Massachusetts Institute of Technology
6.857: Network and Computer Security
Professor Ronald L. Rivest

Handout **??**
March 7, 2016
Due: March 21, 2016

Problem Set 3

This problem set is due on *Monday, March 21, 2016* 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_pset3_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 3-1. More Block Chains

This problem is intended to bolster your understanding of why two keys are necessary for using CBC–MAC mode on variable length messages. To clarify, when two keys are used in CBC–MAC mode, the final block of the ciphertext is encrypted using a different key than all other blocks. This step is crucial to preventing various types of attacks, here we will explore length extension attacks specifically.

Let us consider a variant of CBC–MAC mode, call it CBC-MAC', which encrypts all blocks using the same key k. Here the CBC–MAC' can be computed by first applying a block cipher in CBC mode to a message $M = (m_1, m_2, \ldots, m_n)$ with $IV = 0$. In general, each ciphertext c_i is computed as

$$c_0 = 0$$
$$c_i = E_k(c_{i-1} \oplus m_i)$$

The CBC–MAC' result is taken to be the last ciphertext, specifically c_n. (Note that we assume that the message is padded as usual to be an integral number of blocks long, by always appending a "1" and then appending enough zeros as needed to fill out the last block.)

The problems below will assume an adversary has access to a CBC–MAC' oracle $\mathcal{O}$, which accepts queries of the form $\mathcal{O}_k(M)$ and returns a CBC–MAC' of M under a fixed key k.

(a) Assume the CBC–MAC' oracle uses some fixed, publicly-known IV. Describe a length extension attack that an adversary could exploit to generate a new valid MAC. More specifically, for any M_1 and M_2 of the attacker's choosing, provide a method for generating a CBC–MAC' for a new message $M' = M_1||M_2$. The adversary cannot query $\mathcal{O}$ for CBC–MAC' on M'.

(b) Now assume that the CBC-MAC' oracle uses a fresh random IV for every query, which it returns with the CBC–MAC' result for that query. Find another length extension attack where M_2 does not need to be fixed before querying, again the adversary cannot query for the new message M'. Here the attack must simply produce a valid CBC–MAC' under k for some $M' = M_1||M_2$. In other words, the attack does not need to work for all M_1 and M_2.

Problem 3-2. Apple vs. FBI

Read the following paper by Phillip Rogaway: `http://web.cs.ucdavis.edu/~rogaway/papers/moral-fn.pdf`

Explain how his ideas may be applicable from the point of view of an Apple engineer. Be sure to discuss their possible relevance to the Apple vs. FBI case.

Problem 3-3. Spicy or Random? Distinguishing Reduced Round Salsa

A common way to encrypt messages of variable length is to use a stream cipher. These stream ciphers are commonly built upon trusted block ciphers. One such block cipher is the Salsa20 block cipher (`https://cr.yp.to/snuffle/salsafamily-20071225.pdf`). It is remarkably simple, consisting of only 32-bit xor, addition, and cyclic shifts, giving it great performance on commodity hardware. Much cryptanalysis has gone into the Salsa20 cipher (which consists of 20 rounds). No one has had any (public) success in recovering a secret key when more than 8 out of 20 rounds are run (`https://eprint.iacr.org/2007/472.pdf`). Let's see how we can do!

Instead of conducting key-recovery attacks, we will be looking at distinguishing Salsa20 ciphertexts from random.

One method for distinguishing a cipher is to use the Chi-Squared test `https://en.wikipedia.org/wiki/Pearson's_chi-squared_test#Discrete_uniform_distribution` . The Chi-Squared test helps determine how similar two distributions are. The Chi-Squared distribution is defined as

$$\chi^2 = \sum_{i=0}^{N-1} \frac{(E_i - O_i)^2}{E_i}$$

Where N is the number of possible events of a random variable, E_i is the expected value of event i, and O_i is the observed value of event i.

It has expected value and standard deviation

$$E(\chi^2) = N - 1$$

$$\sigma(\chi^2) = \sqrt{2(N-1)}$$

We will generate B blocks of salsa output by setting a random key and incrementing the nonce for each block. We can leverage this Chi-Squared distribution by considering the random variable R_i = the number of times a specific byte in the 512-bit output has value i throughout all of the B total output blocks. Now we can use the Chi-Squared distribution between R_i and uniformly random bytes. In this case, $N = 2^8$, and $E_i = \frac{B}{2^8}$. We then compute O_i by storing counts of each value of our byte of interest in the salsa output. Finally, we compute χ^2 and decide the output is not random if χ^2 deviates by more than three standard deviations from the mean.

Let $S_k(r, n)$ be an r-round salsa cipher with 256-bit key k, and nonce n, as describe in Section 4.1 of `https://cr.yp.to/snuffle/salsafamily-20071225.pdf`, *except with last initial state XOR-ing omitted.* We compute block i for $i \in 0, \ldots, B-1$ by computing $S_k(r, i)$, or encryptions of all zeros with nonce $= i$ in the python implementation (as in my_salsa.py).

Use the Chi-Squared test to distinguish $S_k(r, n)$ for as high of an r as you can. You may generate as many ciphertext blocks as you wish (arbitrarily high B). Also try varying which and how many bits of the output blocks you analyze. Please submit your code, an explanation of your distinguishing techniques, and your results to Gradescope.

We have provided a python salsa20 implementation in the file salsa20.py and an example usage of it in my_salsa.py.

Note: ChaCha is a different cipher than Salsa20, make sure you use a Salsa20 implementation.