Problem Set 4

This problem set is due *via email,* to `6.857-submit@theory.csail.mit.edu` on *Monday, October 23* by the beginning of class.

You are to work on this problem set in groups of three or four people. Problems turned in by individuals, pairs, pentuples, etc. will not be accepted. Be sure that all group members can explain the solutions. See Handout 1 (*Course Information*) for our policy on collaboration. If you do not have a group, let us know.

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 and Late Policy: Each problem is worth 10 points. Late homework will not be accepted without prior approval. Homework should not be submitted by email except with prior approval. (*Somebody* from your group should be in class on the day that the homework is due.)

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 on your homework.

Problem 4-1. Timing Attack on an RSA Exponent

A cryptosystem that seems secure on paper might be vulnerable, in practice, to attacks that exploit out-of-band information. One such style of attacks uses the time required to exponentiate a given value with a secret exponent (mod a known modulus) to deduce the value of the secret exponent. This attack works when the time required to do multiplication and modular reductions depends on the input data, as is often the case.

To learn more about this attack, read the first 6 sections of Paul Kocher's paper: *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems.* As usual, you can find a link under the *Resources* section of the course web site.

For this problem, we ask you to use the attack outlined in section 5 to figure out the secret exponent being used by a simple server setup by the TAs. Specifically, we have constructed a server that accepts, over a socket connection, a data request of the form: *(start, iters, exp, j).* The server then returns:

$$Var[\sum_{i=0}^{iters-1} t((start+i)^{k_j}) - \sum_{i=0}^{iters-1} t((start+i)^{exp})] \quad (1)$$

such that:

- the function Var returns variance,
- k_j is a secret exponent for group j, $0 \leq j \leq 13$,
- all exponentiations are done mod p_j, $0 \leq j \leq 13$,
- the function $t(a^b \ mod \ p)$ returns the time required for the calculation when using the square and multiply algorithm. (To help reduce the effects of random network—and local—overhead, we are, for this problem, simulating the values returned by t so they are more consistent with what you would expect for square and multiply, so you should disregard the specific magnitude of the timing values as the units are arbitrary).

Each group has been sent a group number from 0 to 13. Your goal will be to recover k_j, where j is your group number. Each secret exponent and public modulus are of size 64 bits. The value of your public modulus isn't really important, but we are happy to send it to you if you would like.

To make your life easier, we have constructed a python program skeleton to handle the socket communication with our server. You can download the file, *client.py*, from the *Resources* section. Scroll down to the bottom of the source file and you will see an example function call that returns the information described above.

Your TAs are not the world's most brilliant server programmers, so we are not exactly sure how well our server will hold up under heavy loads. Accordingly, we have, to start with, launched one server: *osprey.csail.mit.edu*. More servers will be added later on, and will be announced to the class via email. You can change which server you are accessing by changing the value of the HOST variable at the top of *client.py*. If you have trouble connecting to a server, let us know right away so we can fix it.

To prevent undue slow downs, we have programmed the servers to reject excessively large number of iterations. Here's a hint: we have found that around 1500 to 2000 iterations per bit is enough to achieve the needed significance for the variance calculations. Also, *client.py* includes code for you to do local calculations, so that you can check your algorithm locally (you will need to generate a "secret" exponent and a modulus).

Hand-in your groups secret exponent value (k_j) in base-10, hex, or binary. We ask that you also hand in your modified client.py used to produce the value, and provide a written explanation of how your attack worked, and what values you used. Partial credit will be rewarded for partially correct exponents.

Problem 4-2. Chosen plaintext analysis of a block cipher

A local security company has constructed what they believe to be a brilliantly simple, yet secure, block cipher that can be used to encrypt n^2 bit message blocks. Our suspicion, however, is that this scheme is vulnerable to a chosen plaintext attack (i.e., the adversary can provide arbitrary plaintexts and observe the resulting ciphertexts). In this problem, we will ask you to prove our suspicions correct.

The block cipher works as follows:

1. The sender and receiver share a secret key. Using a known transformation (the specifics of which are not really relevant here) the key can be used to generate an S-box which takes n input bits and returns n output bits. The S-box is one-to-one and onto.
2. To encode a n^2-bit message M, the sender first uses its key K to generate S-box T_K.
3. The sender then breaks up the message M into n groups of n bits, and arranges them as a n x n table. For example, when $n = 4$, the first row consists of bits b_0, b_1, b_2, b_3, the second row consists of bits b_4, b_5, b_6, b_7, and so forth. We call this table T_M.
4. The sender then performs the following *round*:
   - Replace each row $T_M[i]$ with $T_K[T_M[i]]$. That is, replace the n bits in each row of T_M with the n output bits you get when you feed them into the S-box T_K.
   - Transpose the rows and columns.
5. Repeat the previous step. Therefore, in total, we end up performing two rounds of row substitution and transposing.
6. To decrypt, the receiver can simply reverse the transpositions and row substitutions.
7. In addition, you can assume that for all K, T_K is defined such that $T_K(0^n) \oplus T_K(1^n) \neq 1^n$.

Argue that this block cipher can be broken using a chosen plaintext attack. That is, detail an efficient procedure for recovering the S-box. It may be helpful when thinking of attacks to think of $n = 4$.

Problem 4-3. Block Cipher Modes

(a) Consider a combination of encryption and authentication that uses CBC mode encryption on the message, then runs a CBC-MAC on the ciphertext to get a MAC. The same key and IV are used as inputs to both the encryption and the MAC. Assess the security of this scheme.

(b) Consider the CBC mode of encryption. Is it $\mathcal{IND}-\mathcal{CCA}$ secure? (See the paper by Desai to remind yourself of the definition of $\mathcal{IND}-\mathcal{CCA}$ security for symmetric block ciphers.)

(c) Suppose that we use a block cipher to encrypt plaintext blocks $P_0, P_1, ...$, according to the rule:

$$C_0 = IV \oplus E(P_0, K), C_1 = C_0 \oplus E(P_1, K), C_2 = C_1 \oplus E(P_2, K), ...$$

What is the corresponding decryption rule? Compare the security of this mode of operation with that of the CBC mode.