
Problem Set 3

This problem set is due *via email*, to `6857-hw@mit.edu` on *Monday, March 17* 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 L^AT_EX 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.

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 3-1. Secret Sharing

Suppose the 6.857 staff decide to use the ElGamal encryption scheme in order to receive secret mail. However, the staff does not want to have one secret key which decrypts any encrypted e-mail, as then if one of the staff members has their laptop stolen, then the thief can decrypt all of the secret 6.857 e-mail. Instead, the staff would like to implement a scheme where multiple staff members must cooperate in order to decrypt encrypted messages. The staff members should be able to cooperatively decrypt any encrypted message without having to reveal their own shares of the secret to each other. Essentially, each staff member should be able to compute a "share" of the decrypted message.

- (a) Design a public-key encryption scheme where the encryption algorithm and public key are the same as in ElGamal, but the secret key is shared amongst 4 decryptors (2 professors and 2 TA's), and in order to decrypt any message, either both professors must cooperate, or any one professor and both TA's must cooperate. Show how the private key shares are generated and distributed, and show how decryption is accomplished.
- (b) Now suppose the staff decide they prefer to use RSA public-key encryption. Can you easily extend your solution to use RSA, or do you see some type of difficulty in doing this?
- (c) How would you define a Chosen Ciphertext Attack in the context of this cryptosystem? Is this system vulnerable to Chosen Ciphertext Attacks as you have defined them?

Problem 3-2. RSA Timing Attacks

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:

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

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 $t(a^b \text{ mod } p)$ 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 is of size 64 bits. The value of your public modulus isn't really important, but can be found at the bottom of the python code we provide.

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, *6857.csail.mit.edu* will hold up under heavy loads. If you have trouble connecting to the 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 awarded for partially correct exponents.

Problem 3-3. Public Key Infrastructure

(Compare and contrast X.509 certificates and SPKI/SDSI certificates.)

An X.509 certificate roughly says: "I am the principal with distinguished name /A/B/C and public key K1, and I hereby certify that the principal with name /A/B/C/D has public key K2. (Certificate issued mm/dd/yyyy; expires mm/dd/yyyy).(signed by K1)"

A (simple) SPKI/SDSI certificate roughly says: "I am the principal with public key K1, and I hereby assert that if you know me by some name X, then you should know the principal with public key K2 by name X/D. (Certificate issued mm/dd/yyyy; expires mm/dd/yyyy.) (signed by K1)"

To get going, SPKI/SDSI automatically allows $hash(K)$ as a name for the principal with public key K ; so names in general take the form $hash(K)/A/B/C$, etc. There is no single "root" identified. Names in general are relative to the starting point K .

Compare and contrast these two forms of certificates as primitive building blocks for a public-key infrastructure (PKI). Discuss as many relevant aspects as you can, include signing documents, certificate chains, flexibility when organizations change or merge, keys are compromised, etc.