
Midterm Solutions

Submit this take-home midterm in PostScript, PDF, or MS Word format to `6857-submit@csail.mit.edu` before lecture on the due date. Each problem solution must appear on separate sheets of paper. Mark the top of each sheet with your name, the course number (6.857), the question number, and the date.

You must complete this midterm *individually*. You are not allowed to discuss or collaborate with your classmates in any way. You are only allowed to discuss the midterm with the TAs or Professor Rivest. You are free to use materials from books, papers, or the web, but you must cite any published resources you use. Cheating or plagiarism will result in a zero score and will be reported to MIT's Dean of Student Affairs.

Problem 1. Traditional Public-Key Encryption versus Identity-Based Encryption [15 Points]

In a traditional public-key cryptosystem, anyone may invoke a key generation algorithm G to produce a public/secret key-pair (PK, SK) . Each individual can make their own public key PK available through some Public-Key Infrastructure (PKI), as you did with your PGP key in the first problem set.

An encryption algorithm E allows anyone to encrypt a message m with a public key PK . A decryption algorithm D allows only the holder of a secret key SK to decrypt a ciphertext encrypted under the corresponding PK . This triplet of algorithms (G, E, D) have the following abstract definitions:

$$\begin{aligned} G(\cdot) &\rightarrow (PK, SK) \\ E(PK, m) &\rightarrow c \\ D(SK, c) &\rightarrow m \end{aligned}$$

Note that each of these algorithms has an implied source of random bits. In contrast, an Identity-Based Encryption (IBE) system consists of four algorithms (MKG, G, E, D) . First, a trusted third-party will invoke a master-key generation algorithm MKG to produce a master public key PK_m and a master secret key SK_m . The value PK_m will be made globally available, while SK_m will be kept secret by the trusted third-party. Given any plaintext identifier ID and SK_m , this trusted third-party can generate individual secret keys SK_{ID} using a key generation algorithm G .

Rather than encrypting messages under individual public keys, in an IBE scheme the encryption algorithm E only needs an ID and the global public key PK_m as input. Ciphertexts encrypted for ID can be decrypted using a decryption algorithm D that takes SK_{ID} as input. An IBE system is summarized as follows:

$$\begin{aligned} MKG(\cdot) &\rightarrow (PK_m, SK_m) \\ G(SK_m, ID) &\rightarrow SK_{ID} \\ E(PK_m, ID, m) &\rightarrow c \\ D(SK_{ID}, ID, c) &\rightarrow m \end{aligned}$$

Alice and Bob work for a large company that wants to let customers send encrypted messages to its employees. Bob suggests using a traditional public-key cryptosystem. He proposes that each employee generates a public/secret key-pair and posts the public-key in a directory on a company's public web server.

- (a) [4 points] Explain how Bob's system works. List all the initial one-time setup steps for the system, each employee, and each customer. After this initial setup, list all the steps necessary for a customer to send an encrypted message to an employee. For example, a single step might be "The customer generates a key-pair (PK_c, SK_c) ".

- (b) [4 points] Alice suggests using an IBE system where SK_{ID} values are distributed by an internal server that is only available to employees on the company's closed intranet. In her system, each employee's ID would simply be his or her e-mail address. Explain how Alice's system works. List all the initial one-time setup steps for the system, each employee, and each customer. After this initial setup, list all the steps necessary for a customer to send an encrypted message to an employee.
- (c) [7 points] Discuss the comparative advantages and disadvantages of Alice's and Bob's proposals. You may want to consider how their respective systems fare in some of the scenarios listed below. Please keep your answers brief and limit your discussion to the given scenarios:
- A new employee is hired and must be added to the system.
 - An employee is fired and must have his or her keys revoked.
 - The company's web server is vulnerable to a security exploit.
 - An employee loses his or her secret key.
 - The company merges with another company with either the same PKI, a different PKI, or no PKI at all.
 - A corrupt insider or disgruntled ex-employee wants to deny service or try to read other people's encrypted e-mail.
 - The company receives a subpoena to provide decrypted plaintext of the CEO's encrypted e-mail. (You may wish to consider this from the viewpoint of both a crooked CEO and a corporate governance officer.)

Solution: *Solution courtesy of Edmond Lau*

- (a) To set up Bob's traditional public-key cryptosystem, the following initial one-time setup steps must be taken:

1. The system sets up a Public-Key Infrastructure (PKI), i.e. a public web server, where public keys of employees can be published.
2. Each employee generates a key-pair (PK_e, SK_e) .
3. Each employee posts his public key in the PKI, i.e. in a directory on the public web server.

A customer who wants to send an encrypted message to an employee takes the following steps:

1. The customer obtains the public key PK_e of the recipient from the PKI. (Note that this step can also be considered an initial one-time setup step if the customer knows beforehand that he only intends to send messages to a limited set of employees; in that case, the customer can just obtain the set of public keys that he ever plans to use.)
2. The customer encrypts his message m with PK_e to obtain ciphertext $c = E(PK_e, m)$.
3. The customer sends the ciphertext c to the employee.

- (b) To set up Alice's Identity-Based Encryption system, the following initial one-time setup steps are necessary:

1. The system invokes the master-key generation algorithm MKG to produce a master key-pair (PK_m, SK_m) .
2. The system posts the one PK_m to a PKI, i.e. a public web server.
3. For each employee, the system generates an individual secret key $SK_{ID} = G(SK_m, ID)$, where ID is the employee's email address.
4. The system publishes a table of employees and their secret keys SK_{ID} on an internal server.
5. Each employee retrieves his own SK_{ID} from the internal server.
6. The customer obtains the public copy of PK_m .

A customer who wants to send an encrypted message to an employee takes the following steps:

1. The customer encrypts his message m to obtain ciphertext $c = E(PK_m, ID, m)$, where ID is simply the employee's email address.
2. The customer sends the ciphertext c to the employee.

(c) Comparative advantages and disadvantages of Alice and Bob's proposals:

- It takes more work for the company to add a new employee to Bob's system than to Alice's system. Bob's proposal only requires the employee to take action by generating a new key-pair (PK_e, SK_e) and by publishing his public key PK_e on the PKI. Alice's proposal, on the other hand, requires both the system and the employee to take action: the system must generate a new $SK_{ID} = G(SK_m, ID)$ for the employee and publish it on the internal server, and the employee must retrieve the new SK_{ID} . One disadvantage of Bob's system, however, is that a customer who wishes to send a message to the new employee would need to obtain yet another public key, whereas a customer using Alice's system would not need to take additional steps.
- If an employee is fired, Bob's system requires revoking a key from a public, external web server while Alice's system requires revoking a key from a closed intranet. Whereas the company clearly has the authority to revoke an old key from a closed intranet, it is not guaranteed that the company is able to revoke a key for the fired employee from a public web server, especially if the server is externally managed by some other authority. In that case, the fired employee would actually need to revoke his own key before leaving the job.
- A denial of service attack on the web server, in Alice's case, would prevent any new customers from obtaining the company's master public key PK_m and sending messages; all old customers with a local copy of PK_m would still maintain the ability to send messages to all employees. In Bob's case, not only would new customers be unable to send messages, but old customers would only be able to send messages to the limited set of employees whom they still had valid public keys for. If the adversary could somehow modify the public keys, then in Alice's case, there would be a blanket disruption of all messages sent with the modified PK'_m , which should be easily detectable, whereas in Bob's case, there would only be isolated instances of un-decryptable email for the few employees whose public keys were modified.
- Alice's proposal requires substantially less work than Bob's to remedy a situation where an employee Calvin loses his secret key. Under Alice's proposal, Calvin merely has to recheck the internal server to re-obtain the lost key. On the other hand, under Bob's proposal, the old key pair can no longer be used; Calvin would need to regenerate another key-pair (PK'_e, SK'_e) and post his new PK'_e on the PKI. Moreover, without his secret key or any other identifying information, Calvin would not be able to authenticate himself to revoke his old public key still on the PKI; instead, only the system would be able to do so, and only if it has the authority. Bob's proposal also requires that all customers who have an old local copy of Calvin's key to be notified that the old key is no longer valid.
- Alice's system deals more readily with mergers involving a company C without a PKI. In that scenario, Alice's system simply use the existing PK_m to generate new SK_{ID} 's for the new employees, and the employees can simply look them up as necessary, whereas Bob's system would require each employee of C to generate a new key-pair (PK_e, SK_e) and publish the PK_e onto the the PKI, thereby disrupting employees' work. On the other hand, Bob's system integrates more readily with another system using the same or different PKI's, requiring only that the two lists of public keys be merged into one. Alice's system, for consistency reasons, would typically require choosing one of the two PK_m 's as the future one to keep and then regenerating SK_{ID} 's for the other company based on the main PK_m . This other company's employees would also need to obtain their new secret keys, and those customers using the old PK_m would need to be notified of the conversion to the new PK_m .
- Alice's system is more susceptible to a corrupt insider; a corrupt insider who wants to read other people's encrypted emails has the potential to intercept the encrypted emails and look up the

individual's SK_{ID} from the internal server (provided he has some mechanism for doing so). Under Bob's system, the secret key only lies with the individual employee and is not available elsewhere.

- If the company receives a subpoena to provide decrypted plaintext of the CEO's encrypted email, a crooked CEO would prefer Bob's system because the plaintext can only be obtained if the CEO cooperates and hands over his secret key, which he can claim to "accidentally" lose. A corporate governance officer would much prefer Alice's system because even if the CEO fails to cooperate, the CEO's secret ID required for decryption would still be available on the company's internal server.

Problem 2. Hash Functions [10 points]:

Bob designed the following hash function h that works in the finite field $GF(p)$, where p is a large prime (e.g. 256-bits long). That is, all arithmetic is done modulo p . Your goal in this problem will be to show that h is neither one-way nor collision-resistant.

There is also available a highly complex and nonlinear function f that maps \mathbb{Z}_p one-to-one to itself, and such that both f and its inverse f^{-1} are efficiently computable by anyone. (There are no "secret parameters" for f .) It is assumed that a message M to be hashed is divided into a sequence of elements (M_1, \dots, M_n) , where each element M_i is in \mathbb{Z}_p . That is, $0 \leq M_i < p$ for all i .

The hash function has an internal state vector (x, y) , where each of x and y are in \mathbb{Z}_p . The state vector is initially set to $(x_0, y_0) = (0, 0)$. Then each message element M_i is processed in turn; causing the state to be updated to some value (x_i, y_i) . The final state, after M_n is processed, is (x_n, y_n) . The desired hash value $h(M)$ is just x_n .

To process M_i and update the state from (x_{i-1}, y_{i-1}) to (x_i, y_i) takes three steps, resulting in intermediate states (r_i, s_i) , (t_i, u_i) , and (finally) (x_i, y_i) . These steps are:

- **Add in message to both halves of state.**

Set $(r_i, s_i) = (x_{i-1} + M_i, y_{i-1} + M_i)$.

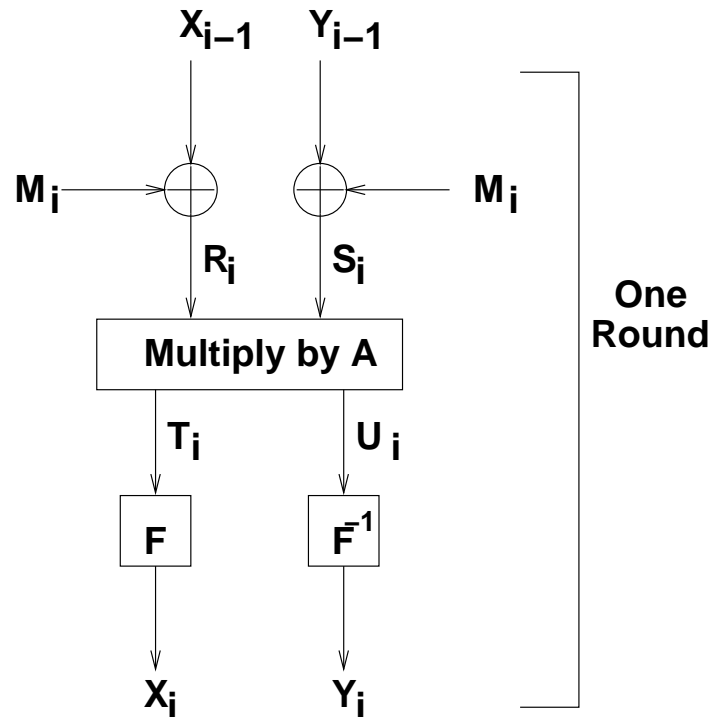
- **Multiply (r_i, s_i) by matrix $A = ((a, b), (c, d))$.**

Set $(t_i, u_i) = A(r_i, s_i) = (ar_i + bs_i, cr_i + ds_i)$, where A is non-singular (i.e. invertible) in $GF(p)$. (For consistency in your answers, let the inverse matrix A^{-1} be represented as $((a', b'), (c', d'))$ if you need it.) These matrices A and A^{-1} are of course public.

- **Apply non-linear function f and f^{-1} to each half.**

Set $(x_i, y_i) = (f(t_i), f^{-1}(u_i))$.

The following diagram illustrates this process:



Show that h is not one-way and not collision-resistant.

Solution: *Solution courtesy of Anthony Johnson*

The hash function described is not one-way, since one can easily reverse the entire process:

One-way: We need to show that we can get an x such that $h(x) = h$

Since every step is reversible, can we get the following steps: Given x_n , we calculate $t_i = f^{-1}(x_n)$.

We now make $r_i = s_i$. We calculate a solution (out of all of the possible solutions) that satisfies:

$$t_i = ar_i + bs_i = (a + b)(r_i)$$

Since we are simply trying to get an M that has the same hash function, we can make $M = r_i$. Since the first round of the hash function has 0 as the Initialization vector, we now have an M that has the same hash value.

DW: Several students pointed out that the above method does not work if $(a + b)$ does not have an inverse modulo p , since it relies on finding the message by taking $(a + b)^{-1}t_i \pmod{p}$. This happens when $a + b = p$. This is absolutely correct, although no points were deducted for not covering this possibility. A complete argument involves handling this separately as a special case. (Collision-resistance is easy, since all one-block messages hash to $M(a + b) = 0 \pmod{p}$; one-wayness is slightly more complicated but not too bad.)

Collision-Resistance:

We can generate a collision with any message that is longer than 256-bits. Due to the nature of the algorithm above, it will always generate a message that is 256-bits long. Therefore, to cause a collision, we can take

any message of length 512-bits or greater, calculate its hash, and then generate the h^{-1} as defined above. These will be two different messages with the same resulting hash value.

$$M_1 = x$$

$$M_2 = h^{-1}(h(x))$$

Problem 3. Web Authentication [5 Points]

Consider a web authentication system where after a user logs in with a user ID and password, the web server sets a cookie-based authenticator using a Message Authentication Code (MAC) as follows:

$$\text{Cookie} = (\text{user ID}, \text{expiration time}, \text{MAC}_k(\text{user ID} \oplus \text{expiration time}))$$

The user ID and expiration time are fixed-length binary integers. The key value k is a key known only to the web server. The \oplus symbol denotes the XOR operation.

- (a) [3 points] Suppose you know a valid cookie of the form (u, e, m) , where e is drawn from a uniform distribution. (This may be because the value e is determined by some arbitrary epoch, rather than by human time.) Suppose we know some other valid user identity v such that $u \neq v$. Show how to selectively forge a valid cookie for v and some expiration date $e' \geq e$ with probability at least $1/2$.

Solution: Let i be the most significant bit where u and v differ. In the case where $u_i = 0$ and $e_i = 0$, we have that $u_i \oplus e_i = v_i \oplus 1$. In the case where $u_i = 1$, if $e_i = 0$, again we have that $u_i \oplus e_i = v_i \oplus 1$. In these two cases, the value $u \oplus e$ will correspond to $v \oplus e'$ where $e'_i = 1$ and $e_i = 0$. Since i is the MSB, this means that $e' > e$. Since e_i is randomly distributed, these cases will occur with probability $1/2$. Thus, we can selectively produce a forgery (v, e', m) with probability at least $1/2$.

- (b) [2 points] Propose a similar web authentication scheme that addresses this selective forgery problem. Would your scheme still be secure if the MAC output were malleable?

Solution:

The simplest (and most common) construction was:

$$\text{Cookie} = (\text{user ID}, \text{expiration time}, \text{MAC}_k(\text{user ID}||\text{expiration time}))$$

Since the user ID and expiration time are fixed-length integers, there is no problem disambiguating between the two fields. This scheme might not be secure if the MAC output were malleable, since we might be able to produce related MAC outputs that correspond to some valid user ID and expiration time values.

Problem 4. One-time MAC's [10 Points]

Alice and Bob have agreed upon a sequence K_1, K_2, \dots , of shared secret keys that they wish to use to authenticate the messages they send back and forth to each other.

They are planning to use a “one-time MAC” scheme, as explained in 6.857. Each key K_i consists of a pair (a_i, b_i) of values in $\mathbb{Z}_p^* \times \mathbb{Z}_p$, so that the MAC of message M computed with the key K_i is just $\text{MAC}(K_i, M) = a_i M + b_i \pmod{p}$; here p is a publicly known large prime.

Alice and Bob agree on the following protocol for managing their use of keys. (Of course, they understand that each key K_i should not be used to MAC more than one message.)

Both parties work to keep track of the index t most recently used by either themselves or their counterparty. Then, when Alice sends a message M , she increments her t by one and then MAC's M with key K_t . The message she sends has the format:

$$(\text{"Alice"}, \text{"Bob"}, M, t, \text{MAC}(K_t, M))$$

When Bob receives this message, he checks that the MAC is correctly computed for the given value of t . If it is not correct for that value of t , he ignores the message altogether. If the MAC is correct for that value of t , Bob accepts the message and sets his local value of t to be equal to the t of the received message.

The same procedures are followed symmetrically (with roles switched) when Bob sends to Alice. (Each party only maintains a single value of t , however.)

- (a) [4 points] Identify as many vulnerabilities as you can in the above protocol design. You may assume that the messages are sent on a communications channel controlled by an active adversary that may insert, delete, modify, or replay messages on the channel.

Solution: Adversary delaying message to Bob can cause Bob to set t too low, leading to forgery.
 Can figure out key if can thus get Bob to issue two MACs on same t . (Disastrous!)
 Replay attack
 Potential DOS attack if can exhaust random numbers, e.g. if Alice always ack's with new key to authenticate.

- (b) [6 points] Propose a revised protocol that doesn't suffer from these problems.

Solution: *Solution courtesy of Fivos Constantinou*

(a) The above protocol design is susceptible to a number of attacks by an active adversary. The adversary can delete, modify or send properly authenticated messages down the channel. Here's a list of vulnerabilities in the protocol design:

- The adversary can delete a message from the channel without this being noticed. There is no way to verify that a message Alice send was received by Bob.
- Alice and Bob may both send a message at the same time, therefore, with the same t . The adversary who eavesdrops the channel will receive two messages that use the same key K_t to compute the MAC of the message. The adversary can compute K_t in the following way.

$$y = \text{MAC}(K_t, M) \equiv a_t M + b_t \pmod{p}$$

$$y' = \text{MAC}(K_t, M') \equiv a_t M' + b_t \pmod{p}$$

The adversary now has two equations with two unknowns so he/she can solve for a_t and b_t .

$$a_t \equiv \frac{y - y'}{M - M'} \pmod{p}$$

$$b_t \equiv y - a_t M \pmod{p}$$

Once the adversary has K_t he/she can use it to compute the MAC for any message. The adversary can, therefore, impersonate Alice or Bob.

- The adversary may also obtain a key K_t in the following way:
 –Alice sends message, ("Alice", "Bob", M , t , $\text{MAC}(K_t, M)$).

- Adversary deletes the message and any subsequent messages Alice sends so that Bob does not receive any of them.
- Wait for Bob to send a message. Bob's message will be: ("Bob", "Alice", M' , t , $MAC(K_t, M')$). As before, the adversary has two messages that use the same key, K_t to compute the MAC of the message. The adversary can therefore compute K_t and impersonate either Bob or Alice.
- The protocol simply sets the local value of t to be equal to the t of the received message, if it is properly authenticated. Using the method described above the adversary can compute a number of keys. Let's say the adversary has computed K_i . The adversary can send any message to both Alice and Bob and cause them to set their t value to i . The adversary can then modify any message sent and still produce a valid MAC. Let's say Alice sends a message ("Alice", "Bob", M , $i+1$, $MAC(K_{i+1}, M)$). The adversary can modify the message to ("Alice", "Bob", M' , i , $MAC(K_i, M)$). The adversary has changed the message M to M' and the t value from $i+1$ to i but he/she is still able to produce a valid MAC.

(b) A revised protocol could be the following:

Alice and Bob agree on the sequence of shared secret keys, K_1, K_2, \dots . The keys are generated as in the previous protocol but now Alice uses the odd number keys to compute the MAC of her messages while Bob uses the even number keys. This guarantees that Alice and Bob will never use the same key to compute a MAC so as long as one of them does not reuse the same key the adversary will not be able to compute any secret key. Both Alice and Bob maintain two variables, s and t , with s keeping a count of Alice's messages and t similarly for Bob's messages. A message Alice sends has the form, ("Alice", "Bob", M , s , $MAC(K_s, M)$), while a message Bob sends is of the form: ("Bob", "Alice", M , t , $MAC(K_t, M)$). When Alice wants to send a message she increments s by 2 and computes the MAC of the message using K_s , after she has incremented s . Similarly Bob increments t by 2 and computes the MAC using K_t . When Alice receives a message, she checks that the value of t is greater than the one she has locally and if so she checks if the MAC has the correct value and then sets her local value of t to match the message. If the message was not affected by the adversary t of the message should be two greater from the value Alice has for t . If the value is less the message must have been changed by the adversary since Bob would not reuse the same secret key. Given that the keys have only been used once so far the adversary should not have any of the keys and, therefore, the MAC code should not be valid. Hence, Alice can disregard the message. If t is by more than 2 greater than her local t then either t was modified by the adversary or some of Bob's previous messages have been deleted, either by the adversary or by an error in the transmission. If the adversary has simply modified t , then the MAC code should not be valid for the given K_t and thus the message can be disregarded. Since the adversary controls the communications channel he/she may simply delete some of the messages Bob has sent. Bob increments his value of t each time he sends a message so if Alice does not receive some of his messages, the messages she receives afterwards will have a higher than expected t . Alice can accept the message if the MAC is valid and she can set her value of t to match the one in the message. She will know, however, that some messages were either deleted or lost in the transmission and she can notify Bob about it. Bob acts similarly when he receives messages from Alice. He checks the value of s and updates his local value accordingly if he receives a valid message.

The proposed protocol does not suffer from any of the vulnerabilities mentioned in (a). Since Alice and Bob use different keys to compute the MAC for their messages, the adversary cannot obtain two messages authenticated with the same secret key. The two ways that the adversary could obtain two messages authenticated with the same secret key are not effective in this protocol. Moreover, by using two variables s and t , Alice and Bob control the value of their own message count and thus will never use the same secret key twice.

Security could be enhanced by encrypting the message sent or using public key cryptography to let the other person know which secret key to use. The protocol as it is, however, solves the vulnerabilities of the original protocol and, even though additional mechanisms are better they reduce the efficiency of communication.

Problem 5. Phun with ϕ [10 points]

- (a) [2 Points] Recall the definition of the ϕ function from class. Suppose that you are given a number n and know that it is the product of two distinct odd primes p and q . Which is “harder”, factoring n or computing $\phi(n)$ (or are they the same difficulty)? (If given the answer to one problem you can easily find the answer to another, the first problem is at least as hard as the second.)
- (b) [2 points] A *generator* g modulo prime p is an element of \mathbb{Z}_p^* of order $p-1$; any of the $p-1$ elements relatively prime to p can be expressed as a power of g . Recall that Fermat’s theorem states that for any a relatively prime to prime p , $a^{p-1} = 1 \pmod{p}$. A more general theorem (due to Euler) states that for any n , if a is relatively prime to n , then $a^{\phi(n)} = 1 \pmod{n}$. Thus, it is natural to think that there might be “generators” modulo composite numbers as well—that is, elements of order $\phi(n)$ modulo n .
- Assume that $n = pq$ where p and q are distinct odd primes. Find the smallest example you can where g is a generator both modulo p and modulo q , but is *not* a generator modulo n .
- (c) [3 points] Letting p and q be two distinct odd primes and $n = pq$, prove that there are no generators for \mathbb{Z}_n^* . A fact you should consider is that if g has order a modulo p and g has order b modulo q , then g has order $\text{lcm}(a, b)$ modulo n , where lcm is the “least common multiple” function.
- (d) [3 points] Again, let n be the product of two distinct primes. A “super-generator” is an element g such that

$$\mathbb{Z}_n^* = \{g^{\frac{a}{b}} : a \in \mathbb{Z}_{\lambda(n)}, b \in \mathbb{Z}_{\lambda(n)}^*\}$$

where $\lambda(n) = \text{lcm}(p-1, q-1)$. Note that $\mathbb{Z}_{\lambda(n)}$ is simply the integers \mathbb{Z} modulo $\lambda(n)$. If g generates \mathbb{Z}_n^* via rational powers (not just integral powers, as for an ordinary generator) then it is a super-generator. Note that $g^{\frac{1}{b}}$ can be defined as g^c where $b * c = 1 \pmod{\lambda(n)}$.

Show that \mathbb{Z}_n^* has no “super-generators”.

Solution: *Solution courtesy of Akshay Patil*

- (a) They are of the same difficulty. Given $n = p \cdot q$, we can easily calculate $(p-1)(q-1) = \phi(n)$. Given $\phi(n)$, we can calculate p, q by finding the roots of the equation $x^2 + (\phi(n) - n - 1) \cdot x + n = 0$.
- DW: For further explanation, this is because $\phi(n) = (p-1)(q-1) = pq - p - q + 1 = n + 1 - (p+q)$, so $\phi(n) - n - 1 = -(p+q)$. The equation thus becomes $x^2 - (p+q)x + pq = 0$, which has roots p and q and can be easily solved by the quadratic formula.*
- (b) 2 is a generator mod 3 and mod 5. 2 is not a generator mod 15 (2 has order 4 mod 15).
- (c) As stated, if g has order a modulo p , and order b modulo q , then g has order $\text{lcm}(a, b)$ modulo $n = p \cdot q$. Fundamental algebra tells us that if $x \leq x'$ and $y \leq y'$ and $x \cdot y = x' \cdot y'$, then $x = x'$ and $y = y'$. Since g would need order $(p-1)(q-1) \pmod{n}$ to be a generator and $a \leq (p-1)$, $b \leq (q-1)$, it holds that unless $a = p-1$ and $b = q-1$, there is no way that $(p-1)(q-1) = \text{lcm}(a, b) \leq a \cdot b$. But, since p and q are odd, $p-1$ and $q-1$ are both even so $\text{lcm}(a, b) = \text{lcm}(p-1, q-1) \leq \frac{(p-1)(q-1)}{2} < (p-1)(q-1)$, meaning that it is impossible for any g to be a generator modulo $p \cdot q$.
- (d) $g^{\frac{a}{b}} = g^{ac}$ where $b * c = 1 \pmod{\lambda(n)}$. Thus if $x = g^{\frac{a}{b}}$, then $x = g^{ac}$ for some $c \in \mathbb{Z}_{\lambda(n)}$. This implies that if g is a “super-generator” then g must also be a generator (i.e. every $x \in \mathbb{Z}_{\lambda(n)}$ can be expressed in the form $g^y \pmod{\lambda(n)}$). But from (c), we know that there are no generators under n , so there must not be any super-generators under n as well.

Problem 6. Password Storage [10 points]

We mentioned in class that a common method for implementing a password login system was for the server to keep a hash of each password locally. This way, an attacker who compromises the password file ideally learns nothing about the users' passwords. In practice, however, it is common to do things a bit differently: one typically stores a random *salt* s for each user along with $h(s||password)$.

- (a) [3 points] Assume that the hash function that is used is ideal. By “ideal” we mean a hash function has all the desirable properties of the oracle or “elf-in-a-box” model, including one-wayness and collision resistance. From a *theoretical* perspective, is there a security benefit to including the salt for a file that contains a single password? Why or why not?
- (b) [3 points] What is the *practical* benefit of using salts?
- (c) [4 points] You wish to use a password-based login system on a single server. Assume there are controls in place that prevent users from choosing weak passwords. Suppose an adversary is able to acquire the privileges necessary to read the password file. Although there are solutions involving multiple servers or trusted third-parties, is it possible for a single server to provide a higher level of security than what the password-salting system offers? Why or why not?

Solution: *Solution courtesy of Marios Assiotis*

- (a) There is no theoretical gain by salting the password. As it is only a single password that is stored, a malicious adversary can perform a dictionary attack against $h(s||password)$ in the same time as $h(password)$.
- (b) The practical benefit is that in a file with thousands of passwords it is very likely that two users would choose the same password therefore their hashes would be the same. Therefore it is likely that a malicious adversary that knows the password for one user might discover the password for another by comparing the hashed values. In addition to that it is easier to perform a dictionary attack in a file that has no salted passwords as all the attacker needs to do is compare the hashed values in the password file with the hashed values from a dictionary file. However if each user has a random salt then the attacker would have to run the dictionary attack on each user individually.
- (c) It is possible to shadow the password file (as done in many *nix systems). However if the attacker is able to read the shadowed file as well then there is not much more that can be done as the login program needs to be able to compare the login credentials supplied by a user at login time to some values stored in a password file.

DW: A variety of answers were accepted for part (c). In general, whether you answered “yes” or “no,” full credit was awarded as long as the answer was well-justified. A point or two may have been docked for proposing schemes that required more than just a password.

Problem 7. Secret-sharing: Sharing Shares [10 points]

The Peppy-Cola company maintains the secret formula for its soda drink in an electronic safe. The safe only opens when authorized to do so by an authorized set of employees. During the opening protocol, a set T of employees wishing to open the safe insert their ID smart-cards into the safe, which determines if T is authorized. (The notion of “authorized” is of course monotonic, so that if a set S is authorized, then so is any superset of S .)

Peppy-Cola’s authorization policy may be more complex than can be handled by ordinary “secret-sharing” schemes, which typically only allow any t out of n specified individuals to be authorized. For example, Peppy-Cola may wish to allow the safe to be opened if any 3 of their 7 vice-presidents agree, but any vice-president may designate a set of “personal alternates”, such that if any two of his alternates agree, it is as

if the VP himself agreed to request the safe to be opened. Indeed, Peppy-Cola wants total flexibility as to what authorization policies are allowed.

The electronic safe supports recursive tree-based secret sharing: a policy is represented by a tree, where the root is the main safe-opening key. Each node in the tree represents a secret. The leaf nodes represent secrets that may be stored on individual employee's ID cards. (A card may store more than one leaf-secret.) Each non-leaf node (including the root) has a single associated secret which is not stored anywhere, but which can be recomputed given a sufficient number of secrets from its children. (The threshold as to what is a "sufficient number" may vary from node to node.) Given enough leaf-secrets, some of the intermediate nodes can be computed, bottom up. If the root node secret is computed, the safe opens up.

Argue that the policies supported by the electronic safe are sufficiently expressive that any (monotonic) authorization policy from Peppy-Cola can be handled.

Hint: Consider the set of "minimally authorized sets" – sets of employees that are authorized but which are not authorized if any one employee is removed from the set.

Solution: *Solution courtesy of Kathryn Shih*

Any monotonic policy can be expressed as some list of all possible (and correct) minimally authorized sets. A group of people should be allowed access to the safe if and only if they can form at least one of the minimally authorized sets. This works; imagine it didn't, and there was some group of people that was supposed to be able to open the safe and couldn't. Either these people are a minimally authorized set, in which case the list was simply incorrect, or they aren't. By the definition of a minimally authorized set, if they should be able to access the safe and are not a minimally authorized set, it means they have more people than are required to open it. You can remove at least one person from the group, and be back to the original problem. Eventually you will either remove everyone from the group, in which case there's not actually any access controls on the safe and the minimally authorized set is the null set, or you'll encounter a minimally authorized set with some group of people that should have been on the list. Either way, you end up with a contradiction.

Since any monotonic policy can be expressed as a list of minimally authorized sets, it means that as long as the electronic safe can allow access to any combination of minimally authorized sets (and nothing else), it will be able to express any monotonic policy.

Luckily, expressing any combination of minimally authorized sets is easy. It should only take one minimally authorized set to open the safe, so put each minimally authorized set on its own branch off the root node and require one branch secret to compute the root node. Each branch should contain all of the individuals from a given minimally authorized set. It should take every individual on the branch to compute the branch secret. Each individual will then get one secret on their ID card for every minimally authorized set that they're a member of. (But the ID cards can store more than one leaf-secret, so this isn't a problem.) By construction, if you remove a member from the group and they can no longer form a minimally authorized set, they will be unable to compute the branch secret and unable to unlock the safe. On the other hand, if you have a superset of a minimally authorized set, you know that they will be able to compute at least the one branch secret from the minimally authorized subset (and possibly more branch secrets, but all they need is one), and will be able to unlock the safe.

SW: This problem may also have a nice, elegant solution based on the monotonicity of sigmoid functions or "threshold neurons".

Problem 8. Academic Honesty [2 points]

Write a couple of sentences to testify that you have not collaborated with anyone on this midterm and that you have cited all your sources.