoding (e.g."A" is encoded as 0x41).

(a) Carefully explain how you can attack this scheme to derive the message.

(b) Implement your attack scheme in Python. Submit your Python code. (Note, your program may, if you wish, use a list of English words, obtainable for download in many places on the web.)
NOTE: The file `cotp.py` uses the library `bitstring.py`, available either via `pip` or on the class website. This file is not necessary for your solution, but it is necessary to run `cotp.py` and may be helpful in your solution.

(c) Run your code on the provided ciphertext. What is the hidden message? (If you can only obtain *some* of the message, give that.)

Problem 1-3. EFAIL

OpenPGP and S/MIME are two standards that offer *end-to-end encryption* of emails. You can read a little bit about end-to-end encryption at `https://blog.mailfence.com/end-to-end-email-encryption/`. (It's not mandatory to read the whole article; only the first few sections are relevant to the problem.)

(a) To start off, how does end-to-end-encrypted email differ from normal email (say, managed by Google or MIT)? Who might be able to read your normal Gmail emails, and who can read emails that have been encrypted by OpenPGP? Who can't read your emails regardless?

Recently, researchers discovered vulnerabilities christened "EFAIL" in the OpenPGP and S/MIME which could allow attackers to view your email. Read the writeup on the vulnerability at `https://efail.de/`. (The sections about CBC and CFB gadgets are somewhat out of scope for this problem, and we'll focus on the direct exfiltration attack. If you'd like to dive deeper, you can also watch the conference presentation at `https://www.usenix.org/conference/usenixsecurity18/presentation/poddebniak`).

(b) Briefly describe the model of the attacker. What data does the attack require? Does the attack require any special resources or servers? With this attack, now who could read your encrypted emails?

(c) The direct exfiltration attack falls into a broad class of attacks called *replay attacks*, in which an attacker can resend correctly encrypted data to achieve adverse results, even without knowing the plaintext or encryption key. More sophisticated replay attacks may use only a fragment of the original ciphertext/plaintext. Describe one way to prevent, or at least detect a replay. What requirements or computational overhead does this impose on your email client/email server?