P U M ∴ C

PUMaC 2016 Power Round

"Out of nothing I have created a strange new universe." —János Bolyai

Rules and Reminders

1. Your solutions are to be turned in when your team checks in on the morning of PUMaC or emailed to us at **pumac2016power@gmail.com** by 8AM Eastern Standard Time on the morning of PUMaC, November 19, 2016 with the subject line "PUMaC 2016 Power Round." Please staple your solutions together, including the power round cover sheet (the last page of this document) as the first page. Each page should also have on it the **team number** (not team name) and **problem number**. This team number, which was emailed to your team, is a "T" followed by a three-digit number (for example: "T100"). Solutions to problems may span multiple pages, but staple them in continuing order of proof.
2. You are encouraged, but not required, to use LaTeX to write your solutions. If you submit your power round electronically, **you may not submit multiple times**. The first version of the power round solutions that we receive from your team will be graded. If submitting electronically, **you must submit a PDF**. Any other file type will not be graded.
3. Do not include identifying information aside from your team number in your solutions.
4. Please collate the solutions in order in your solution packet. Each problem should start on a new page, and solutions should be written on one side of the paper only (there is a point deduction for not following this formatting).
5. On any problem, you may use without proof any result that is stated earlier in the test, as well as any problem from earlier in the test, even if it is a problem that your team has not solved. You may not cite parts of your proof of other problems: if you wish to use a lemma in multiple problems, please reproduce it in each one.
6. When a problem asks you to "show" or "prove" a result, a formal proof is expected. When a problem instead uses the word "explain," an informal explanation suffices.
7. All problems are numbered as "Problem x.y" where x is the section number. Each problem's point distribution can be found in parentheses before the problem statement.
8. **You may NOT use any references, such as books or electronic resources. You may NOT use computer programs, calculators, or any other computational aids.**
9. Teams whose members use English as a foreign language may use dictionaries for reference.
10. Communication with humans outside your team of 8 students about the content of these problems is prohibited.
11. There are two places where you may ask questions about the test. The first is Piazza. Please ask your coach for instructions to access our Piazza forum. On Piazza, you may ask any question **so long as it does not give away any part of your solution to any problem**. If you ask a question on Piazza, all other teams will be able to see it. If such a question reveals all or part of your solution to a power round question, your team's power round score will be penalized severely. For any questions you have that might reveal part of your solution, or if you are not sure if your question is appropriate for Piazza, please email us at pumac@math.princeton.edu. We will email coaches with important clarifications that are posted on Piazza.

Introduction and Advice

The topic of this power round is **encryption scheme security**. Many of the world's encryption systems are built on the theory that you will learn from this power round. Section 1 establishes some conventions. Sections 2 through 4 introduce basic concepts that you will need. In Sections 5 through 8, you will build a secure encryption scheme, based on some assumptions. Finally, in Section 9, you will build an encryption scheme that satisfies a much stronger definition of security.

Here is some advice with regard to the power round:

- **Read the text between the problems!** It will help you understand what's going on. In particular, some of the definitions in this power round are challenging to understand, and the text after these definitions tries to guide you to a better understanding of them.
- **Read the footnotes.** If you are confused that something doesn't quite follow, or have a question, there is a good chance that it is answered in a footnote.
- **Make sure you understand the definitions**, especially in the last few sections. If you don't, then you will not be able to do the problems. Feel free to ask clarifying questions about the definitions on Piazza (or email us).
- **Check Piazza often!** Clarifications will be posted there, and if you have a question it is possible that it has already been asked and answered in a Piazza thread (and if not, you can ask it, assuming it does not reveal any part of your solution to a question).
- Just to reiterate: **if in doubt about whether a question is appropriate for Piazza, please email us at pumac@math.princeton.edu.**
- A majority of the points are awarded for problems in the final three sections, even though they take up a relatively small fraction of the power round text. To help you plan how to spend your time on this power round, here is the point distribution by section:

Section	Points
Section 2	40
Section 3	35
Section 4	20
Section 5	30
Section 6	25
Section 7	60
Section 8	35
Section 9	105
Total	350

Good luck, and have fun!

-Eric Neyman.

We'd like to acknowledge and thank many individuals and organizations for their support; without their help, this power round (and the entire competition) could not exist. Please refer to the solution of the power round for full acknowledgments.

Contents

1 Notation and Preliminary Definitions

In this power round, we will use the following notations and definitions.

- $\mathbb{N}$ denotes $\{0, 1, 2, 3, \dots\}$.
- $\lg(n)$ denotes $\lceil \log_2(n+1) \rceil$, i.e. the number of digits in the binary representation of n.
- $f : S \to T$ means that f is a function whose domain is S and whose range is T. f is defined on every element of S, but it is not necessarily the case that for all $t \in T$ there is an element $s \in S$ such that $f(s) = t$.
- A *bit* is a binary digit (either 0 or 1); a *bit string* is a sequence of bits (for example, 100101). A generic bit string can be written as $x_1 x_2 \dots x_n$ (in the example, $n = 6$, $x_1 = 1$, $x_2 = 0$, $x_3 = 0$, and so on). The set of all bit strings is denoted $\{0,1\}^*$. The set of all bit strings of length n is denoted $\{0,1\}^n$. The length of a bit string x is denoted $|x|$. Given two bit strings x and y, $x \circ y$ denotes the concatenation of x and y. For example, $100 \circ 01101 = 10001101$. The bit string of n ones is denoted 1^n.
- $\Pr[\text{event}]$ denotes the probability of an event.
- For foreign students: the English alphabet goes as follows:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

2 Complexity

2.1 Big O Notation

Definition 2.1.1. For two functions $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$, we write $f(n) = O(g(n))$ if there exists $N \in \mathbb{N}$ and a positive real number c such that for all $n \geq N$, $f(n) \leq cg(n)$. This is called *big-O notation*.[1]

Example 2.1.2. If $f(n) = 2n^2 + 5n - 4$, then $f(n) = O(n^2)$, because there exists $N \in \mathbb{N}$ such that $f(n) \leq 3g(n)$ for all $n \geq N$. (Specifically we can take $N = 4$ or for that matter any larger N.) On the other hand, if $f(n) = n^3$ then $f(n) \neq O(n^2)$ because no matter what c and N you choose, it will not be the case that for all $n \geq N$, $f(n) \leq cn^2$.

Problem 2.1. (5 points) For each of the functions f and g below, determine whether $f(n) = O(g(n))$. Justify your answer only for part (d).

(a) (1 point) $f(n) = 2$ and $g(n) = 1$.

(b) (1 point) $f(n) = 2$ and $g(n) = n$.

(c) (1 point) $f(n) = n$ and $g(n) = 2$.

(d) (2 points) $f(n) = n^3$ and $g(n) = 2^n$.

Problem 2.2. (3 points) Show that if $f(n) = O(g(n))$ and $g(n) = O(h(n))$, then $f(n) = O(h(n))$.

[1] Note that this notation uses the equality sign differently from how it is used typically: $O(g(n))$ is not a specific function; rather, we can think of it as a collection of functions that grows at most as quickly as $g(n)$, and when we write "$f(n) = O(g(n))$" we are saying that f is in this collection.

2.2 Polynomial and Negligible Functions

Definition 2.2.1. Given functions $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$, $f(n)$ is said to be *polynomial in* $g(n)$ if there exists a positive integer k such that $f(n) = O((g(n))^k)$.

Problem 2.3. (5 points) For each of the functions f and g below, determine whether $f(n)$ is polynomial in $g(n)$. Justify your answer only for part (d).

(a) (1 point) $f(n) = 3n^2 - 3n + 1$ and $g(n) = n$.

(b) (1 point) $f(n) = \lfloor \sqrt{n} \rfloor$ and $g(n) = \lg(n)$.

(c) (1 point) $f(n) = n!$ and $g(n) = n$.

(d) (2 points) $f(n) = n^{\lg(n)}$ and $g(n) = n$.

Problem 2.4. (6 points)

(a) (3 points) Show that if $f(n)$ and $g(n)$ are polynomial in $h(n)$, then $f(n) + g(n)$ is polynomial in $h(n)$.

(b) (3 points) Show that if $f(n)$ and $g(n)$ are polynomial in $h(n)$, then $f(n) \cdot g(n)$ is polynomial in $h(n)$.

Definition 2.2.2. A function $f : \mathbb{N} \to \mathbb{R}$ is *negligible* if for every positive integer k, there exists $N_k \in \mathbb{N}$ such that for all $n \geq N_k$, $n^k \left| f(n) \right| < 1$.

Problem 2.5. (5 points) For each of the functions f below, determine whether $f(n)$ is negligible. Justify your answer only in parts (a) and (c).

(a) (2 points) $f(n) = \frac{1}{n^2}$

(b) (1 point) $f(n) = \frac{1}{2^n}$

(c) (2 points) $f(n) = \frac{1}{n^{\lg(n)}}$

Problem 2.6. (6 points)

(a) (3 points) Show that if $f(n)$ and $g(n)$ are negligible, then $f(n) + g(n)$ is negligible.

(b) (3 points) Show that if $f(n)$ and $g(n)$ are negligible, then $f(n) \cdot g(n)$ is negligible.

Problem 2.7. (5 points) Show that if $f : \mathbb{N} \to \mathbb{N}$ is polynomial in n and $g : \mathbb{N} \to \mathbb{R}$ is negligible, then $f \cdot g$ is negligible.

2.3 Polynomial-Time Algorithms

In computer science, big O notation is frequently used to express how fast a sequence of instructions (called an *algorithm*) will run in terms of the size of its inputs. This is called the *complexity* of the algorithm. An important concept is that of *polynomial time*. An algorithm runs in polynomial time if the number of steps it takes to execute the algorithm is a polynomial function of its input.

However, this is very vague: polynomial in *what*? Sometimes an algorithm $A(n)$ is said to run in polynomial time if the number of steps it takes is polynomial in n.[2] By this definition, an algorithm

[2] If A is an algorithm and is followed by something in parentheses — in this case, n — what is in the parentheses is the input to the algorithm A.

that adds up the numbers from 1 to n (where n is given to the algorithm as input) by starting with 0, then adding 1, then adding 2, and so on, up to n, runs in polynomial time.[3]

However, this is not the definition of polynomial time used in cryptography. In this power round, the definition of polynomial time that we will use is as follows.

Definition 2.3.1. Let A be an algorithm that takes n as input, and let $t(n)$ be the time it takes for A to run on input n (i.e. the number of operations that A performs if given input n). A runs in *polynomial time* if there exists $k \in \mathbb{N}$ such that $t(n) = O((\lg n)^k)$. In other words, A runs in polynomial time if $t(n)$ is polynomial in $\lg(n)$, i.e. the length of n.

Problem 2.8. (5 points)

(a) (3 points) Explain why the algorithm described above, that finds $1 + 2 + \cdots + n$ by adding up the numbers, is not a polynomial-time algorithm by Definition 2.3.1.

(b) (2 points) Describe, in words, an algorithm that finds $1+2+\cdots+n$ in polynomial time. (Hint: the algorithm is quite simple; don't overthink it!)

Polynomial-time algorithms are important because they are fast compared to algorithms that do not take polynomial time. For this reason, some of the most important problems in computer science center around which algorithms run in polynomial time and which do not. As we will discuss later in the power round, this concept is crucial for cryptography: the idea behind cryptography is that you want to encrypt things quickly (the encryption algorithm should take polynomial time), but decryption (the algorithm that determines the original message from the encrypted message) should have to be slow (not be a polynomial-time algorithm).

An example of a process that is thought to not be possible in polynomial time (though this is a hotly debated question among computer scientists) is that of factoring a positive integer n. There is no known algorithm that factors n in polynomial time in the length of n, and many computer scientists believe that such an algorithm does not exist. This is the basis of many encryption schemes, some of which you will be introduced to later.

In general, when people speak of polynomial-time algorithms, they mean *deterministic* polynomial-time algorithms, i.e. algorithms that do not make use of randomness and thus give the same output. However, sometimes it is useful to generalize the concept of polynomial-time algorithms to include *probabilistic* algorithms, i.e. ones that make use of randomness.

Definition 2.3.2. A *probabilistic polynomial-time algorithm* (henceforth *PPT algorithm*) is a generalization of the concept of polynomial-time algorithms that includes non-deterministic algorithms.

For example, an algorithm that takes an integer n as input, randomly chooses two integers between 1 and n, and outputs their sum is a PPT algorithm.[4] The algorithm that takes an integer n as input and outputs $2n$ is also a PPT algorithm: PPT algorithms aren't required to use randomness, so all polynomial-time algorithms are considered PPT algorithms.

[3] Specifically, this algorithm takes time $O(n \lg n)$. This is because adding a number with b bits takes b operations, on average the numbers from 1 to n have roughly $\lg(n)$ bits, and there are n numbers to be added. However, the precise complexity is not important for the purposes of this power round.

[4] Unless of course such an algorithm is implemented really inefficiently.

3 Some Probability Theory

Starting with Section 5, we'll be working with probabilities and PPT algorithms a lot. This section will introduce some of the concepts you'll need. Recall that Pr [event] denotes the probability of an event.

3.1 Probabilities Over Different Spaces

I rolled a fair six-sided die and looked at the result. Then I rolled another fair six-sided die and didn't look at the result. What is the probability that the sum of the rolls is 9?

And the answer is... well, it depends on whom you ask. Some people will tell you that the probability is either 0 or 1: either the sum is 9 or it isn't. But let's leave this view aside, since it doesn't tell us anything interesting.

Another possible answer to the question is $\frac{1}{9}$. This is because out of the 36 possible pairs of rolls, four of them ($(3,6)$, $(4,5)$, $(5,4)$, and $(6,3)$) give a sum of 9. This makes sense from your perspective, and from the perspective of anyone who hasn't seen the first die. Based only on the information you have, there's a one-in-nine chance that the sum of the two dice is 9.

A third possible answer is that it depends on the first die's roll. If the first roll was 1 or 2 then the probability is 0, and if it was 3, 4, 5, or 6 then the probability is $\frac{1}{6}$. And indeed, if you asked me what the probability that the sum is 9 is, I would tell you either 0 or $\frac{1}{6}$ (depending on what I observed on the first die).

The distinction here is a matter of probabilities over different spaces: in the former case, the probability is taken over the space of all possible pairs of die rolls; in the latter case, the probability is taken over the space of all possible rolls of the second die, with the roll of the first die already known.

Problem 3.1. (4 points) You roll two fair dice. For each statement below, indicate whether it is true or false. Justify your answer only for part (b).

(a) (1 point) Over the space of all possible rolls of the two dice, the probability that the sum of the rolls is at least 9 is less than $\frac{1}{2}$.

(b) (2 points) For all possible rolls of the first die, over the space of all possible rolls of the second die, the probability that the sum of the rolls is at least 9 is less than $\frac{1}{2}$.

(c) (1 point) There exists a possible roll of the first die such that, over the space of all possible rolls of the second die, the probability that the sum of the rolls is at least 9 is less than $\frac{1}{2}$.

In this power round, wherever we believe that the probability space is not clear, we state the probability space explicitly (for example by saying "the probability is over the space of all possible rolls of two dice" or something to that effect). If you are ever unclear on what the probability space is, feel free to ask on Piazza.

3.2 Random Variables and Bounds on Probabilities

Let's say you have a probability space (e.g. the space of all possible pairs of rolls of six-sided dice). The elements of a probability space are called *events* (so here an example of an event is $(2,4)$, indicating that the first roll was a 2 and the second roll was a 4). A *random variable* is a number assigned to an event.[5] For example, one random variable for our example probability space is the sum of the two rolls.

In this subsection, we will only work with *discrete* probability spaces, i.e. probability spaces with a finite number of events. Similarly, all random variables will be discrete, i.e. will take on only

[5] More formally, it is a function from the probability space to $\mathbb{R}$.

finitely many possible values. For every problem in this subsection, you may assume that we are working over a discrete probability space and discrete random variables.

Definition 3.2.1. Random variables X_1, X_2, X_n are *independent* if for all $x_1, x_2, \ldots, x_n$,

$$\Pr[X_1 = x_1 \text{ and } X_2 = x_2 \text{ and } \ldots \text{ and } X_n = x_n] = \prod_{i=1}^{n} \Pr[X_i = x_i].$$

In our example, if we define X_1 to be the value of the first roll and X_2 to be the value of the second roll, then X_1 and X_2 are independent. However, if we let X_1 be the value of the first roll and X_2 be the sum of the two rolls, then X_1 and X_2 are not independent.

Definition 3.2.2. Random variables $X_1, X_2, \ldots, X_n$ are *pairwise independent* if for all $1 \leq i \neq j \leq n$, X_i and X_j are independent.

Problem 3.2. (8 points)

(a) (5 points) Show that if $X_1, X_2, \ldots, X_n$ are independent, then they are pairwise independent.

(b) (3 points) Let w be a bit string randomly selected from $\{000, 011, 101, 110\}$. For $1 \leq i \leq 3$, let X_i be the random variable equal to the i-th bit of w. Are X_1, X_2, and X_3 pairwise independent? Are X_1, X_2, and X_3 independent? Explain.

Definition 3.2.3. The *expected value* of a random variable X, denoted $\mathbb{E}(x)$, is $\sum_x x\Pr[X = x]$.

For example, the expected value of the roll of a fair six-sided die is 3.5. This is because

$$1 \cdot \frac{1}{6} + 2 \cdot \frac{1}{6} + 3 \cdot \frac{1}{6} + 4 \cdot \frac{1}{6} + 5 \cdot \frac{1}{6} + 6 \cdot \frac{1}{6} = \frac{7}{2}.$$

Here we are taking all possible rolls, multiplying them by their probabilities (always $\frac{1}{6}$), and adding up the results. The term "expected value" makes sense: the expected value of a random variable is what you would expect, on average, the value of the random variable to be.[6]

A very useful theorem about expected values is the theorem of the linearity of expectation.

Theorem 3.2.4 (Linearity of expectation)**.** *Let $X_1, X_2, \ldots, X_n$ be random variables over a probability space. Then $\mathbb{E}(X_1 + X_2 + \cdots + X_n) = \mathbb{E}(X_1) + \mathbb{E}(X_2) + \cdots + \mathbb{E}(X_n)$. This is true even if $X_1, \ldots, X_n$ are not independent.*

Example 3.2.5. What is the expected number of aces among the first three cards of a 52-card deck?[7] One way to do this problem is this: the probability that the first two cards are aces is $\frac{4}{52} \cdot \frac{3}{51}$. The probability that the first two cards are both not aces is $\frac{48}{52} \cdot \frac{47}{51}$. The probability that one of the cards is an ace is $1 - \frac{4}{52} \cdot \frac{3}{51} - \frac{48}{52} \cdot \frac{47}{51}$. Then, conditional on these three cases we can calculate the probability that the third card is an ace and, after a mess of calculations, get the answer. Or, we can let X_1 be the random variable that is 1 if the first card is an ace and 0 otherwise, X_2 be 1 if the second card is an ace and 0 otherwise, and X_3 be 1 if the third card is an ace and 0 otherwise. Then by linearity of expectation, the expected number of aces among the first three cards is

$$\mathbb{E}(X_1 + X_2 + X_3) = \mathbb{E}(X_1) + \mathbb{E}(X_2) + \mathbb{E}(X_3) = 3 \cdot \frac{4}{52} = \frac{3}{13}.$$

We now discuss some bounds that one can place on probabilities of events.

[6] The expected value need not be an achievable value, as our example shows (you can't actually get 3.5 as an outcome when you roll a die).

[7] There are four aces in a 52-card deck.

Theorem 3.2.6 (Markov's inequality)**.** *If X is a random variable such that all possible values of X are nonnegative, and a is a positive real number, then $Pr[X \geq a] \leq \frac{\mathbb{E}(X)}{a}$.*

Problem 3.3. (3 points) Prove Markov's inequality.

Markov's inequality is rather weak, but you can't really say anything stronger unless you take into account a notion of how close to $\mathbb{E}(X)$ the distribution of possible values of X is. In other words, if the distribution is narrow, you should intuitively be able to make a stronger statement than Markov's inequality. For this purpose we introduce the purpose of *variance*.

Definition 3.2.7. Given a random variable X, the *variance* of X is

$$V(X) = \mathbb{E}((X - \mathbb{E}(X))^2).$$

This definition makes sense: $\mathbb{E}(X)$ is what you'd expect X to be on average, so $(X - \mathbb{E}(X))^2$ is a measure how much X deviates from its expected value on average. The problem below is a useful property of variances.

Problem 3.4. (8 points) Prove that if $X_1, X_2, \ldots, X_n$ are pairwise independent random variables, then $V(\sum_i X_i) = \sum_i V(X_i)$.

Theorem 3.2.8 (Chebyshev's inequality)**.** *Let X be a random variable. Then for all $t > 0$,*

$$Pr[|X - \mathbb{E}(X)| \geq t] \leq \frac{V(X)}{t^2}.$$

Problem 3.5. (7 points) Prove Chebyshev's inequality.

Problem 3.6. (5 points) 1000 fair coins are flipped. Give an upper bound on the probability that 600 or more coins land heads-up:

(a) (2 points) Use Markov's inequality.

(b) (3 points) Use Chebyshev's inequality.

4 Introduction to Cryptography

4.1 Definition of an Encryption Scheme

Suppose that Alice has a message that she wants to deliver to Bob, so that Bob can read it but no one else can. Alice can send her message by mail to Bob, but this has an obvious problem: Eve, a worker at the post office, could open the letter and read it. So Alice decides to put her message in a locked box and send it to Bob. But this is also problematic: how can Bob open the box? Well, if Alice and Bob meet in advance of the message being sent, Alice can give Bob a key with which to open the box. Essentially, Alice and Bob share a *private key* with which they can communicate: Alice can put a message in a box, lock it with the key, and send it to Bob, who can then open the box with the key.

While this is a secure system[8], it is cumbersome: every message has to be sent inside a box. It would be nice if Alice could send her message in an envelope, so that even if Eve opens the envelope, she can't understand what Alice wrote. This idea is the basis of encryption, and just as a physical key was crucial to Alice and Bob's plan in the previous paragraph, Alice can now use a figurative key to keep the message secret from Eve. Essentially, a (secret key) encryption scheme is an algorithm

[8] We're using "secure" in an informal way until we define it in the next section. If we call a system secure, it means that it's either impossible or really hard for Eve to decipher the message.

that encrypts a message using a key, together with an algorithm that can decipher the message using the key. Thus, Alice can share a key with Bob when they meet, and then send an encrypted message to Bob that Bob can decipher using the key. If the encryption scheme is secure (which will be defined rigorously later), Eve won't be able to decipher the message.

Definition 4.1.1. An *encryption scheme* is an ordered triple of algorithms (G, E, D), where G and E are PPT algorithms and D is a (deterministic) polynomial-time algorithm, such that for every positive integer n (n is called the *security parameter*):

- G, the *key generation algorithm*, takes as input 1^n, the string of n ones (for some n), and uses randomness to choose a *key* k from a set of possible keys K_n, called the *key space*.[9]
- E, the *encryption algorithm*, takes as input a message $m \in P_n$ (P_n is called the *plaintext space* for the security parameter n), and, using k, outputs an encrypted message $c \in C_n$ (C_n is called the *ciphertext space* for the security parameter n). E might also use randomness (in the sense that $E_k(m)$ may not always output the same ciphertext c). If E uses the key k to encrypt the message m and outputs c, we write $E_k(m) = c$.
- D, the *decryption algorithm*, takes as input an encrypted message and uses k to decipher it. D is deterministic (does not use randomness). Thus, D_k, the decryption algorithm given the key k, is a function from the ciphertext space to the plaintext space. This means that for all $m \in P$, we have $D_k(E_k(m)) = m$.[10]

This definition has a lot packed into it. The next definition should help you gain some intuition for it.

4.2 Examples of Encryption Schemes

The following examples are intended to help you gain intuition for Definition 4.1.1.

4.2.1 Caesar Cipher

The *Caesar cipher* takes a message written using the English alphabet (which has 26 letters) and performs a cyclic shift of a randomly chosen, pre-determined amount on all of the letters. So if Alice and Bob want to communicate using the Caesar cipher, Alice and Bob meet and agree on a shift (let's say they agree on 5). Later, if Alice wants to send Bob the word "YOGURT", she will replace each letter with the letter that is 5 after it, looping around the alphabet if necessary. Thus, Alice sends Bob "DTLZWY". Then Bob can decipher the message by "subtracting 5" from every letter. (Note that, as in this example, when we specify a shift it is always in the forward direction.)

Problem 4.1. (4 points)

(a) (1 point) If Alice wants to send Bob the word "MATH" with shift 10, what will the encrypted message be? (No explanation is necessary.)

(b) (1 point) Suppose that Bob receives "XCUIK" from Alice, and the agreed-upon shift was 8. What was Alice's message? (No explanation is necessary.)

(c) (2 points) Eve, the post office worker, intercepts a message from Alice to Bob that was encrypted using the Caesar cipher. The message says:

FKQBDOXQFLKYVPJXOQP

[9] What's the point of taking 1^n as input? Since G is a polynomial-time algorithm, this allows G to take a polynomial number of steps in n, the length of its input.

[10] You may assume that E and D both "know" the security parameter n in the same way that they know k.

What was the original message, and what was the shift amount? (The message is a three-word English phrase with spaces omitted. No explanation is necessary.)

Hopefully that problem wasn't too difficult. But if you were able to do the last part of the problem, this says something interesting: the Caesar cipher is not secure! Eve can decipher Alice's message without knowing the key beforehand. The question of encryption-scheme security will be central to this power round.

The Caesar cipher can be stated in terms of Definition 4.1.1. For a positive integer n, we let the plaintext space P_n be the set of all strings of n English letters. The key space is $K_n = \{0, 1, 2, \ldots, 25\}$ and the ciphertext space C_n is also the set of all strings of n English letters. The key generation algorithm G simply picks $k \in K_n$ uniformly at random. The encryption algorithm E_k takes each letter and replaces it with the letter k later than it in the alphabet (looping around if necessary). The decryption algorithm D_k takes each letter and replaces it with the letter k earlier than it in the alphabet.[11]

4.2.2 One-Time Pad

Another encryption scheme is the one-time pad. The one-time pad encrypts a bit string into another bit string, as opposed to the Caesar cipher, which encrypts a string of English letters into another string of English letters. The one-time pad makes use of the "bitwise exclusive or" operation.

Definition 4.2.1. Given two bits b_1 and b_2, the *exclusive or* (often abbreviated *xor*) of b_1 and b_2, written as $b_1 \oplus b_2$, is equal to 0 if $b_1 = b_2$ and 1 if $b_1 \neq b_2$. In other words, $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, and $1 \oplus 1 = 0$.

Definition 4.2.2. Given two bit strings $x = x_1x_2 \ldots x_n$ and $y = y_1y_2 \ldots y_n$ of the same length, the *bitwise exclusive or* (often abbreviated *bitwise xor* or simply *xor*) of x and y, written as $x \oplus y$, is the bit string $z_1z_2 \ldots z_n$, where $z_i = x_i \oplus y_i$ for $1 \leq i \leq n$.

Problem 4.2. (2 poitns)

(a) (1 point) What is $100111 \oplus 010010$? (No explanation is necessary.)

(b) (1 point) If $x \oplus 110101 = 100111$, what is x? (No explanation is necessary.)

Problem 4.3. (7 points)

(a) (1 point) For any x, find $x \oplus x$. (No explanation is necessary.)

(b) (1 point) For any x, find $x \oplus 0$, where 0 is the appropriate-length bit string of all zeros. (No explanation is necessary.)

(c) (1 point) Is it true that $x \oplus y = y \oplus x$? (No explanation is necessary.)

(d) (1 point) Is it true that $(x \oplus y) \oplus z = x \oplus (y \oplus z)$? Explain.

(e) (1 point) Prove that if $x \oplus y = z$, then $x \oplus z = y$

(f) (2 points) Prove that $x \oplus y = x \oplus z$ if and only if $y = z$.

The *one-time pad cipher* works as follows. Suppose that Alice then wants to send a message m of some length ℓ to Bob. Alice and Bob agree on a uniformly randomly chosen bit string k of length ℓ that serves as the key. She sends $m \oplus k$ to Bob.

[11] There are multiple ways of defining the Caesar cipher in terms of our definition of an encryption scheme, but this is the most natural one.

Problem 4.4. (2 points) Suppose that Bob receives a ciphertext message c from Alice. How can Bob use k to retrieve the original message m? Explain.

Problem 4.5. (5 points) Characterize the one-time pad cipher using Definition 4.1.1. In particular, if for a given security parameter n the plaintext space is $\{0,1\}^n$ (the set of length-n bit strings), what are the key space K_n and ciphertext space C_n? Briefly describe, in words, the key generation algorithm G, the encryption algorithm E, and the decryption algorithm D.

In practice, Alice and Bob would exchange a very long key (of, say, a million bits). When Alice sends a message of length ℓ to Bob, she uses the first ℓ bits of the key that she had not previously used. Unlike the Caesar cipher, the one-time pad cipher is in fact perfectly secure. The issue with it is that the one-time pad cipher requires Alice and Bob to exchange a very long key: for every bit of communication between Alice and Bob, there must be a corresponding bit of the key that was exchanged between Alice and Bob originally. For this reason, the one-time pad is not used much in practice.

5 Security of Encryption Schemes

5.1 Indistinguishability of Encryptions

We are now ready to define a notion of security for encryption schemes. There are many notions of encryption scheme security; this one is called *statistical indistinguishability of encryptions*. Note that for convenience, from now on we will only consider encryption schemes whose plaintext and ciphertext spaces are subsets of $\{0,1\}^*$, the set of all bit strings.

Definition 5.1.1. An encryption scheme (G, E, D) satisfies *computational indistinguishability of encryptions* if there exists a negligible function $\text{neg}(n)$ such that for all n, for all messages $m_0, m_1 \in P_n$, for every PPT algorithm A,

$$|\Pr[A(E_k(m_0)) = 1] - \Pr[A(E_k(m_1)) = 1]| \leq \text{neg}(n),$$

where the probability is over the randomness of $G(1^n)$ (i.e. the randomness of they key), the randomness of E, and the randomness of A.

Let's try to make sense of this definition. The algorithm A, often called an *adversary*[12] (because it attempts to break the encryption), outputs 1 with a certain probability when given a ciphertext. (This probability might always be 0 or 1, but it doesn't have to be, since A is not required to be deterministic.) The goal of A is to be able to distinguish between the encryptions of different messages: it wants to output 1 more often when given the encryption of one message than when given the encryption of another message. If A can do this successfully, then A can potentially give a hacker or spy useful information about the original message, and so the encryption scheme is insecure. If, on the other hand, for every pair of messages there is no adversary A that can distinguish effectively between their encryptions, then the encryption scheme satisfies computational indistinguishability of encryptions, one notion of security.

Problem 5.1. (3 points) Your teammate comments, "This definition of security is dumb. Consider the encryption scheme where $E_k(m) = 0$ regardless of k and m. Then clearly A can't distinguish between the encryptions of two messages m_1 and m_2, since their encryptions are the same." What mistake is your teammate making? (Hint: examine Definition 4.1.1 closely.)

Let us entertain another definition.

[12] The word "adversary" means "opponent."

Definition 5.1.2. An encryption scheme (G, E, D) satisfies *guessing indistinguishability of encryptions* if there exists a negligible function $\text{neg}(n)$ such that for all n, any PPT algorithm A wins the following game with probability at most $\frac{1}{2} + \text{neg}(n)$:

1) A picks $m_0, m_1 \in P_n$.

2) A "referee" generates $k = G(1^n)$, uniformly randomly chooses $b \in \{0, 1\}$, and gives $E_k(m_b)$ to A.

3) A outputs[13] $b' \in \{0, 1\}$.

4) A wins if $b = b'$.

This seems to have some similarities with the previous definition. Here, as well, the goal of A is to figure out whether it is given $E_k(m_0)$ or $E_k(m_1)$. In fact, the two definitions are identical.

Problem 5.2. (14 points) Prove that an encryption scheme (G, E, D) satisfies computational indistinguishability of encryptions if and only if it satisfies guessing indistinguishability of encryptions.

For the next few sections, this notion of indistinguishability of encryptions will be our only notion of cryptographic security. For this reason, in the next few sections we will simply say that an encryption scheme is *secure* if it satisfies computational indistinguishability of encryptions.

Problem 5.3. (7 points) Prove that the one-time pad cipher (formalized in Problem 4.5) is secure.

Problem 5.4. (6 points)

(a) (4 points) Recall that $x \circ y$ denotes the concatenation of x and y (see Section 1). Consider the following encryption scheme (G, E, D) with $P_n = \{0, 1\}^{8n}$. G generates a key $k \in \{0, 1\}^8$ uniformly at random. E encrypts a message $m \in P_n$ by breaking m into 8-bit chunks $m_1, m_2, \ldots, m_n$ and outputting $(m_1 \oplus k) \circ (m_2 \oplus k) \circ \cdots \circ (m_n \oplus k)$. D decrypts a ciphertext c by breaking c into 8-bit chunks $c_1, c_2, \ldots, c_n$ and outputting $(c_1 \oplus k) \circ (c_2 \oplus k) \circ \cdots \circ (c_n \oplus k)$. Show that this encryption scheme is insecure.

(b) (2 points) Explain why this means that the one-time pad is insecure if the same key is used on multiple messages.[14]

6 Pseudorandom Generators

6.1 What is a Pseudorandom Generator?

We've discussed why the one-time pad isn't used much in practice: it is unreasonable to exchange a key that is as long as the message itself (or, in the case of multiple messages, as long as all of the messages put together). In practice, we want our key to be much shorter than the message.

So here's the idea: the one-time pad works by performing a binary xor of the message with as many randomly chosen bits as are in the message. So let's say our key is in fact a lot shorter than the message. We're going to give our key as input to some complicated algorithm that will output a longer string than the key itself. But the algorithm is so complicated that the output cannot be realistically distinguished from random bits. If we can accomplish this, then we can simulate a one-time pad and produce a secure encryption scheme.

Such an algorithm is called a *pseudorandom generator*[15] (PRG) and is crucial in creating a secure but feasible encryption scheme. We explore PRGs in this section.

[13] It is best to think of this step as a separate algorithm run by A with inputs m_0, m_1, and $E_k(m_b)$.

[14] We haven't defined security on multiple messages, so your justification need not be rigorous.

[15] "Pseudo" is a root word meaning "fake." This makes sense, because a pseudorandom generator is deterministic, and so it produces an output that looks random but isn't.

Definition 6.1.1. $H : \{0,1\}^* \to \{0,1\}^*$ is a *pseudorandom generator (PRG)* if:

- H is a polynomial-time algorithm.[16]
- There exists a function $\ell : \mathbb{N} \to \mathbb{N}$, called the *length extension function* of H, such that $\ell(n) > n$ for all n and $|H(x)| = \ell(|x|)$ for all $x \in \{0,1\}^*$. That is, the output of H is longer than the input, and all inputs of the same length have the same output length.[17]
- There exists a negligible function $\text{neg}(n)$ such that for all n, for every PPT algorithm D,

$$\left|\Pr\left[D(H(U_n)) = 1\right] - \Pr\left[D(U_{\ell(n)}) = 1\right]\right| < \text{neg}(n),$$

  where U_k denotes a uniformly randomly chosen k-bit string.

Again, let's try to make sense of this definition. The first idea is that H is an algorithm that takes an input of length n and efficiently and deterministically creates a longer output of length $\ell(n)$. But this is easy: H could, for instance, just add a 0 to the end of its input. The crucial component of the definition is thus the third component: there is no efficient way to distinguish the output of H on a random input of length n from a random string of length $\ell(n)$. No matter what PPT algorithm D you choose, the probability that it will return 1 given $H(x)$ for a random x of length n will be very close to the probability that it will return 1 given a random bit string of length $\ell(n)$.

Problem 6.1. (9 points) Show that the following are not pseudorandom generators.

(a) (2 points) $H(x) = x \circ 0$.

(b) (2 points) $H(x) = x \circ x$.

(c) (2 points) $H(x)$ takes each bit of x and flips it, outputting the result.

(d) (3 points) For $x = x_1 x_2 \dots x_n$, $H(x) = x_1 \circ (x_1 \oplus x_2) \circ (x_2 \oplus x_3) \circ \cdots \circ (x_{n-1} \oplus x_n) \circ (x_n \oplus x_1)$.

6.2 Constructing Secure Encryption Schemes from PRGs

As we informally discussed earlier, secure encryption schemes can be built out of PRGs.

Problem 6.2. (16 points) Let H be a pseudorandom generator with length extension function $\ell(n)$. Define an encryption scheme (G, E, D) with $P_n = C_n = \{0,1\}^{\ell(n)}$ as follows:

- $G(1^n)$ chooses an n-bit string k uniformly at random.
- $E_k(m) = m \oplus H(k)$.
- $D_k(c) = c \oplus H(k)$.

Prove that (G, E, D) is a secure encryption scheme. As part of your solution, be sure to verify that (G, E, D) satisfies the definition of an encryption scheme.

The previous problem shows how to construct a secure encryption scheme from a PRG. Note that the encryption scheme (G, E, D) that you proved to be secure in the previous problem is very similar to the one-time pad. However, keys of length n are used to encrypt messages of length $\ell(n)$ instead of messages of length n, an improvement on the main problem of the one-time pad (the necessity of long keys).

But do PRGs even exist? (So far we've only seen *non-examples* of PRGs, in Problem 6.1.) And if they exist, how can we construct one? We address this question in the next two sections.

[16] Recall that this means that H must be a deterministic algorithm.

[17] Recall that for a bit string x, $|x|$ denotes the length of x.

7 One-Way Functions

7.1 What is a One-Way Function?

In Section 8, we will build PRGs out of functions called one-way permutations, which are a special class of *one-way functions*. A one-way function is basically a function f with the property that given x, it is easy to find $f(x)$, but given $f(x)$, it is difficult to find x. More formally:

Definition 7.1.1. $f : \{0,1\}^* \to \{0,1\}^*$, is a *one-way function (OWF)* if f can be evaluated in polynomial time, but there is a negligible function $\text{neg}(n)$ such that for all n, for every PPT algorithm A,

$$\Pr\left[f(A(f(x), 1^n)) = f(x)\right] \leq \text{neg}(n),$$

where the probability is over x randomly chosen from $\{0,1\}^n$ and the randomness of A.

Intuitively, a one-way function is a function that is efficiently computed but not efficiently reverted. Think of A as an adversary that attempts to invert f. The adversary A takes as input $f(x)$ for some x that it does not know; its goal is to output some x' such that $f(x') = f(x)$. A function is a one-way function if A cannot do this with a non-negligible probability as a function of n. (The input 1^n to A tells A the length of the input string and also ensures that the input to A has length at least n, allowing A to do anything that is polynomial time in n.)

In this section, whenever we talk about specific one-way functions, we will not be using the full rigor of the above definition.[18] Some of our "one-way functions" will have their domain be only a subset of binary strings, or something else entirely (e.g. ordered pairs of primes). Our definition still makes sense, though, because it is meant to communicate two things: that one-way functions are efficiently computed but not efficiently inverted. What do we mean by "efficiently"? Well, the concept of polynomial time still applies to inputs that are not binary strings. As an example, an ordered pair of primes (p, q) can be encoded (very roughly speaking) as the binary representation of p followed by the binary representation of q, and the length of the resulting string is on the order of $\lg(p) + \lg(q)$. We hope that this intuition is sufficient for understanding and solving problems about specific one-way functions in this section, despite our less-than-rigorous treatment of the matter.

Problem 7.1. (10 points) Show that the following are not one-way functions.

(a) (3 points) $f(x)$ outputs the sum of the bits of x.

(b) (3 points) $f(x) = x^2$ (the binary string x is treated as the corresponding nonnegative integer).

(c) (4 points) f takes as input (p, k), where p is prime and $1 \leq k < p$ is an integer, and outputs $(p, k^{-1} \pmod p)$. (k^{-1} is the number m modulo p such that $km \equiv 1 \pmod p$.)

7.2 Number-Theoretic One-Way Functions and Assumptions

This section doesn't have any problems, but the information here will hopefully ground what we've been talking about using concrete examples.

It turns out that we don't know whether one-way functions exist. However, we have many candidates for one-way functions. Many of these candidates are built on certain observations about number theory. The first such observation is known as the *factoring assumption*, which is that the function f that takes two primes p and q as input and outputs pq is a one-way function.

Why is this called the factoring assumption? Think about what an adversary A would need to do, if given an integer N as input: it needs to output something that, if given to f as input, outputs N; that is, given $N = pq$, it needs to output (p, q).

[18] We will be rigorous when talking about one-way functions in general, though, and your solutions to problems about general one-way functions rather than specific ones should be fully rigorous.

We don't know whether factoring a product of two primes can be done in polynomial time. However, no polynomial-time algorithm for factoring numbers has been found, and many computer scientists believe that no such algorithm can exist.

Another candidate family of one-way functions is built on an assumption called the *discrete log assumption*. Under this assumption, for p prime and g a *generator* of p, meaning that $p-1$ is the smallest positive exponent to which you can raise g to get 1 modulo p, the function $f_{p,g}(x) = g^x \pmod{p}$ is a one-way function. Computing g^x modulo p can be done pretty efficiently, but how would you reverse the function — how would you find the number x such that $85^x \equiv 48 \pmod{127}$? In fact, computer scientists don't know how to do such calculations efficiently and many believe that such a calculation can't be done efficiently.

It turns out, though, that we can efficiently compute x such that $x^{85} \equiv 48 \pmod{127}$. To do this, we would find $85^{-1} \pmod{126}$ (which can be done in polynomial time); it happens to be 43. So we raise both sides to the power 43, which gives us

$$56 \equiv 48^{43} \equiv (x^{85})^{43} \equiv x^{85 \cdot 43} \equiv x \pmod{127},$$

which gives us $x = 56$. (The last step above was by Fermat's little theorem, which tells us that $x^{126} \equiv 1 \pmod{127}$, so for some k, $x^{85 \cdot 43} = x^{126k+1} \equiv (x^{126})^k \cdot x \equiv x \pmod{127}$.)

However, for integers c and e, in general we don't know how to efficiently find x such that $x^e \equiv c \pmod{N}$ where N is composite. In other words, under what is known as the *RSA assumption*, the function $f_{N,e}(x) = x^e \pmod{N}$, is a one-way function.

7.3 Hard-Core Bits

One-way functions seem like natural tools for encryption. After all, they take a bit string and perform an operation on it after which the original bit string cannot be efficiently recovered. The problem is that there needs to be *some* way to recover the initial bit string; otherwise the person for whom the message was intended cannot decrypt the message. Unfortunately, for this reason there is no known way of using a one-way function this directly to encrypt information. To build an encryption scheme from a one-way function, we need to build a little more theory.

In this section you will prove a theorem that will be crucial to our goal of creating a pseudorandom generator from a one-way function. We first define the following concept.

Definition 7.3.1. Given a one-way function f, the function $b : \{0,1\}^* \to \{0,1\}$ is a *hard-core bit* for f if b is computable in polynomial time and there is a negligible function $\text{neg}(n)$ such that for all n, for every PPT algorithm A,

$$\Pr\left[A(f(x), 1^n) = b(x)\right] \leq \frac{1}{2} + \text{neg}(n),$$

where the probability is over x selected uniformly at random from $\{0,1\}^n$ and the randomness of A.

Intuitively, $b(x)$ is a bit dependent on x that can be computed efficiently given x. However, if you are only given $f(x)$, it is impossible to efficiently make a good guess for what $b(x)$ is (let alone to determine $b(x)$).

Does every one-way function have a hard-core bit? An important theorem answers a related question in the affirmative.

Theorem 7.3.2. *Let f be a one-way function. Define $g : \{0,1\}^{2n} \to \{0,1\}^*$ as $g(x \circ y) = f(x) \circ y$, where $|x| = |y| = n$. Then g is a one-way function with hard-core bit*

$$b(x,y) = \bigoplus_{i=1}^{n} x_i \wedge y_i = \sum_{i=1}^{n} x_i y_i \pmod{2}.$$

In Theorem 7.3.2, note that the domain $\{0,1\}^{2n}$ is the set of all bit strings of even length. If you've been paying close attention to our definitions, you may note that g actually isn't a one-way function the way we've defined it, since its domain is only the set of even-length strings. In practice, it is okay for a one-way function's domain to not be the set of all bit strings. Note that Definition 7.1.1 makes sense for g anyway, if "for all n" is taken to mean "for all even n."

Problem 7.2. (45 points) In this problem, you will prove Theorem 7.3.2.

(a) (7 points) Prove that g is a one-way function (with respect to the modified definition given in the above paragraph).

(b) (5 points) Use what you learned in Section 3.2 to prove the following: if $X_1, X_2, \ldots, X_m$ are pairwise independent random variables that are each 1 with probability μ and 0 otherwise, then

$$\Pr\left[\left|\frac{\sum X_i}{m} - \mu\right| \geq \epsilon\right] \leq \frac{\mu(1-\mu)}{m\epsilon^2}.$$

(c) (6 points) Proceed by contradiction. Assume for contradiction that $b(x,y)$ (we will simply write $b(x,y)$) is not a hard-core bit of g. Then there is an algorithm A that, given $f(x)$ and y, computes $b(x,y)$ with probability $\frac{1}{2}+\epsilon$, where ϵ is a non-negligible function of n.[19] Consider the following magical algorithm B that inverts f:

1. For a pre-determined positive integer m, B magically randomly generates strings $r_1, r_2, \ldots r_m$ of length n so that the values of the strings are uniform random variables that are pairwise independent, and magically computes $b_j = b(x, r_j)$ for $1 \leq j \leq m$.
2. For i from 1 to n, where n is the length of x:
   i. For each j from 1 to m, let $g_j = b_j \oplus A(f(x), e_i \oplus r_j)$, where $e_i = 00\ldots010\ldots00$, where the 1 is in the i-th position (e.g. $e_1 = 100\ldots0$; all of the e_i have length n).
   ii. Let t_i be the more common bit among $g_1, \ldots, g_m$.[20]
3. B returns $t_1 t_2 \ldots t_n$.

Explain why, if A always outputs $b(x,y)$ correctly on input $(f(x), y)$, then B will always return x correctly, thus inverting f.

(d) (7 points) But A is not perfect; in fact, it only gives the correct output with probability $\frac{1}{2}+\epsilon$. Let

$$S = \left\{x \in \{0,1\}^n \mid \Pr\left[A(f(x), r) = b(x,r)\right] > \frac{1}{2} + \frac{\epsilon}{2}\right\}$$

where the probability is over r chosen uniformly at random from $\{0,1\}^n$ and the randomness of A. Show that $|S| \geq 2^n \cdot \frac{\epsilon}{2}$. Using Part (b), show that if $x \in S$ and $m \geq \frac{2n}{\epsilon^2}$, then B returns x correctly with a non-negligible probability.

(e) (15 points) But we're not done, because B uses magic in its first step. Figure out how to modify the first step of B so that it doesn't use magic. (Hint: B should first generate $\lg(m)$ strings $s_1, s_2, \ldots, s_{\lg(m)}$ and let the strings r_j be the bitwise binary xors of various subsets of these strings. It should then somehow guess the bits $b_1, b_2, \ldots, b_m$ so that it gets all of the bits right with a non-negligible probability.)

(f) (5 points) Conclude by explaining why the algorithm B that we have constructed is a PPT algorithm that inverts f with a non-negligible probability. This finishes our proof by contradiction, since it contradicts the initial assumption that f is a one-way function.

[19] This isn't quite obvious, but you should figure out why this follows with a bit of thought.

[20] If half of the g_j are zero and half are one we may select t_i to be zero or one at random.

For use in the following section, we will need to define a special kind of one-way function, called a *one-way permutation*. For this definition, recall that a function $f : S \to T$ is *bijective* if for all $y \in T$ there is exactly one $x \in S$ such that $f(x) = y$.

Definition 7.3.3. A *one-way permutation* is a bijective one-way function f such that $f(x)$ has length n for all x of length n.

Problem 7.3. (5 points) Show that if f in Theorem 7.3.2 is a one-way permutation, then g in Theorem 7.3.2 is a one-way permutation. Explain why this means that Theorem 7.3.2 still holds if "function" is replaced by "permutation" in the theorem statement.

8 Constructing Pseudorandom Generators

8.1 Overview

In Section 6.2, you showed how to construct a secure encryption scheme from a pseudorandom generator. The following theorem is therefore sufficient to prove that if one-way functions exist, then secure encryptions exist.

Theorem 8.1.1. *If one-way functions exist, then pseudorandom generators exist.*

Unfortunately, this theorem is very difficult to prove and is beyond the scope of this power round.

8.2 Constructing a PRG From a One-Way Permutation

Instead, you will prove by construction that if one-way permutations exist, then pseudorandom generators exist.

Problem 8.1. (15 points) Let f be a one-way permutation and b be a hard-core bit for f. Let $H(x) = f(x) \circ b(x)$. Prove that $H(x)$ is a pseudorandom generator. Explain why this means that if one-way permutations exist, then pseudorandom generators exist. (Note: you may not use Theorem 8.1.1 to solve this problem. Also, for the second part, it is sufficient to explain why, if one-way functions exist then pseudorandom generators with (reasonable) input-length limitations exist — that is, functions that are pseudorandom generators modulo the same caveat about g not quite being a one-way function (see text below Theorem 7.3.2).)

8.3 Extending the Stretch of a PRG

The construction in the previous section has a limitation: it creates a PRG with a length extension function $\ell(n) = n + 1$, and by the construction in Section 6.2, a secure encryption scheme whose key length is only one bit shorter than the message length. This isn't much better than the one-time pad! Let's see if we can make a pseudorandom generator with a more impressive length extension function (larger "stretch").

What if we take our pseudorandom generator that takes an input x of length n and outputs a bit string of length $n + 1$, but instead of applying H just once, we apply it, take the output and apply H again, and so on, arbitrarily many times? It turns out that this natural idea in fact works!

Problem 8.2. (20 points) Let H be a pseudorandom generator with length extension function $\ell(n) = n+1$. Let $\ell'(n)$ be a polynomial function of n. Define H' to be a function that takes as input a bit string x of length n (for any n) and outputs $H(H(\dots H(x)\dots))$, where H is applied $\ell'(n) - n$ times (and thus $H'(x)$ has length $\ell'(n)$). Prove that H' is a PRG.

9 Improving on Indistinguishability of Encryptions

Before this section, we've dealt with one particular notion of security: computational indistinguishability of encryptions. But in some applications, this isn't a sufficiently strict definition of security.

9.1 CPA Security

We now define a stricter standard of security than computational indistinguishability of encryptions.

Definition 9.1.1. An encryption scheme (G, E, D) is *secure against chosen-plaintext attacks* (we say that the encryption scheme is *CPA secure*) if there exists a negligible function $\text{neg}(n)$ such that for all n, every PPT algorithm A wins the following game with probability at most $\frac{1}{2} + \text{neg}(n)$:

1) A referee uses G to generate a key k.

2) A gets *oracle access* to E_k. This means that, as many times as it wants, A can input any message $m \in \{0,1\}^*$ into the algorithm E_k and receive the output $E_k(m)$. However, A does not learn k. Then, A chooses two messages $m_0, m_1 \in \{0,1\}^n$ and gives (m_0, m_1) to the referee. (A gets 1^n as input, meaning that its run-time must be polynomial in n.)

3) The referee chooses $b \in \{0,1\}$ uniformly at random and sends $c = E_k(m_b)$ to A.

4) A, which still has oracle access to E_k, is run on input $(c, 1^n)$ (so A must run in polynomial time in the larger of n and the length of c), and outputs $b' \in \{0,1\}$.

5) A wins if $b = b'$.

Problem 9.1. (3 points) Your teammate tells you, "Wait! Nothing can be CPA-secure: A can simply choose two messages m_0 and m_1, ask the oracle for $E_k(m_0)$ and $E_k(m_1)$, and then when the referee gives c to A, output 0 if $c = E_k(m_0)$ and 1 if $c = E_k(m_1)$." Explain why your teammate is incorrect. (Hint: examine Definition 4.1.1 closely.)

Problem 9.2. (7 points) Prove that if an encryption scheme is CPA-secure, then it satisfies computational indistinguishability of encryptions.

9.2 Constructing a CPA-Secure Encryption Scheme

This section consists of three problems that will, together, show you how to construct a CPA-secure encryption scheme from a PRG. All three problems are hard, so don't be discouraged if you can't solve them. But first, a definition.[21]

Definition 9.2.1. For $n \geq 0$, let I_n be a set of keys (called the *key index space*). Let $\ell(n) : \mathbb{N} \to \mathbb{N}$ be a function (usually $\ell(n) = n$ though we'll see one case where $\ell(n) = 2n$). Let $\mathcal{F} = \bigcup_{n \geq 0} \mathcal{F}_n$, where

$$\mathcal{F}_n = \{f_k : \{0,1\}^{\ell(n)} \to \{0,1\}^{\ell(n)}\}_{k \in I_n}.$$

$\mathcal{F}$ is a *family of pseudorandom functions* (for short we say that $\mathcal{F}$ is a PRF) if:

- There is a PPT algorithm G such that $G(1^n)$ generates a key $k \in I_n$ (not necessarily uniformly at random from I_n).
- Given $k \in I_n$, f_k is computable in polynomial time.

[21] In the definition and throughout this section, $\bigcup_{n \geq 0}$ denotes the union of the stated set for all $n \geq 0$.

- There exists a negligible function $\text{neg}(n)$ such that for all n, for every PPT algorithm D,

$$\left|\Pr\left[D^{f_k(\cdot)}=1\right]-\Pr\left[D^{f(\cdot)}(1^n)=1\right]\right|\leq \text{neg}(n),$$

  where the first probability is for k randomly generated by $G(1^n)$ and the second probability is for a function chosen uniformly at random from the set of all functions $f:\{0,1\}^{\ell(n)}\to\{0,1\}^{\ell(n)}$. Here $D^{f(\cdot)}$ means that D has oracle access to f.

Okay, let's talk about this. A PRF $\mathcal{F}$ is an infinite collection of easily-computable length-preserving functions. But $\mathcal{F}$ is pseudorandom; think of it this way. You're given a mysterious input-output machine inside a black box you can't open. All you can do is put an input into the machine and get out an output (as many times as you want). The function was randomly chosen in one of two ways: either it's literally a randomly chosen function from the set of functions from $\{0,1\}^{\ell(n)}$ to $\{0,1\}^{\ell(n)}$, or it's a function chosen from $\mathcal{F}_n$ by G. Your goal is to try to distinguish between these two possibilities by outputting 1 with a (non-negligibly) higher probability in one case than in the other. The last (pseudorandom) property of a PRF says you can't succeed in doing this using a polynomial-time distinguisher.

Problem 9.3 (PRG $\to$ PRF)**.** (30 points) Let H be a PRG whose output on an input of length n has length $2n$. For $x\in\{0,1\}^n$, let $H_0(x)$ be the first n bits of $H(x)$ and $H_1(x)$ be the last n bits of $H(x)$. For $k,x\in\{0,1\}^n$, let

$$f_k(x)=H_{x_n}(H_{x_{n-1}}(\dots(H_{x_2}(H_{x_1}(k)))\dots)).$$

Prove that

$$\mathcal{F}=\bigcup_{n\geq 0}\mathcal{F}_n=\bigcup_{n\geq 0}\{f_k\mid k\in\{0,1\}^n\}$$

is a PRF.

Definition 9.2.2. A *family of pseudorandom permutations* (PRP) is a PRF such that for all k, f_k is bijective and f_k^{-1} is computable in polynominal time.

Problem 9.4 (PRF $\to$ PRP)**.** (35 points) Given a PRF

$$\mathcal{F}=\bigcup_{n\geq 0}\{f_k:\{0,1\}^n\to\{0,1\}^n\}_{k\in I_n},$$

let

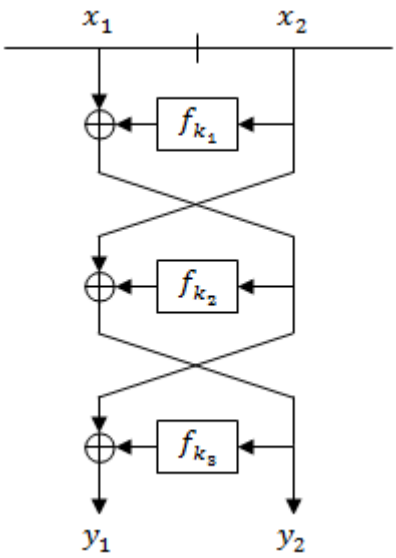
$$\mathcal{G}=\bigcup_{n\geq 0}\{g_k:\{0,1\}^{2n}\to\{0,1\}^{2n}\}_{k\in I_n^3},$$

where

$$g_{(k_1,k_2,k_3)}(x)=(x_1\oplus f_{k_1}(x_2)\oplus f_{k_3}(x_2\oplus f_{k_2}(x_1\oplus f_{k_1}(x_2))))\circ(x_2\oplus f_{k_2}(x_1\oplus f_{k_1}(x_2)))$$

for $k_1,k_2,k_3\in I_n$ for some n, where x_1 and x_2 are respectively the first n bits and the last n bits of x. Prove that $\mathcal{G}$ is a PRP.

It may be helpful for you to think of $g_{(k_1,k_2,k_3)}(x)$ as in the following diagram, where the output of g is $y_1\circ y_2$.



Problem 9.5 (PRP $\to$ secure encryption)**.** (30 points) Let $\mathcal{F}$ be a PRP with key generation algorithm G. Define an encryption algorithm as follows:

- The key generation algorithm is G.
- The plaintext space P_n is the set of bit strings whose length is a multiple of n not exceeding n^2.
- The encryption algorithm $E_k(m)$ is defined as follows. Let $m = m_1 \circ m_2 \circ \cdots \circ m_t$, where each m_i has length n. Uniformly randomly choose a string $V_I \in \{0,1\}^n$ (this is known as the *initial vector*). Let $V_I + a$ (for any $a \in \{0,1\}^n$) denote the bit string corresponding to the sum of V_I and a, modulo 2^n. Let $c_1 = f_k(V_I) \oplus m_1$, $c_2 = f_k(V_I + 1) \oplus m_2$, $c_3 = f_k(V_I + 2) \oplus m_3$, and so on, until $c_t = f_k(V_I + t - 1) \oplus m_t$. Then $E_k(m) = V_I \circ c_1 \circ c_2 \circ \cdots \circ c_t$.

What is the corresponding decryption algorithm D? Prove that (G, E, D) is a CPA-secure encryption scheme.[22]

Well, that's it! You have now shown, conditional on some assumptions, how to construct a very secure encryption scheme. We hope you enjoyed these problems!

[22]The choice of n^2 in the second bullet point is arbitrary and could have been any polynomial. We want a polynomial bound on the length of the messages in P_n.

Team Number: _____

PUMaC 2016 Power Round Cover Sheet

Remember that this sheet comes first in your stapled solutions. You should submit solutions for the problems in increasing order. Write on one side of the page only. The start of a solution to a problem should start on a new page. Please mark which questions for which you submitted a solution to help us keep track of your solutions.

Problem Number	Points	Attempted?
2.1 (a)	1	
2.1 (b)	1	
2.1 (c)	1	
2.1 (d)	2	
2.2	3	
2.3 (a)	1	
2.3 (b)	1	
2.3 (c)	1	
2.3 (d)	2	
2.4 (a)	3	
2.4 (b)	3	
2.5 (a)	2	
2.5 (b)	1	
2.5 (c)	2	
2.6 (a)	3	
2.6 (b)	3	
2.7	5	
2.8 (a)	3	
2.8 (b)	2	
3.1 (a)	1	
3.1 (b)	2	
3.1 (c)	1	
3.2 (a)	5	
3.2 (b)	3	
3.3	3	
3.4	8	
3.5	7	
3.6 (a)	2	
3.6 (b)	3	
4.1 (a)	1	
4.1 (b)	1	
4.1 (c)	2	
4.2 (a)	1	
4.2 (b)	1	
4.3 (a)	1	

Problem Number	Points	Attempted?
4.3 (b)	1	
4.3 (c)	1	
4.3 (d)	1	
4.3 (e)	1	
4.3 (f)	2	
4.4	2	
4.5	5	
5.1	3	
5.2	14	
5.3	7	
5.4 (a)	4	
5.4 (b)	2	
6.1 (a)	2	
6.1 (b)	2	
6.1 (c)	2	
6.1 (d)	3	
6.2	16	
7.1 (a)	3	
7.1 (b)	3	
7.1 (c)	4	
7.2 (a)	7	
7.2 (b)	5	
7.2 (c)	6	
7.2 (d)	7	
7.2 (e)	15	
7.2 (f)	5	
7.3	5	
8.1	15	
8.2	20	
9.1	3	
9.2	7	
9.3	30	
9.4	35	
9.5	30	
Total:	350	