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

Handout 2
February 10, 2014
Due: February 24, 2014

Problem Set 1

This problem set is due on *Monday, February 24* 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 1-1. Security Policy for Github

Write a security policy for a distributed version control hosting site like Github. Make sure to define relevant roles, functions, and policies. Your policy should be of the sort that would be usable to an implementor of a clone of Github. For the purposes of this problem, assume that Github is the maintainer of Git.

If you can't find relevant material on Github as currently implemented, invent new material as appropriate. Try to be as complete as you can, but emphasize the Github-specific aspects—in particular, what security goals are the most relevant for Github? What roles does (or, rather, *should*) Github have, and what should each principal be allowed to do?

(This problem is a bit open-ended, but should give you excellent practice in writing a security policy. Also, you may actually care about such security policies for designing systems used by large numbers of people—or if you use Github yourself! We have included sample solutions from similar questions in previous years on the course website.)

Problem 1-2. One-time pad with ciphertext feedback

It is well known that re-using a "one-time pad" can be insecure. This problem explores this issue, with some variations.

In this problem all characters are represented as 8-bit bytes with the usual US-ASCII encoding (e.g."A" is encoded as 0x41). The bitwise exclusive-or of two bytes x and y is denoted $x \oplus y$.

Let $M = (m_1, m_2, \ldots, m_n)$ be a message, consisting of a sequence of n message bytes, to be encrypted. Let $P = (p_1, p_2, \ldots, p_n)$ denote a pad, consisting of a coresponding sequence of (randomly chosen) "pad bytes" (key bytes).

In the usual one-time pad, the sequence $C = (c_1, c_2, \ldots, c_n)$ of ciphertext bytes is obtained by xor-ing each message byte with the corresponding pad byte:

$$c_i = m_i \oplus p_i, \text{ for } i = 1 \ldots n \, .$$

When we talk about more than one message, we will denote the messages as $M_1, M_2, \ldots, M_k$ and the bytes of message M_j as m_{ji}, namely $M_j = (m_{j1}, \ldots, m_{jn})$; we'll also use similar notation for the corresponding ciphertexts.

(a) Here are two 8-character English words encrypted with the same "one-time pad". What are the words?

```
e9 3a e9 c5 fc 73 55 d5
f4 3a fe c7 e1 68 4a df
```

Describe how you figured out the words.

(b) Ben Bitdiddle decided to fix this problem by making sure that you can't just "cancel" pad bytes by xor-ing the ciphertext bytes.

In his scheme the key is still as long as the ciphertext. If we define $c_0 = 0$ for notational convenience, then the ciphertext bytes $c_1, c_2, \ldots, c_n$ are obtained as follows:

$$c_i = m_i \oplus ((p_i + c_{i-1}) \bmod 256) .$$

That is, each ciphertext byte is added to the next key byte and the addition result (modulo 256) is used to encrypt to the next plaintext byte.

Ben is now confident he can reuse his pad, since $(k_i + c_{i-1}) \bmod 256$ will be different for different messages, so nobody would be able to cancel the k_i's out. You are provided with `otp-feedback.py`, which contains an implementation of Ben's algorithm.

You are also given the file `tenciphs.txt`, containing ten ciphertexts $C_1, C_2, \ldots, C_{10}$ produced by Ben, using the *same* pad P. You know that these messages contain valid English text.

Submit the messages and the pad, along with a careful explanation of how you found them, and any code you used to help find the messages. The most important part is the explanation.

Problem 1-3. Detecting Pad Reuse

In the previous problem, we saw how to attack a scheme in which a one-time pad, or a scheme like it, is reused. This problem will walk you through how rare pad reuse for the standard one-time pad (not the previous part's scheme) can be efficiently detected. We will look at the extreme case in which a pad is reused just once.

The following fact may be useful in this problem: Let $R_p(n)$ be the longest run of heads in n coin flips, each of which is heads with probability p. With high probability as n grows, $R_p(n)$ is between $\log_{\frac{1}{p}} n - \log_{\frac{1}{p}} \ln \ln n$ and $\log_{\frac{1}{p}} n + \log_{\frac{1}{p}} \ln n$. See `http://www.cmb.usc.edu/papers/msw_papers/msw-070.pdf` for more information on this bound. Note: you are not responsible for this paper; it is provided as a reference only.

(a) Suppose you are given $poly(n)$ n-bit ciphertexts $c_1, \ldots, c_l$, each of which are properly encrypted with an independently uniformly randomly chosen one-time pad. Show that with high probability, the length of the longest repeated bitstring (i.e. a substring of both c_i and c_j for some $i \neq j$) is less than $\log_2 n + \log_2 \ln n$.

(b) We have collected data on the average length of the longest run of identical characters in identical positions in passages of English text by randomly sampling from the collected works of American writer Winston Churchill. You can find a graph of run length (in characters) versus log of the passage length in characters at `http://courses.csail.mit.edu/6.857/2014/files/run_lengths.png`. How do these run lengths compare to the previous part's upper bound on the longest repeated bitstring if English plaintext is US-ASCII encoded?

(c) Assume that each pair of plaintexts does share a long common run of identical characters, and any pair of ciphertexts with independently chosen pads does not. Show that given $poly(n)$ n-bit ciphertexts with total length N and one instance of pad reuse, you can find the two ciphertexts which share a key in time which is $\tilde{O}(N)$ in the total ciphertext length. This means that your solution should run in $O(N \log^c N)$ time for some constant c.

Hint: You may wish to read about and use suffix trees in your solution. They are a type of tree which store all of the suffixes of an n-character string, can be constructed in $O(n \log n)$ time and can be used, among other things, to find a string's longest repeated substring in linearithmic time by finding the deepest non-leaf node. Note that longest repeated substring isn't exactly what you need to solve this problem. If you're not familiar with suffix trees, you may or may not find suffix arrays simpler and easier to understand.