Introduction to Cryptography
Courant, Fall 2016

Homework 1

Instructor: Oded Regev
Student: YOUR NAME HERE

Homework is due by **7am of Sep 19**. Send by email to both "regev" (under the cs.nyu.edu domain) and "avt237@nyu.edu" with subject line "CSCI-GA 3210 Homework 1" and name the attachment "YOUR NAME HERE HW1.tex/pdf", and please also bring a printed copy to class. Start early!

Instructions. Solutions must be typeset in LaTeX (a template for this homework is available on the course web page). Your work will be graded on *correctness*, *clarity*, and *conciseness*. You should only submit work that you believe to be correct; if you cannot solve a problem completely, you will get significantly more partial credit if you clearly identify the gap(s) in your solution. It is good practice to start any long solution with an informal (but accurate) "proof summary" that describes the main idea.

You are expected to read all the hints either before or after submission, but before the next class.

You may collaborate with others on this problem set and consult external sources. However, you must ***write your own solutions***. You must also ***list your collaborators/sources*** for each problem.

1. *(The group $\mathbb{Z}_p^*$)* Let p be an odd prime. We use $\mathbb{Z}_p^*$ to denote the multiplicative group of integers modulo p. (In mathematics the common notation is $(\mathbb{Z}/p\mathbb{Z})^*$.)

   (a) (1 point) Find an efficient algorithm that given $a \in \mathbb{Z}_p^*$ and an integer $b \geq 0$ computes $a^b \in \mathbb{Z}_p^*$. Can we simply compute a^b as integers and then reduce the result modulo p? (if not, say exactly why)

   Solution: The issue with taking a^b and then reducing modulo p is that if p is very large (such as 2^{1024}), the memory requirements may be impossibly large, since we may be forced to keep track of a number up to $O(p^p)$, which would require $O(plog(p))$ bits. Instead of doing this, we could take the modulus after each successive exponentiation, using the fact that $(a * a^{'} modp) = ((amodp) * (a^{'} modp))modp$. Then we would have to keep track of at most $O(log(p))$ bits, and perform b multiplications. Furthermore, rather than multiplying by a each time, we could square the result each time (with a possible final mutiplication by a if b is odd), requiring the same amount (to the order of magnitude) of memory, but reducing the the number of multiplications to $O(log(b))$.

   (b) (2 points) Find an efficient algorithm to check if a given $a \in \mathbb{Z}_p^*$ is a quadratic residue.

   Solution: Euler's criterion gives us the desired result. It says that if p is prime and $a = x^2$, $a^{\frac{p-1}{2}} = 1$. Therefore, we can simply plug a and p into this equation to test whether a is a quadratic residue. We can use the algorithm described in the previous problem to make this modular exponentiation efficient.

   (c) (2 points) What fraction of the elements of $\mathbb{Z}_p^*$ are generators? How does it behave asymptotically? (You can use Wikipedia for the latter; there is no need for very precise asymptotics, just the order of magnitude)

   Solution: We start with the fact that $\mathbb{Z}_p^*$ is cyclic. Let $g \in \mathbb{Z}_p^*$ be a generator of g, and let $x \in \mathbb{Z}_p^*$ be any element. I claim that x is a generator of $\mathbb{Z}_p^*$ iff x goes not divide (p-1). Since g is a generator, we can write b as $b = g^i$. We know from Fermat's Little Theorem that $g^{i*(p-1)} = 1$. Now we need to show that if q is a prime factor of (p-1), then $g^{\frac{i*(p-1)}{q}} \neq 1$ for any exponent

$\leq (p-1)$ If we want to minimize this exponent, we let $q = GCD(i,(p-1))$. If $q > 1$, then the exponent is an integer less than (p-1), and therefore x cannot be a generator. On the other hand, if q=1, i.e. if i and (p-1) are relatively prime, then we have that $g^{i*(p-1)} = 1$ minimally, i.e. $b = g^i$ is a generator. Thus we have proven that the set of generators of $\mathbb{Z}_p^*$ are precisely the elements that do not divide (p-1), so that there are $\phi(p-1)$ of them, where $\phi()$ is Euler's totient function. Now, since (p-1) is even, we can write $(p-1) = 2^k * t$ for some integer k and odd integer t. Similarly, note that $\phi(p-1) = \phi(2^k) * \phi(t) = 2^{k-1} * \phi(t)$. The the proportion of elements that are generators is then given by $\frac{2^{k-1}*\phi(t)}{2^k*t} = \frac{\phi(t)}{2t}$. Clearly since $\phi(t) < t$, this fraction is less than .5. As found on the internet, the fraction $\frac{\phi(t)}{t}$ approaches 1 asymptotically, and therefore the ratio of elements that are generators asymptotically approaches $\frac{1}{2}$ for large p.

(d) (2 points) Describe an efficient algorithm to check if a given $g \in \mathbb{Z}_p^*$ is a generator. Assume that the algorithm is also given a factorization of $p-1$. (It is not known how to perform this task efficiently without this factorization.)

Solution: We know that any element $g \in \mathbb{Z}_p^*$ must satisfy $g^{p-1} = 1$. Suppose that g is not a generator. Then there must be some integer $q < (p-1) such that g^q = 1$. By Lagrange's Theorem, $|g|$ must divide (p-1). Take the prime factorization of $(p-1) = p_1 * p_2 * ... * p_k$. Then there is some subset of the prime factors (call it S) such that $g \prod_{i \in S} p_i = (p-1)$. Now let $p_j \in S$ be any prime factor. Since $g^{p-1} = 1$, we can write $(g^q)^{\prod_{i \in S} p_i} = ((g^{q \prod_{i \in S, i \neq j} p_i})^{p_j}$. So there must be some $p_j \in S such that g^{\frac{p-1}{p_j}} = g^{q \prod_{i \in S, i \neq j} p_i} = 1$. Therefore, in order to check that g is a generator, it suffices to show that $g^{\frac{p-1}{q}}) \neq 1$ for each prime factor q of (p-1).

(e) (2 points) There is a known efficient algorithm that given a number n (in unary) outputs a uniform n-bit prime p, together with a generator g of $\mathbb{Z}_p^*$. How can that be in light of what we said earlier about the necessity of the factorization of $p-1$? Explain the apparent paradox and suggest a solution.

Solution: One could use the following algorithm: randomly generate an n-bit number p. Then use the algorithm developed in part 1 to determine whether the number is prime by choosing some base $a, 1 < a < (p-1)$ and calculating the exponent a^{p-1}, which has time complexity of $O(n)$. If the result is 1, then p is probably prime. We can repeat this procedure several times (say k times) to increase confidence in the primaity of p. Since $O(1/log(n))$ numbers in this range are prime, we should have to repeat this procedure $log(n)$ times in order to find a prime. This brings the expected time complexity of the whole algorithm up to $O(knlog(n))$. Now, if we carefully chose a so that it is relatively prime to (p-1), then we know that a is also a generator of the group Z_p^*. We can do this by picking p to be of the form 2*k+1, or in other words, picking a random digit to be set to 1, with all other digits except the rightmost set equal to zero. The 2*k is guaranteed to be relatively prime to a.

2. (5 points) *(Shannon)* Prove that in any perfectly secret shared-key encryption scheme, $|\mathcal{K}| \geq |\mathcal{M}|$.

Solution: Suppose that $|\mathcal{K}| < |\mathcal{M}|$, and that $N = |\mathcal{K}|$. Then suppose our implementation is that we split the message into chunks with at most N bits. WLOG, suppose that $N = |\mathcal{M}| - 1$. Then we have one message of length N, and a second message consisting of only 1 bit. In this scheme, We can see that a cyphertext with different bit values in the first bit and N+1-th bit must have come from a plaintext message with different values in those bits ass well. Similarly, if the cyphertext has the same bit values in the first and N+1-th bits, then the plaintext message must have also had either both0's or both 1's . Suppose that our cyphertext c has the same bit values in the 1st and N+1-th bit, $P[E_k(m_1) = c] = 0$ for all m_1 with different bit values in those places. On the other hand, $P[E_k(m_2) = c] = 2^{-N}$ for all m_2 with the same bit vaues in those places. We know this because as we learned in teh first homework, the XORed cyphertext is uniformly and independently distributed, so the probability of a message with bits 2 -N given the cyphertext is 2^{1-N}. Since there are only two different ways that the 1st and N+1-th bits can be set with the same value, the probability of any such message is 2^{-N}. In summary, for the messages $m_1 and m_2$ described, $P[E_k(m_1) = c] \neq P[E_k(m_2) = c]$, so perfectly security is violated.

3. *(Perfect secrecy.*[1]*)* Prove or disprove (giving the simplest counterexample you can find) the following statements about perfect secrecy for shared-key encryption. You may use any of the facts from class.

   (a) (1 point) There is a perfectly secret encryption scheme for which the ciphertext always reveals 99% of the bits of the key k to the adversary.

   Solution: Yes, this is possible. Suppose we have a key of length 100, and a message of length 1. Then we can pad the last 99 bits of the plaintext with 0's before XORing it with the key, thereby revealing the last 99 digits of the secret key. However, no information can be inferred about the first bit of the key or the original plaintext, and therefore the encryption scheme is still perfecty secret.

   (b) (2 points) There is an encryption scheme that is not perfectly secure, yet the adversary cannot guess the key with probability greater than $1/|\mathcal{K}|$.

   Solution: Yes, and in fact an example of this is the encryption scheme described in problem 2. While this encryption scheme is not perfectly secure because of the repeated first bit, nevertheless we do not gain any information about the key. For example if the 1st and N+1-th bit are both the same we learn something about the message: if the bits are 1,0 or 0,1 this tells us that the message has different bits in those places, but it does not tell us which ones, and therefre does not tell us anything about the key. So the probability of all keys given any message is still 2^{-N}

   (c) (2 points) In a perfectly secret encryption scheme, the ciphertext is uniformly random. That is, for every $m \in \mathcal{M}$, the probability $\Pr_{k \leftarrow \mathsf{Gen}}[\mathsf{Enc}_k(m) = \bar{c}]$ is the same for every ciphertext $\bar{c} \in \mathcal{C}$.

   Solution: This is not true. Consider the following counterexample: if we prepend all of our plaintext messages with a constant bit of 0, then the distribution of cyphertext will also have a

[1] Based on a question from Peikert's class

constant bit in this digit, meaning that the distribution is not uniformly random. However, this encryption scheme could still be perfectly secure. In general, the distribution of the cyphertext depends on the distribution of the messages.

(d) (5 points) Perfect secrecy is equivalent to the following definition, which says that the adversary cannot determine which of two messages was encrypted any better than by random guessing. Formally, for any $m_0, m_1 \in \mathcal{M}$, and any function $\mathcal{A} : \mathcal{C} \to \{0, 1\}$,

$$\Pr_{k \leftarrow \mathsf{Gen},\, b \leftarrow \{0,1\}}[\mathcal{A}(\mathsf{Enc}_k(m_b)) = b] = \frac{1}{2}.$$

Hint: recall Q3c from HW0

Solution: This is true. From homework 0 problem 3c, we have the the probability of guessing the correct distribution from which a sample was taken is $\frac{1}{2}\Delta(D_0, D_1) + \frac{1}{2}$. In this scenario the two distributions are the distribution of C given some message m_0 and the distribution of C given a different message m_2. Perfect secrecy implies that these two distributions are the same, which in turn implies that the statistical distance is 0, so the probability of guessing correctly is exactly .5 .

(e) (5 points) Perfect secrecy is equivalent to the following definition, which says that the ciphertext and message are independent (as random variables). Formally, for any probability distribution $\mathcal{D}$ over the message space $\mathcal{M}$ and any $\bar{m} \in \mathcal{M}$ and $\bar{c} \in \mathcal{C}$,

$$\Pr_{m \leftarrow \mathcal{D},\, k \leftarrow \mathsf{Gen}}[m = \bar{m} \wedge \mathsf{Enc}_k(m) = \bar{c}] = \Pr_{m \leftarrow \mathcal{D}}[m = \bar{m}] \cdot \Pr_{m \leftarrow \mathcal{D},\, k \leftarrow \mathsf{Gen}}[\mathsf{Enc}_k(m) = \bar{c}].$$

Solution: $\Pr_{m \leftarrow \mathcal{D},\, k \leftarrow \mathsf{Gen}}[m = \bar{m} \wedge \mathsf{Enc}_k(m) = \bar{c}] = Pr[M = m | C = c] * Pr[C = c] = Pr[K = k] * Pr[C = c] = Pr[K = k] * Pr[C = c]$, where the last equality holds because the question of getting a certain message given a cyphertext is simply a question of which key is being used. Now, $\Pr_{m \leftarrow \mathcal{D}}[m = \bar{m}] \cdot \Pr_{m \leftarrow \mathcal{D},\, k \leftarrow \mathsf{Gen}}[\mathsf{Enc}_k(m) = \bar{c}]$ =$\Pr_{m \leftarrow \mathcal{D}}[m = \bar{m}] * P[C = c]$ Cancelling the $P[C = c]$ term from both sides, we get the equation $Pr[K = k] = \Pr_{m \leftarrow \mathcal{D}}[m = \bar{m}]$. This means that the equivalence above holds if and only if the distribution of plaintext is exactly the same as the distribution for the keys; namely a uniform distribution. So in general, the statement is false.

4. *(Encryption schemes with a computationally bounded adversary.)*

Consider the scenario of an encryption scheme in which Alice wants to send a message to Bob in such a way that Eve, who monitors the transmission, cannot read the message.

(a) (1 point) Explain in one sentence why Bob needs to have a secret from Eve.

Solution: Bob needs to have a secret from Eve because otherwise he would have the same information as Eve, and therefore she would be able know everything that he knows, including the context of the original plaintext.

(b) (1 point) Explain in one or two sentences why Alice needs to have a secret from Eve.

> **Solution:** Alice needs to have a secret from eve because otherwise they would have the same information, meaning that Alice would only know the encrypted message, without any way of reading the original plaintext message.

(c) (2 points) Now assume that Eve is computationally bounded (i.e., is restricted to run in polynomial time in the length of the message). Does Bob still need to have a secret from Eve? Does Alice? (your feeling for the latter is enough)

> **Solution:** Yes, bob still needs to have a secret from eve, for the same reasons as above: if they have the same information, then by defintiion Eve would know the original message. Alice also needs to have secret from Eve, again for the same reason as above.

I'm done solving and want to know more! (ID 18764)

5. (2 points) *(Defining one-way functions.♣)* Next class we will define the notion of a *one-way function*. Informally, this is a function $f : \{0,1\}^* \to \{0,1\}^*$ that is (1) easy to compute, and (2) hard to invert.

(a) Suggest a way (or ways) to formally define it. After thinking about this question for at least 10 minutes and before writing your solution, click here for some food for thought (ID 19166)

> **Solution:** One could define 'easy to compute' as folloows: given some input of bits x, there is a function f such that the value f(x) can be calculated in polynomial time. We could define 'hard to invert' as folows: for an unknown input of bits x and a known function f and known computed value of f(x), it is not possible to guess the preimage of x in polynomial time, or in other words the inverse function has time complexity that is a negligible function of the length of x. If the function f is not injective, there may be multiple preimages of f(x). We could consider this function to be 'hard to invert' if it is impossible to calculate any or all of the pre-images of f(x). the latter condition would be stronger of course. Note that calculating the exact value of x would not be an appropriate definition, since the inverse of such a function does not exist, or in other words any such inverse function is not well-defined. Since the inversion problem amounts to evaluating the function f^{-1}, intuitively this would be asking the impossible of the adversary since no such function exists.

Next, for each of the following functions, say if you think it's one way according to your definition.

(b) The function that given an n-bit string outputs the same string with its first half zeroed out.

> **Solution:** While it would be impossible to guess the exact preimage of f(x) in this case, it is certainly possible to make a guess that maps to the preimage of f(x). For example the function f(x) itself is in the preimage of f(x). We could even guess the entire set of preimages by taking the second half of the string, and then appending all possible first halves of the string (of which there are $2^{\frac{n}{2}}$). So this is not a one-way function.

♣Questions marked with a club are more open-ended and meant to encourage you to think in preparation for next class. You are not expected to answer correctly. Instead, you are expected to spend time thinking about it.

(c) The function f on domain $\{1,\ldots,N\} \times \{1,\ldots,N\}$ that maps a pair (x,y) to their product xy.

Solution: The preimage of the function f is just the set $(a,b) : a,b$ are factors of xy and ab = xy. While the problem of factorization has a high complexity, there is a trivial case that has complexity O(1) – namely, if $x,y \leq \sqrt{N}$, we can simply take the pair (1, xy). Therefore this is not a one-way function. However, in the general case for arbitrary x and y, we must look at the worst case scenario. In general, the factorization problem has exponential complexity. So if we restricted our domain to be $\{\sqrt{N},\ldots,N\} \times \{\sqrt{N},\ldots,N\}$, then the function would be one-way.

(d) Choose elements $a_1,\ldots,a_n$ uniformly from $\mathbb{Z}_N$ for $N = 2^n$ and define $f : \{0,1\}^n \to \mathbb{Z}_N$ by $f(b_1,\ldots,b_n) = \sum_{i=1}^n b_i a_i$.

Solution: The pre-image of this function is the set of all combinations of the elements of $a_1,...a_n$ whose sum is equal to $f(b_1,...,b_n)$. In the general case, this involves taking every possible combination of $a_1,...a_n$, of which there are 2^n possibilities (not to mention the additional linear complexity of adding the numbers). Ignoring the effect of multiple combinations that lead to the same sum, which is negligible when n is large, the probability of guessing correctly is $O(2^{-n})$, so this function is indeed one-way.

(e) Same as previous part, except $a_1,\ldots,a_n$ are chosen uniformly from $\mathbb{Z}_2^n$.

Solution: In this scenario, the function is not one-way. We can construct a guess as follows: for each of the elements $a_i = 0$, there is no contribution to the sum, and therefore we can select the corresponding bits b_i randomly and uniformly from $\{0,1\}$ since either choice will be in the preimage. Since the bits where $a_i = 1$ are the only ones that may contribute to the sum, we know that exactly $S = f(b_1,...,b_n)$ of the corresponding b_i bits must be 1, with the rest of them 0. Therefore, we can set S of these bits to be set to 1, with the rest of them equal to 0 (for example, we could set the first n bits to 1). This guess can be performed in O(n) and we are guaranteed that it will lie in the preimage of f. Therefore the function is not one-way.