Problem Set 4

This problem set is due on *Friday, April 12* at **11:59 PM**. Please note that no late submissions will be accepted. Please submit your problem set, in PDF format, *by email* to `6857-staff@mit.edu`. Submit only one problem set per group (with all of your group members' names on it).

You are to work on this problem set with a group of three or four people. You may choose your own groups for this problem set. If you do not have a group, please email `6857-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 4-1. Discreteness is the Better Part of Valor

As seen in class, the Discrete Logarithm Problem is very important to public-key cryptography. In this problem, you will explore ways to solve this problem (using smaller primes than are typically used for cryptography).

(a) Assume you are given a k-bit prime p, generator g for $\mathbb{Z}_p^*$, and element $y \in \mathbb{Z}_p^*$, and you wish to find x such that $g^x \equiv y \bmod p$. Of course, a brute-force search would be expected to take $\Theta(p) = \Theta(2^k)$ time.

We know that x is at most $p-1$, so it is at most k bits long. Express it as $a \cdot 2^{k/2} + b$, where a and b are both $k/2$-bit numbers. (Essentially, a is the high-order bits of x, and b is the low-order bits of x.)

Consider the algorithm that calculates $g^i \bmod p$ and $(g^{2^{k/2}})^j \bmod p$ for increasing values of i and j, and tests each pair (i, j) to see if $g^i \cdot (g^{2^{k/2}})^j \equiv y$. With a smart algorithm, approximately how many pairs (i, j) do you expect to test (on average) in order to find x? How much memory do you expect this algorithm to use?

(b) Other algorithms for the discrete log exist that are more practical than the one in part (a). These generally work by defining a sequence of values x_i such that $x_i = g^{v_i} y^{w_i}$, using a deterministic function to define x_{i+1} from x_i. They then attempt to find a *cycle* in that sequence in order to find two indices i, j such that $x_i = x_j$. If $g^{v_i} y^{w_i} = g^{v_j} y^{w_j}$ for $(v_i, w_i) \neq (v_j, w_j)$, one can solve for the discrete logarithm of y.

Two examples of cycle-finding algorithms are:

- The *Pollard rho algorithm*: This algorithm uses two pointers (a "fast" and "slow" pointer that go through the sequence $\{x_i\}$ at different speeds. If there is a cycle, the "fast" pointer will make a full cycle and catch up to the "slow" pointer. For more information about this algorithm, you can consult *The Handbook of Applied Cryptography*, by Menezes, van Oorschot, and Vanstone, Section 3.6.3, which is online at `http://cacr.uwaterloo.ca/hac/`. (Wikipedia also has a reasonable summary. Note that there is also a similar "Pollard rho algorithm" for the factoring problem.) You can also find a suggestion for the function to obtain x_{i+1} from x_i here.

- The *Nivasch stack algorithm*: This algorithm maintains a *stack* of values in order with the least value on top. When a new element is processed, any element less than it is popped from the stack, and that element is pushed to the stack. Eventually, this algorithm detects pushing the same element that is already at the top of the stack, at which point you have found a cycle. For more information, see `http://www.gabrielnivasch.org/fun/cycle-detection`.

Consider primes of the form $p = 2rs + 1$ where r and s are prime. The multiplicative group $\mathbb{Z}_p^*$ has size $2rs$. If we consider all the *squares* (or *quadratic residues*) in this group, they form a *subgroup*–if g_0 is a generator of $\mathbb{Z}_p^*$, then $g_1 = g_0^2$ is a generator of the squares of $\mathbb{Z}_p^*$. This subgroup has order rs.

We can consider various subgroups of this group–for example, $g = (g_1)^s$ is a generator of a subgroup of order r, and $h = (g_1)^r$ is a generator of a subgroup of order s. Any square modulo p can be represented by a pair of elements of these two subgroups.[1]

Specifically, you can express each square z as $g^a h^b$ for $a \in \{0, 1, ..., r-1\}$, $b \in \{0, 1, ...s-1\}$.

On the class website, we have provided a text file with a table of decimal values. Each line is arranged in the form $i, r_i, s_i, p_i, g_i, h_i$. (For ease in notation, we will omit the subscripts in the rest of the explanation, so the values in a single line will be called i, r, s, p, g, h.) Each p is an i-bit prime that is equal to $2rs + 1$; r and s are both prime. g is a generator of the subgroup of size r modulo p, and h is a generator of the subgroup of size s modulo p.

On the class website, there is also a list of students and associated square numbers. Let z for your group be equal to the *product* of the numbers associated with your group members.

Find (a, b) such that $g^a h^b \equiv z \bmod p$ for as many i as you can.

Turn in:

- Your group's number z
- As many (a_i, b_i) as you can find such that $g_i^{a_i} h_i^{b_i} \equiv z \bmod p_i$ (please use decimal notation, and label which i they correspond to)
- Any code you used for this problem
- A brief explanation of how the code works (no more than half a page).

Please do not turn in large outputs/data–only the code and the answers.

Hint: You can certainly do this by taking discrete logs modulo p, but there's a more efficient way that involves taking discrete logs modulo smaller values. Consider Fermat's Little Theorem, and its extension to general finite fields.

Note that we do *not* expect you to get all of these!

Problem 4-2. The Secure Remote Password Protocol (SRP)

The SRP protocol is a protocol designed at Stanford that allows a client to authenticate to a server using his/her password as well as exchange a secret key with the server for future secure communication. The homepage of the protocol is `http://srp.stanford.edu/` which contains the protocol description:

- the main paper describing SRP is `http://srp.stanford.edu/doc.html#papers`
- a second paper describing a follow-up improvement to SRP is `http://srp.stanford.edu/srp6.ps`.

You might find the Wikipedia article `http://en.wikipedia.org/wiki/Secure_Remote_Password_protocol` useful to provide you with a quick overview and then you can refer to the papers above for details; the Wikipedia article already incorporates the improvement.

In this problem, we will explore the security properties of SRP.

[1] This can be seen as analogous to the Chinese Remainder Theorem. You will not need to apply the Chinese Remainder Theorem for this problem, but the idea that a value can be represented in this way is central.

(a) A first observation is that the server does not store in the clear the salted hash of the password, $H(\mathsf{salt}, I, p)$, for p being the password and I the identity of a user; instead, the server stores $g^{H(\mathsf{salt},I,p)}$ (concretely, it stores $\langle \mathsf{salt}, I, g^{H(\mathsf{salt},I,p)} \rangle$ for each user I). What is the reason for this decision? And what, if any, cryptographic assumption does the argument rely on? Can the adversary still mount a dictionary attack on $g^{H(\mathsf{salt},I,p)}$ as it would on $H(\mathsf{salt}, I, p)$?

(b) The Diffie-Helman key exchange protocol suffers from a man-in-the-middle (MITM) attack. Eve, the "man in the middle", can interpose in the key exchange protocol between Alice (with secret a) and Bob (with secret b). Eve can intercept the message from Alice g^a and respond with g^e to Alice (where e is some value chosen by Eve) and send Bob g^e and receive g^b from Bob. In this way, Alice establishes a secret key g^{ae} with Eve, and Bob establishes a secret key g^{be} with Eve, without knowing that Eve is in the middle. From now on, Even can snoop on all conversation of Alice and Bob, by playing as an intermediary on all their messages.

1. Explain why the SRP protocol does not suffer from the man-in-the-middle problem if the adversary does not have the $\langle \mathsf{salt}, I, g^{H(\mathsf{salt},I,p)} \rangle$ authentication information of the user. That is, why can't the adversary authenticate himself to the client as the server and to the server as the client?

Now consider that the adversary was able to steal the password tables $\langle \mathsf{salt}, I, g^{H(\mathsf{salt},I,p)} \rangle$ but it does not know and cannot compute $H(\mathsf{salt}, I, p)$. Can the adversary mount a MITM attack and be able to subsequently eavesdrop on the conversation between the server and the client? (That is, after the authentication protocol finished, a succesful MITM adversary is able to decrypt all the messages that the client and the server are sending to each other.) To guide you in answering this question, answer the following smaller questions:

2. Can the adversary use the stolen information and fool the client into thinking he/she is the server by successfully establishing a shared key with the client?
3. Can the adversary use the stolen information and authenticate himself/herself to the server as the legitimate client? In other words, at the end of the authentication protocol, can the adversary convince the server that their keys match?
4. If the adversary simply eavesdrops on the authentication protocol of the client and the server (by watching the values exchanged) and does not inject any messages, can the adversary use the $\langle \mathsf{salt}, I, g^{H(\mathsf{salt},I,p)} \rangle$ information to learn the secret key that the client and the server agree on?
5. Conclude whether or not the adversary is able to launch a MITM attack.

(c) Consider modifying the SRP protocol by setting $u := 1$ instead of $H(A, B)$. The SRP protocol now becomes less secure because an adversary who stole $\langle \mathsf{salt}, I, g^{H(\mathsf{salt},I,p)} \rangle$ from the server can now authenticate to the server as the legitimate client. How? Provide the concrete steps of the authentication protocol for such an attack.

Problem 4-3. Authenticated Encryption

The goal of authenticated encryption (AE) is to provide confidentiality, integrity, and authentication (CIA). A natural idea to construct such a scheme is to combine an encryption scheme and a MAC scheme; however, one must be careful about the method to combine these schemes for not all possibilities are secure.

Let MAC be a secure MAC scheme with the unforgeability property: no adversary can forge a MAC for a new message (for which it did not observe a MAC already) or a new MAC for a message it already saw. Let E be a symmetric key encryption scheme that is IND-CCA secure. Let k_E be a key for the encryption scheme and k_M be a key for the MAC scheme. The desired AE scheme should also be IND-CCA and have the property that an adversary cannot come up with a new encryption (that it did not see before) that passes the authenticity/integrity verification. (Warning: E is IND-CCA rather than IND-CPA or semantic security: similar questions have been answered online using IND-CPA, but things may be different with IND-CCA).

Below is a list of candidate AE schemes, where we only show the encryption algorithm (the decryption and verification algorithm is straightforward). For each of these, decide if it is a secure authenticated encryption scheme and describe a flaw or argue its security. We use | to denote concatenation.

(a) $\mathsf{AE}(k_E, k_M, \mathsf{message}) = E(k_E, \mathsf{message}) \ \mid \ \mathsf{MAC}(k_M, \mathsf{message})$.

(b) $\mathsf{AE}(k_E, k_M, \mathsf{message}) = E\Big(k_E, \mathsf{message} \mid \mathsf{MAC}(k_M, \mathsf{message})\Big)$.

(c) $\mathsf{AE}(k_E, k_M, \mathsf{message}) = E(k_E, \mathsf{message}) \mid \mathsf{MAC}(k_M, E(k_E, \mathsf{message}))$.