CS 601.642/442
Modern Cryptography
Lecture Notes

Abhishek Jain

November 4, 2020

THE DOCUMENT IS UNDER CONTINUAL UPDATE

Contents

List of Figures

Acknowledgment

Thanks to the students of CS 600.442 (Fall 2016) (Alex Badiceanu, Ke Wu, David Li, Yeon Woo Kim, Aarushi Goel, Jesse Fowers, Neil Fendley, Katie Chang, Alishah Chator, Arka Rai Choudhuri, Cheng-Hao Cho) for scribing the original lecture notes.

Chapter 1

Introduction

These are an edited collection of the lecture notes for the *Modern Cryptography* course. The notes are meant to complement the lecture, and not meant as a replacement to the actual lectures.

While we shall attempt to be as formal as possible in these notes, formalism is not the main goal of the course. Often, where obvious, we shall forgo some of the formalism for what we believe to be a simpler exposition. But we shall make it a point to provide sufficient references for the complete formal exposition.

Structure. The structure of the notes will follow the structure of the course. We shall start with a recap of some of the important mathematical concepts that will be essential to the course. For the main content, we start with one of the most important conceptual building blocks of cryptography, one-way functions.

We shall then discuss the importance of randomness in cryptography, and describe what it means to be "almost random". This will motivate how we can take a small amount of randomness and generate large amounts of "almost randomness".

We then move to encryption, and define what it means for encryption to be secure. We shall see how the topics we've covered up to this point will help us construct secure encryption.

Following this we shall see how cryptography allows us to achieve a digital analogue of signatures that will be broadly referred to as authentication. We provide both definitions and constructions based on the tools developed thus far.

We shall then move to more advanced topics covering proof systems that are leak no information (zero-knowledge), secure-computation and encryption schemes that are resistant to tampering of ciphertexts.

Errata. There might be errors of various forms that may have crept in during the editing of these notes, and we would be grateful if you could send an email to Arka Rai Choudhuri achoud@cs.jhu.edu pointing them out.

Chapter 2

Preliminaries

In this chapter, we're going to discuss some of the essential mathematics that we will require through the course. This will also provide an opportunity for us to establish notation moving forward. It should be noted that we're only going to cover these concepts at a very high level, and we've provided references at the end of this chapter if you feel the need to have a more in-depth coverage of these topics.

A large section of this chapter was motivated by Boaz Barak's wonderful notes for his cryptography course [Bar].

It is not expected for a student to read this chapter in its entirety before proceeding with the rest of the notes. In fact, the student is encouraged to glance through this chapter to understand notation, and come back when certain concepts from other chapters are unclear.

2.1 Sets

We start with one of the simplest notions in mathematics, *sets*, which is a collection of *distinct* objects. Our first example is the infinite set of natural numbers,

$$\mathbb{N} := \{1, 2, 3, \cdots\}.$$

This also gives us the opportunity to establish our first notation of ":=", which we will use as the assignment operator. This is to be contrasted with "=", which is the operator used to establish equality.

The most common set we shall encounter in the course is the finite set $\{0, 1\}$. This will often be referred to as an *alphabet* as we shall use it to build strings

$$\{0,1\}^2 \coloneqq \{00, 01, 10, 11\}$$
$$\{0,1\}^* \coloneqq \{\epsilon, 0, 1, 00, 01, 10, 11, 000 \cdots\}$$

Size. We shall restrict our discussion of size to *finite sets.* We denote by $|S|$ the size of the set S, defined to be the number of elements in the set.

Example 1. *Let the set S be defined as follows:*

$$S \coloneqq \{1, 01, 10, 100, 010, 001\}.$$

Then,

$$|S| = 6$$

Note that by definition, a set only consists of *distinct* elements, and thereby it suffices to define the size simply as the number of elements.

Membership. For an element x, we shall indicate by $x \in S$ if x is in the set S. If not, we shall indicate it by $x \notin S$.

2.1.1 Operations

Below we describe the common set operations:

Union. The union of two sets A and B, denoted by $A \cup B$ is defined as

$$A \cup B \coloneqq \{x \mid x \in A \text{ OR } x \in B\},$$

to be the set of elements that are present in either A or B.

The above shorthand notation is called the *set-builder notation*, that describes the properties an element x must satisfy to be a part of the set $A \cup B$. We shall use the *set-builder notation* frequently in this course.

Intersection The intersection of two sets A and B, denoted by $A \cap B$ is defined as

$$A \cap B \coloneqq \{x \mid x \in A \text{ AND } x \in B\},$$

to be the elements present only in both A and B.

Difference The set difference of A and B, denoted by $A \setminus B$, is defined as

$$A \setminus B \coloneqq \{x \mid x \in A \text{ AND } x \notin B\}\,,$$

to be the elements present in A but not in B.

Note that the for set difference, the order of the two sets matter, and $A \setminus B \neq B \setminus A$. Think of a simple example to illustrate this.

Complement The complement of a set A is defined with respect to the *universe*, denoted by U. The universe denotes all possible elements which can be used to construct the set A. The complement of A, denoted by $\overline{A}$, is defined as

$$\overline{A} \coloneqq \{x \mid x \notin A\}\,,$$

to be the elements not present in A.

There are standard relations between these set operations, and we encourage the reader to work these out if they aren't familiar with them.

These four common set operations are illustrated in Figures 2.1,2.2, 2.3 and 2.4.

For our context, it is easiest of think of the universe to be $\{0,1\}^*$, the set of all binary strings.

Cartesian product. The cartesian product of sets A and B, denoted by $A \times B$ is defined as

$$A \times B \coloneqq \{(a,b) \mid a \in A \text{ AND } b \in B\}$$

is the set of all *ordered pairs* (a,b) when $a \in A$ and $b \in B$. To differentiate unordered sets from an ordered tuple, we will represented ordered tuples by brackets $(\cdot)$.

Since we are talking about ordered tuples, $A \times B$ and $B \times A$ give rise to different sets since $(a,b) \neq (b,a)$ when a and b are distinct.

Subsets. For sets A and B, A is a subset of B, denoted by $A \subseteq B$, if every element of A is also an element of B.

Two sets A and B are equal if *both* $A \subseteq B$ and $B \subseteq A$. When arguing two sets are equal, this is the go-to method for a proof.

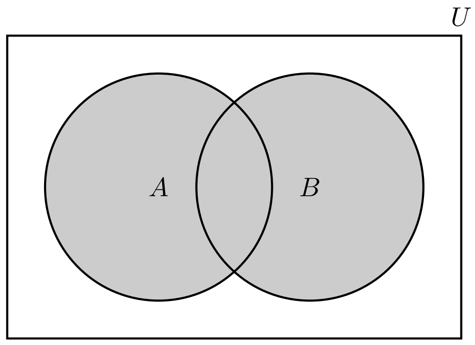


Figure 2.1: $A \cup B$

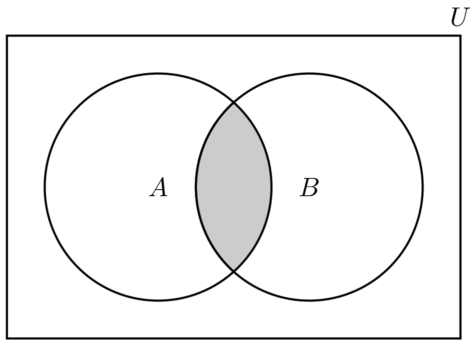


Figure 2.2: $A \cap B$

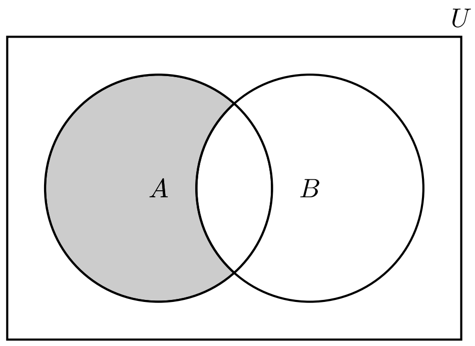


Figure 2.3: $A \setminus B$

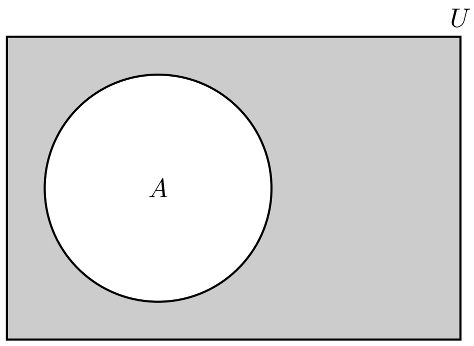


Figure 2.4: $\overline{A}$

2.2 Relations and Functions

2.2.1 Relation

A relation R from A to B is a subset of the Cartesian product $A \times B$. Consider the following relation R_1 from $A \coloneqq \{x_1, x_2, x_3, x_4\}$ to $B \coloneqq \{y_1, y_2, y_3, y_4\}$,

$$R_1 \coloneqq \{(x_1, y_2), (x_1, y_3), (x_2, y_4), (x_3, y_3), (x_4, y_1)\}$$

The *domain* of a relation is the subset of A that appear as the first component in a relation; and the *range* is the subset of B that appears as the second component in the relation. We shall typically consider relations where the domain is the entire set A. The set B is referred to as the co-domain of the relation.

2.2.2 Functions

A function $f : A \mapsto B$ is a relation from A to B with the additional restriction that each element in the domain A appears in exactly one ordered pair in the relation. The relation R_1 is not a function since x_1 appears in two ordered pairs. Consider R_2 over the same sets A and B.

$$R_s \coloneqq \{(x_1, y_2), (x_2, y_1), (x_3, y_3), (x_4, y_1)\}$$

The relations R_1 and R_2 are illustrated in Figures 2.5 and 2.6.

In function terminology, the elements from A are referred to as inputs, and the corresponding second element in the pair from B to the output. When we consider boolean functions, it is common to represent functions by a truth table, i.e. a row corresponding to each possible input and the corresponding output of the function.

Example 2. *What are the total number of boolean functions with n_1 inputs and n_2 outputs?*

Solution Let us consider first the simple case of a single output, and we shall see how to extend this to multiple outputs. As discussed above, every function can be represented by the corresponding truth table. For a function with n_1 inputs, any truth table will have 2^{n_1} rows.

For each row, since there is a single output, we can specify the output for that row to be either 0 or 1. Specifying such a value for each row determines the function. The total number of ways one can fill in this column then determines the number of functions. Hence the total number of functions with a single bit of output is $2^{2^{n_1}}$.

x_1	x_2	$\cdots$	x_{n_1}	y
0	0	$\cdots$	0	r_1
0	0	$\cdots$	1	r_2
		$\vdots$		
1	1	$\cdots$	0	r_{N-1}
1	1	$\cdots$	1	r_N

Now let us extend this to multiple output bits. The first important point to note is that the number of rows in the truth table remains unchanged.

x_1	x_2	$\cdots$	x_{n_1}	y_1	$\cdots$	y_{n_2}
0	0	$\cdots$	0	$r_{1,1}$	$\cdots$	$r_{n_2,1}$
0	0	$\cdots$	1	$r_{1,2}$	$\cdots$	$r_{n_2,2}$
			$\vdots$			
1	1	$\cdots$	0	$r_{1,N-1}$	$\cdots$	$r_{n_2,N-1}$
1	1	$\cdots$	1	$r_{1,N}$	$\cdots$	$r_{n_2,N}$

$$\underbrace{2^{2^{n_1}} \times \cdots \times 2^{2^{n_1}}}_{n_2 \text{times}} = 2^{n_2 2^{n_1}}$$

□

We now describe some special cases of functions below.

Injective Function. *Injective* functions or 1-1 functions are functions where each input has a distinct output, i.e. there does not exist $x, x' \in A$ such that $f(x) = f(x')$. This immediately requires $|B| \geq |A|$.

Surjective Function. A function is *surjective* if the co-domain B is the same as the range, i.e. every element of B is an output of the function for some element of A. This requires $|A| \geq |B|$.

Bijective Function. A function that is both *injective* and *surjective* is called a *bijective* function.

Permutation. A permutation on a set A is defined to be a *bijection* from A to itself. It is common to denote a permutation as $\Pi : A \mapsto A$.

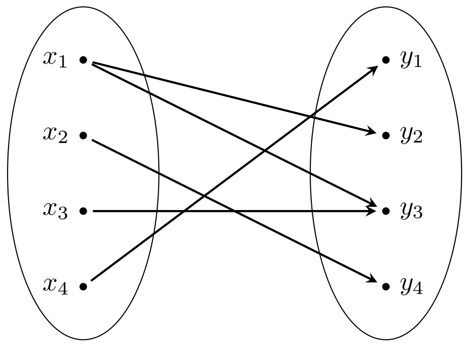


Figure 2.5: An example relation R_1

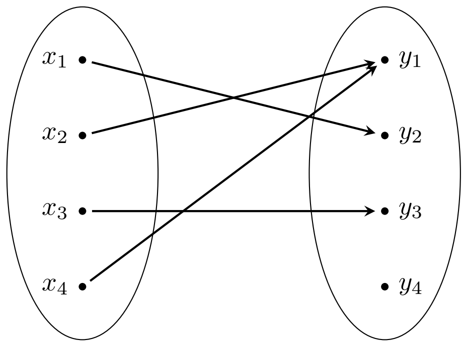


Figure 2.6: An example function R_2

2.2.3 Logical Operations

We describe below the four most common logical/boolean operations. While the AND, OR and XOR gate have been described with two inputs, they can easily be extended to multiple inputs.

AND. AND gate

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

OR. OR gate

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

XOR. XOR gate

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

NOT.

x	$\overline{x}$ (or $\neg x$)
0	1
1	0

2.2.4 Quantification.

We will often want to make a claim about a variable by either claiming a statement holds for all values that the variable takes, or there is at least one value satisfying the statement. The following are largely taken from the notes in [Lov], and you should take a look there for further details.

Universal Quantification.

$$\forall x \in A,\ P(x)$$

which indicates that for all x in the set A, the statement $P(x)$ is true.

Existential Quantification.

$$\exists x \in A,\ P(x)$$

which indicates that there is at least one x in A such that the statement $P(x)$ is true.

Nesting Quantifiers. We can nest the above quantifiers in an arbitrary manner. For instance, the following is a valid nesting

$$\forall x_1 \in A_1, \forall x_2 \in A_2, \exists x_3 \in A_3, \forall x_4 \in A_4\ P(x_1, x_2, x_3, x_4).$$

Given the fact that the nesting can be arbitrary, it is important to ask if there order of quantification matters. If the quantifiers are the same, then the order of nesting does not matter. Therefore, $\forall x_1 \in A_1, \forall x_2 \in A_2\ P(x_1, x_2)$ and $\forall x_2 \in A_2, \forall x_1 \in A_1\ P(x_1, x_2)$ are equivalent, and the same is true for the existential quantifiers. But things aren't as simple when the quantifiers are different.

Remark 1 (Order of Quantification). *When the nested quantifiers are different, the order of quantifiers matters. We illustrate this with an example below.*

Example 3. *Consider the two following statements*

1. $\forall x \in \mathbb{Z}, \exists y \in \mathbb{Z}$ *s.t.* $x + y = 4$
2. $\exists x \in \mathbb{Z},$ *s.t.* $\forall y \in \mathbb{Z}\ x + y = 4$

The first statement is true, since for every fixed x, we can set y to be $4 - x$. Thus existence of such a $y \in \mathbb{Z}$ is guaranteed. On the other hand, the second statement says that there must be a single x such that for every value of $y \in \mathbb{Z}$, $x + y = 4$. It is clear to see that this statement cannot be true.

In fact, the second ordering is a stronger claim than the first.

Negation of Quantifiers. When negating a quantified statement, negate all the quantifiers first, from left to right (keeping the same order), then negate the statement. By negating quantifiers we mean swapping $\forall$ and $\exists$. Below are few examples of negation

1. $\neg (\forall x \in A,\ P(x)) \iff \exists x \in A,\ \neg P(x)$
2. $\neg (\exists x \in A,\ P(x)) \iff \forall x \in A,\ \neg P(x)$
3. $\neg (\exists x \in A, \forall y \in B,\ P(x, y)) \iff \forall x \in A, \exists y \in B,\ \neg P(x, y)$
4. $\neg (\forall x \in A, \exists y \in B,\ P(x, y)) \iff \exists x \in A, \forall y \in B,\ \neg P(x, y)$

Here, $\iff$ is the notation for *if and only if* that connects two statements, where either both statements are true or both are false.

2.3 Reductio ad Absurdum

Reduction to absurdity, or proof by contradiction is a common proof technique that we shall employ throughout this course. We start with the statement we want to prove, and then assume that the statement is false, and then derive as a consequence a statement we believe to be false.

The following is an example taken from Boaz Barak's lecture notes:

Theorem 1. *There are infinitely many primes.*

Proof Let us assume for that the above statement is indeed false and there are a finite number of primes. Let us denote the primes by $p_1, \cdots, p_N$ with N indicating the total number of primes.

Consider the following number,

$$P = p_1 \cdot p_2 \cdots p_N + 1,$$

i.e. P is one greater than the product of all the primes. It is clear that none of the primes, $p_1, \cdots, p_N$ divide P, since the remainder with respect to all of them is 1.

Therefore, P has only two factors 1 and P, establishing it as a prime. But P is clearly not in the set of finite primes, therefore a contradiction to our assumption of a finite set of primes.

Since the same argument can be applied to any set of finite primes, it must be the case that the number of primes are infinite. □

Another common example of this type of proof technique is to prove that $\sqrt{2}$ cannot be expressed as an irreducible fraction, thereby establish that it is irrational.

2.4 Graphs

We describe below the notations we shall use to describe graphs, and their corresponding representation.

Definition 1 (Graphs). *A Graph $G = (V, E)$ where V is a set of vertices and E is a set of edges s.t. $|V| = n$, $|E| = m$.*

Example 4.

$$\begin{aligned} V &\coloneqq \{v_1, v_2, v_3, v_4, v_5, v_6\} \\ E &\coloneqq \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_4, v_1\}, \{v_4, v_5\}, \{v_5, v_6\}\} \end{aligned}$$

The order of the graph is the number of vertices, $|V| = n$, and the size of the graph is the number of edges, $|E| = m$.

To use graphs, we shall need to find a way to represent them. For this course, we shall find it convenient to represent graphs by an *adjacency matrix*.

Adjacency Matrix. Any graph can be represented as an adjacency matrix. The number of rows and columns of this matrix is the same as the number of vertices in the graph. A value of 1 at a given position represents the presence of an edge between the vertices corresponding to the row and column. More specifically:

Definition 2 (Adjacency Matrix). *A graph $G = (V, E)$ with $|V| = n$, can be represented as an $n \times n$ adjacency matrix M_G of boolean values such that:*

$$M_G[i,j] = \begin{cases} 1 & \textit{if } (i,j) \textit{ or } (j,i) \in E \\ 0 & \textit{otherwise} \end{cases}$$

$$\begin{array}{cc} \begin{array}{cccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} & \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} & \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} \end{array}$$

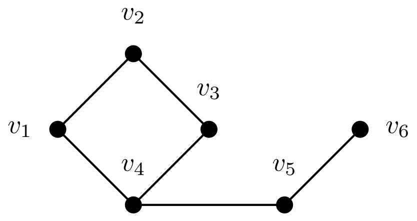


Figure 2.7: M_G and the corresponding graph G

2.5 Probability

Throughout this course, we shall see the importance of randomness, in that it permeates every aspect of cryptography. In this section, we shall review some of the relevant material for discrete probability.

Sample Space: Sample space S of a probabilistic experiment is the set of all possible outcomes of the experiment. A couple points of note:

- We will only consider finite sample spaces
- In most cases, the sample space will be the set $\{0, 1\}^k$ of size 2^k

Probability Distribution: With the sample space, we now define a distribution which assigns probabilities to this sample space.

Definition 3 (Distribution). *X is a **distribution** over a sample space $\mathcal{S}$ if it assigns a probability p_s to the element $s \in \mathcal{S}$ such that*

$$\sum_{s \in \mathcal{S}} p_s = 1.$$

A common distribution is a *uniform distribution* where each element in the sample space is assigned the same probability $\frac{1}{|S|}$. For example, when the sample space is the set $\{0,1\}^k$, this distribution is denoted by U_k with each k-bit string assigned probability $\frac{1}{2^k}$.

Sampling From Distribution: We say we sample an element x from the distribution X if each element in S is picked proportional to the probability defined by the distribution X. We denote this by $x \leftarrow\$ X$. For uniform distribution, we shall find it convenient to denote sampling from the uniform distribution by $x \leftarrow\$ \{0,1\}^k$ to indicate sampling uniformly from the set $\{0,1\}^k$.

Event: Event is a subset of the sample space. Let $E \subset S$ be an event, the probability of the event E, denoted by $\Pr[E]$, is defined as

$$\Pr[E] \coloneqq \sum_{s \in E} p_s$$

Union Bound: If S is a sample space and $A, A' \subseteq S$, then the probability that either A or A' occurs is $\Pr[A \cup A'] \leq \Pr[A] + \Pr[A']$

Random Variables: A random variable is a function that maps elements of the sample space to another set. It assigns values to each of the experiment's outcomes.

Example 5. *In the uniform distribution $\{0,1\}^3$, Let Random Variable N denote the number of $1s$ in the chosen string, i.e., for every $x \in \{0,1\}^3$, $N(x)$ is the number of $1s$ in x.*

$$\Pr_{x \leftarrow\$ \{0,1\}^3}[N(x) = 2] = \frac{3}{8}$$

Expectation: The expectation of a random variable is its weighted average, where the average is weighted according to the probability measure on the sample space. The expectation of random variable N is defined as:

$$\mathbb{E}[N] = \sum_{x \in S} N(x) \cdot \Pr_{y \leftarrow\$ S}[y = x] \tag{2.1}$$

Here S is the sample space and $\Pr_{y \leftarrow\$ S}[y = x]$ is the probability of obtaining x when sampling from S.

Example 6. *If N is defined as in the above example,*

$$\mathbb{E}[N] = 0 \cdot \frac{1}{8} + 1 \cdot \frac{3}{8} + 2 \cdot \frac{3}{8} + 3 \cdot \frac{1}{8} = 1.5$$

Expectation is a linear function, i.e.,

$$\mathbb{E}[N + M] = \mathbb{E}[N] + \mathbb{E}[M]$$

Variance: Variance of a random variable N is defined as the expectation of the square of the distance of N from its expectation.

$$\mathsf{Var}[N] = \mathbb{E}\Big[(N - \mathbb{E}[N])^2\Big] \tag{2.2}$$

If N is defined as in the first example,

$$\begin{aligned}\mathsf{Var}[N] &= (0-1.5)^2 \cdot \frac{1}{8} + (1-1.5)^2 \cdot \frac{3}{8} + (2-1.5)^2 \cdot \frac{3}{8} + (3-1.5)^2 \cdot \frac{1}{8} \\ &= 0.75\end{aligned}$$

Variance is a measure of how spread out the values in a distribution are. A low variance means the outcomes will usually be very close to one another.

Standard Deviation: Standard deviation of N is the square root of $\mathsf{Var}[N]$

Conditional Probability: The conditional probability of event B in relation to event A is the probability of event B occurring when we know that A has already occurred.

$$\Pr[B \mid A] = \frac{\Pr[A \cap B]}{\Pr[A]} \tag{2.3}$$

Example 7. *Drawing Kings from a deck of cards. Event A is drawing a King first, and Event B is drawing a King second.*

$$\Pr[A] = \frac{4}{52}$$

$$\Pr[B|A] = \frac{3}{51}$$

Law of Total Probability: Throughout the course we will constantly use the law of total probability with regards to conditional probability.

Let $B_1, B_2, \cdots, B_n$ be a disjoint partition of a sample space S (i.e. $\forall i, j\ B_i \cap B_j = \emptyset$). Then for any event E,

$$\boxed{\Pr[E] = \sum_{i=1}^{n} \Pr[E \cap B_i] = \sum_{i=1}^{n} \Pr[E \mid B_i] \Pr[B_i]}$$

Let's look at a very simple example.

Example 8. *Consider any event E. Let us sample a single bit b uniformly at random. B_1 is the event that $b = 0$, and B_2 the event that $b = 1$. We can write, the probability of event E as*

$$\begin{aligned}\Pr[E] &= \Pr[E \mid b = 0] \Pr_{b \leftarrow\$ \{0,1\}}[b = 0] + \Pr[E \mid b = 1] \Pr_{b \leftarrow\$ \{0,1\}}[b = 1] \\ &= \Pr[E \mid b = 0] \cdot \frac{1}{2} + \Pr[E \mid b = 1] \cdot \frac{1}{2} \\ &= \frac{1}{2} \cdot (\Pr[E \mid b = 0] + \Pr[E \mid b = 1])\end{aligned}$$

Independent Events: We say that B is independent from A if $\Pr[B \mid A] = \Pr[B]$, i.e.,

$$\Pr[A \cap B] = \Pr[A] \cdot \Pr[B]$$

Example 9. *Tossing a coin. The probability that heads shows up on two consecutive coin tosses,*

$$\Pr[HH] = \Pr[H]. \Pr[H] = \frac{1}{2}.\frac{1}{2} = 0.25$$

Each toss of a coin is an Independent Event.

Now let's look at example where the individual samples are random, but the joint distributions are not.

Example 10. *Consider the following random variables X and Y both taking values in $\{0, 1\}$ with the following joint distribution:*

$$\begin{aligned}\Pr[X = 0 \wedge Y = 0] &= \frac{3}{16}, \quad \Pr[X = 0 \wedge Y = 1] = \frac{5}{16} \\ \Pr[X = 1 \wedge Y = 0] &= \frac{5}{16}, \quad \Pr[X = 1 \wedge Y = 1] = \frac{3}{16}\end{aligned}$$

Using the law of total probability, we have

$$\Pr[X=0]=\Pr[X=1]=\frac{1}{2}$$
$$\Pr[Y=0]=\Pr[Y=1]=\frac{1}{2}$$

It is going to be important to keep this point in mind: even if individual distributions are uniform joint distributions need not be.

but for each $x, y \in \{0,1\}$ we have

$$\Pr[X=a \wedge Y=b] \neq \Pr[X=a]\cdot\Pr[X=b].$$

Note that we're using the symbol $\wedge$ indicating 'AND' to indicate independence while we've previously spoken about the set intersection $\cap$ when describing independence. These two symbols will be used to convey the same meaning; while $\cap$ will be used exclusively for sets, $\wedge$ will be used when we talk about random variables.

Pairwise Independent Random Variables: Let $X_1, X_2......X_n$ be random variables. We say that the X_i's are pairwise-independent if for every $i \neq j$ and all a and b, it holds that:

$$\Pr[X_i=a \wedge X_j=b]=\Pr[X_i=a]\cdot\Pr[X_j=a] \tag{2.4}$$

Example 11. *We throw two dice. Let (i) A be the event "the sum of the points is 7"; (ii) B the event "die #1 came up 3"; and (iii) C the event "die #2 came up 4".*

$$\Pr[A]=\Pr[B]=\Pr[C]=\frac{1}{6}$$

$$\Pr[A\cap B]=\Pr[B\cap C]=\Pr[A\cap C]=\frac{1}{36}$$

But,

$$\Pr[A\cap B\cap C]=\frac{1}{36}\neq\Pr[A]\cdot\Pr[B]\cdot\Pr[C]$$

A, B and C are pairwise independent but not independent as a triplet.

2.6 Tail Bounds

Markov's Inequality : Let X be a *non-negative random variable* and let $k \geq 1$. Then,

$$\Pr[X \geq k] \leq \frac{\mathbb{E}[X]}{k} \tag{2.5}$$

or

$$\Pr[X \geq k \cdot \mathbb{E}[X]] \leq \frac{1}{k} \tag{2.6}$$

Example 12. *Suppose we roll a single fair die and let X be the outcome. Then,*

$$\mathbb{E}[X] = 3.5$$
$$\mathsf{Var}[X] = \frac{35}{12}$$

Suppose we want to compute $\Pr[X \geq 6]$. *We can easily see that* $\Pr[X \geq 6] \geq \Pr[X = 6] \approx 0.167$. *Using Markov's Inequality we can get an upper bound,*

$$\Pr[X \geq 6] \leq \frac{3.5}{6} \approx 0.583$$

Markov's inequality gives an answer to the question “what is the probability that the value of the r.v., X, is ‘far’ from its expectation?".

Chebyshev's Inequality: Let X be a random variable, and $k \geq 1$. Then,

$$\Pr[|X - \mathbb{E}[X]| \geq k] \leq \frac{\mathsf{Var}[X]}{k^2} \tag{2.7}$$

Another answer to the question of “what is the probability that the value of X is far from its expectation" is given by Chebyshev's Inequality, which works for *any random variable* (not necessarily a non-negative one).

Example 13. *Let X be as defined in the above example. We can get a better bound on* $\Pr[X \geq 6]$ *using Chebyshev's inequality,*

$$\begin{aligned}\Pr[X \geq 6] &\leq \Pr[X \geq 6 \vee X \leq 1] \\ &= \Pr[|X - 3.5| \geq 2.5] \\ &\leq \frac{35}{12} \cdot \frac{1}{(2.5)^2} = \frac{7}{15} \approx 0.46\end{aligned}$$

Chernoff Bound: Let $X_1, X_2 \ldots X_n$ be independent random variables with $0 \leq X_i \leq 1$ (they need not have the same distribution). Let $X = X_1 + \ldots + X_n$, and $\mu = \mathbb{E}[X] = \mathbb{E}[X_1] + \cdots + \mathbb{E}[X_n]$, then for any $\delta \geq 0$

$$\boxed{\Pr[X \geq (1+\delta)\mu] \leq e^{-\frac{\delta^2}{2+\delta}\mu} \qquad (2.8)}$$

This inequality is a particularly useful for analysis of the error probability of approximation via repeated sampling.

Example 14. *A biased coin, which lands heads with probability $\frac{1}{10}$ each time it is flipped, is flipped 200 times consecutively. We want an upper bound on the probability that it lands heads at least 120 times. The number of heads is a binomially distributed r.v., X, with parameters $p = \frac{1}{10}$ and $n = 200$. Thus, the expected number of heads is*

$$\mathbb{E}[x] = n \cdot p = 200 \cdot \frac{1}{10} = 20$$

Using Markov's inequality

$$\Pr[X \geq 120] \leq \frac{20}{120} = \frac{1}{6}$$

Using Chernoff bound we get,

$$\Pr[X \geq 120] = \Pr[X \geq (1+5)\cdot 20] \leq e^{-\frac{5^2}{2+5}20} \approx e^{-71.4}$$

which is a much better bound.

For specific random variables, particularly those that arise as sums of many independent random variables, we can get much better bounds on the probability of deviation from expectation. See Figure 2.8 for an example of how quickly the above example centers around the mean with a Gaussian distribution.

2.7 Model of Computation - Turing Machines

Throughout this course, it is important to understand the model of computation that we shall work with. This will be useful when we want to model various participants in the primitives and protocols we develop.

In this course, we shall limit our model of computation to that of Turing Machines, which we describe informally below. While it is not important to understand the workings of a Turing Machine, it is presented here for completeness. The below description is taken verbatim from [Kat].

A Turing Machine is defined by an integer $k \geq 1$, a finite set of states Q, an alphabet Γ, a transition function $\delta : Q \times \Gamma^k \mapsto Q \times \Gamma^{k-1} \times \{\mathsf{L}, \mathsf{S}, \mathsf{R}\}^k$ where:

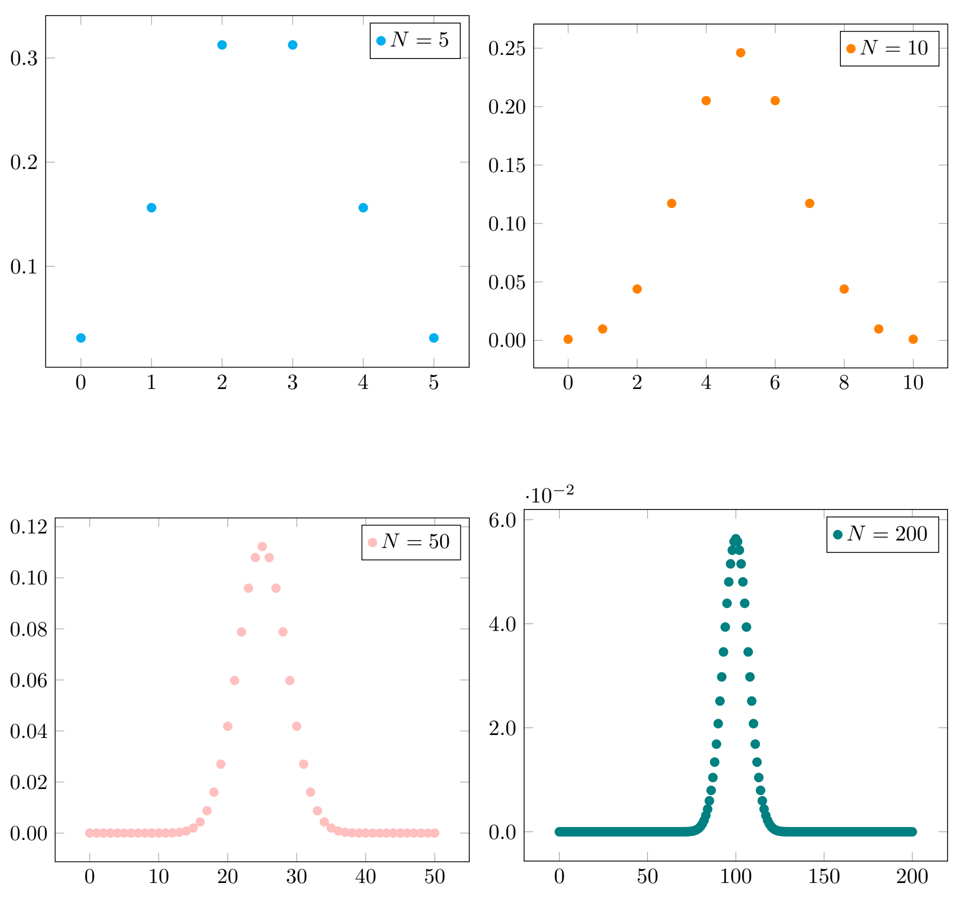


Figure 2.8: The above example, taken from [Bar] illustrates the Chernoff bound shows how the sum of coin tosses quickly converge (with N, the total number of samples) to the Gaussian Distribution; where the probability density is centered around the mean.

- k is the number of infinite, one-dimensional tapes used by the machine. We typically have $k \geq 3$, where the first tape is denoted as the the *input tape*, and the last tape the *output tape*. The position of the tape currently being read is specified by a separate tape head for each tape.
- the set of states Q contains two special states: (a) the start state q_{start}; and (b) the halt state q_{halt}.
- Γ contains $\{0, 1\}$, a special "blank symbol", and a special "start symbol".

The computation of a Turing machine M on input $x \in \{0,1\}^*$ proceeds as follows: All tapes of the Turing machine contain the start symbol followed by the blank symbols, with the exception of the input tape which contains the input x. The machine starts in state $q = q_{\mathsf{start}}$ with its k heads at the leftmost position of each tape. Then, until q is the halt state, repeat the following:

1. Let the current contents of the cells being scanned by the k heads be $\gamma_1, \cdots, \gamma_k \in \Gamma$.
2. Compute $\delta(q, \gamma_1, \cdots, \gamma_k) = (q', \gamma'_2, \cdots, \gamma'_k, D_1, \cdots, D_k)$ where $q' \in Q$, $\gamma'_2, \cdots, \gamma'_k \in \Gamma$ and $D_i \in \{\mathsf{L}, \mathsf{S}, \mathsf{R}\}$.
3. Overwrite the contents of the currently scanned cell on tape i to γ'_i for $2 \leq i \leq k$; move head i to the left, to the same position, or to the right depending on whether D_i = L, S, or R, respectively; and then set the current state to $q = q'$.

The output of M on input x, denoted $\mathsf{M}(x)$, is the binary string contained on the output tape when the machine halts. It is possible that M never halts for certain inputs x.

The above description is that of a **Deterministic Turing Machine**. We contrast this with a **Non-Deterministic Turing Machine** (NDTM). An NDTM instead of having a single transition function δ, has *two* transition function δ_0, δ_1 and at each step decides in a non-deterministic/arbitrary manner. Therefore, after n steps the machine can be in at most 2^n configurations, where a configuration is the joint description of all the tapes of the Turing Machine.

We fill often find it more convenient to talk about algorithms, defined below.

Definition 4 (Algorithm). *An algorithm is a Turing Machine whose input and output are strings over the binary alphabet* $\Sigma = \{0,1\}$.

Note, that the two terms *Turing Machine* and *Algorithm* will be used interchangeably from now on.

Definition 5 (Running Time). *An algorithm A is said to run in time $T(n)$ if for all strings of length n over the input alphabet ($x \in \{0,1\}^n$), $A(x)$ halts within $T(|x|)$ steps.*

In this course, we will primarily focus on algorithms that have a *polynomial* running time.

Definition 6 (Polynomial Running Time). *An algorithm A is said to run in polynomial time if there exists a constant c such that A runs in time $T(n) = n^c$.*

We say an algorithm is *efficient* if it runs in polynomial time. Even for efficient algorithms, the constant c can be a large value. For example, consider c=100. In practice, n^{100} may actually be considered “inefficient". For our purposes, however, we will stick with this definition of efficiency.

If an algorithm runs exponential or super-polynomial time, i.e., $T(n) = 2^n$ or $T(n) = n^{(\log n)}$, then we will say it is *inefficient*.

One might consider other notions of efficiency, and there is nothing pristine about using polynomial time as the measure of efficiency. But throughout this course, we shall see that this is often a convenient measure since the composition of polynomials remain a polynomial.

So far, we have only considered deterministic algorithms. In computer science, and specifically cryptography, randomness plays a central role. Therefore, throughout the course, we will be interested in randomized (a.k.a. probabilistic) algorithms.

Definition 7 (Randomized Algorithm). *A randomized algorithm, also called a probabilistic polynomial time Turing machine (*PPT*) is a Turing machine that runs in polynomial time and is equipped with an extra randomness tape. Each bit of randomness tape is uniformly and independently chosen.The output of a randomized algorithm is a distribution.*

Think about the difference between a NDTM and a Randomized Algorithm.

Remark 2. *Note that we do not place any limits on the length of the ranom tape, once the randmness has been fixed, the computation is completely determinisitic.*

When we talk about randomized algorithms, wherever possible we shall make explicit the randomness used. So an algorithm $\mathsf{M}(x; r)$ indicates that M runs on input x using randomness r that is sampled as $r \leftarrow\!\$ \{0, 1\}^m$ for some m. In fact, one can think of the output of $\mathsf{M}(x; r)$ as a random variable with randomness from r.

As mentioned earlier, in practice, everyone including the adversary has some bounded computational resources. These resources can be used in a variety of intelligent ways, but they are still limited. Turing machines are able to capture all the types of computations possible given these resources. Therefore, a adversary will be a computer program or algorithm modeled as a Turing machine.

This captures what we can do efficiently ourselves and can be described as a uniform PPT Turing machine. When it comes to adversaries, we will allow them to have some extra power. Instead of having only one algorithm that works for different input lengths, it can write down potentially a different algorithm for every input size. Each of them individually could be efficient. If that is the case, overall the adversary still runs in polynomial time.

The notion of non-uniform TMs has a connection with another model of computation, computation by circuits. Look up the connection.

Definition 8 (Non-Uniform PPT Machine). *A non-uniform probabilistic polynomial time Turing machine is a Turing machine A made up of a sequence of probabilistic machines $\mathsf{M} = \{\mathsf{M}_1, \mathsf{M}_2, \cdots\}$ for which there exists a polynomial $\mathsf{p}(\cdot)$ such that for every $\mathsf{M}_i \in \mathsf{M}$, the description size $|\mathsf{M}_i|$ and the running time of M_i are at most $\mathsf{p}(i)$. We write $\mathsf{M}(x)$ to denote the distribution obtained by running $\mathsf{M}_{|x|}(x)$.*

Our adversary will usually be a non-uniform probabilistic polynomial running time algorithm (n.u. PPT).

2.8 Complexity Classes

We'll discuss some important complexity classes now. Each of these complexity classes is a collection of languages that share some common property. A language is simply a subset of $\{0,1\}^*$.

We say Turing Machine M *decides* a language L if $x \in L \implies \mathsf{M}(x) = 1$, and $x \notin L \implies \mathsf{M}(x) = 0$. We can now define the complexity classes.

Complexity Class P: A language L is in P if there exists a (deterministic) Turing Machine M_L and a polynomial $\mathsf{p}(\cdot)$ such that:

- on input strings x, machine M halts after at most $\mathsf{p}(|x|)$ steps; and
- M_L decides L.

Complexity Class NP: There are two (equivalent) ways to define the class NP. The classical way is with respect to NDTMs. We say that an NDTM M outputs 1 if there is *at least* one sequence of non-deterministic choices that results in the output tape containing 1. Similar to P, a language L is in NP if there a polynomial time NDTM M_L that decides L.

For us, it will be convenient to work with the alternate definition because it is defined with respect to : A language L is in NP if there exists a Boolean relation $R_L \subseteq \{0,1\}^* \times \{0,1\}^*$ and a polynomial $\mathsf{p}(\cdot)$ such that R_L can be recognized in (deterministic) polynomial time, and $x \in L$ if and only if there exists a y such that $|w| \leq (\mathsf{p}|x|)$ and $(x, w) \in R_L$. Such a w is called a witness for membership of $x \in L$.

Try to think of why the two notions are equivalent.

Polynomial Reduction: While this is not a class, we will need it to define the subsequent two classes. We shall not provide a formal treatment here. The notion of a reduction is useful for answering questions of the form “Is language L easier that language L'?”.

We say a language L is reducible to a language L' if there exists a polynomial-time computable function f such that $x \in L$ if and only if $f(x) \in L'$. This is expressed by writing $L \leq_p L'$.

This is called a *Karp reduction.* There are other notions of reductions in computer science.

This lets us state that L is no harder than L' since any Turin Machine M that decides L' can be used to decide L.

NP-Hardness: A language is $\mathsf{NP-Hard}$ if every language in NP is polynomially reducible to it.

Thus $\mathsf{NP-Hard}$ is the set of all languages that are at least as hard as any problem in NP.

NP-Completeness: A language is $\mathsf{NP-Complete}$ if it is both (i) in NP; and (ii) in $\mathsf{NP-Hard}$.

The class $\mathsf{NP-Complete}$ consists the set of languages (or problems) that represent all of NP on account of being the hardest problems in NP. So any statement we want to make about all of NP, it suffices to make about any language that is $\mathsf{NP-Complete}$.
Our current known understanding of the relationship between the complexity classes discussed is illustrated in Figure 2.9.

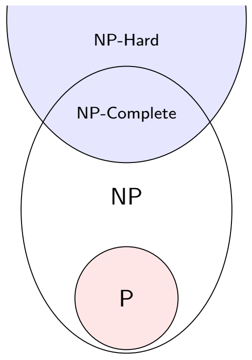


Figure 2.9: Relation between common complexity classes

Bounded error Probabilistic Polynomial Time, BPP: We now define a language for randomized Turing Machines. We say that L is recognized by the probabilistic polynomial Time Turing machine M if:

- for $x \in L$, then $\Pr[\mathsf{M}(x) = 1] \geq 2/3$
- for $x \notin L$, then $\Pr[\mathsf{M}(x) = 1] \leq 1/3$

2.9 Asymptotic Notations

When describing the running time of algorithms, instead of describing the running as a complicated function in the size of the input, it is more convenient to describe the running time with the function that reflects the growth. In view of this, we shall see that for large inputs, the multiplicative factors and lower order terms are dominated by the effects of the input size.

Big-O. The function $f(n)$ is $O(g(n))$ if $\exists$ constants c, n_0, such that $\forall n \geq n_0$

$$0 \leq f(n) \leq c \cdot g(n)$$

Big-Omega. The function $f(n)$ is $\Omega(g(n))$ if $\exists$ constants c, n_0, such that $\forall n \geq n_0$

$$0 \leq c \cdot g(n) \leq f(n)$$

Theta. The function $f(n)$ is $\theta(g(n))$ if $\exists$ constants c_1, c_2, n_0, such that $\forall n \geq n_0$

$$0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

We have illustrated these notions, and the importance of the constants c and n_0, in Figure 2.10.

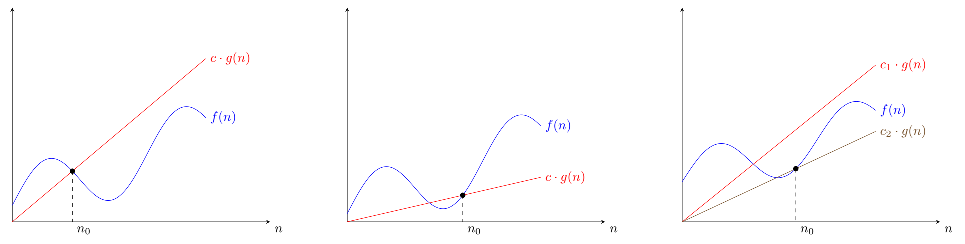


Figure 2.10: Here we give examples of the O, Ω and θ notations.

The above asymptotic notations of O and Ω may not be asymptotically, so below we describe their corresponding asymptotically tight notions.

Small-o. The function is $f(n)$ is $o(g(n))$ if $\forall$ constant c, $\exists$ constant n_0, such that $\forall$

$$0 \leq f(n) < c \cdot g(n)$$

Small-omega. The function is $f(n)$ is $\omega(g(n))$ if $\forall$ constant c, $\exists$ constant n_0, such that $\forall$

$$0 \leq c \cdot g(n) < f(n)$$

Note how the quantifiers, and their order, have changed in the above definition. Recall from our discussion of quantifiers how the order affects the definition.

Remark 3. *We have slightly abused notation above since $O(g(n))$ defines sets, but we shall find it convenient to use the above description. See Chapter 3 of [CLRS09] for a detailed discussion of these concepts.*

Chapter 3

One-Way Functions

3.1 Introduction

Intuitively, a given function f is "one-way" if it is very easy to compute $f(x)$ efficiently, but it is hard to recover x if given $f(x)$. Over the course, we shall see importance of one-way function, and its role in cryptography. Before we can get there, we need to formally define our above intuition for a function f to be one-way. Let's take a stab at such a definition below.

$$\begin{tikzcd} x \arrow[r, bend left, "\text{EASY}"] & f(x) \arrow[l, bend left, "\text{HARD}"] \end{tikzcd}$$

Definition 9 (One-way Function (Informal)). *A function f is one-way if it satisfies the following two informal properties:*

1. ***Functionality - Easy to compute:*** *Given any input x from the domain, it should be easy to compute $f(x)$. In other words it is possible to compute $f(x)$ in polynomial time.*
2. ***Security - Hard to invert:*** *Any polynomial time algorithm should fail to recover x given $f(x)$. This can also be expressed as: the probability of inverting $f(x)$ is "small".*

Who's trying to invert this function? We'll always assign this task to an adversary who we shall try to thwart. Let's make a couple of notes about the adversary.

Adversary's Resources: In practice, everyone has bounded computational resources. Therefore, it is reasonable to model the adversary as such an entity, instead of an all powerful entity. But based on such a choice, we arrive at two different notions of security:

Computational Security: Security against efficient adversaries. We will mostly focus on computational security throughout the course.

Information-theoretic Security: Security against inefficient adversaries.

Adversary's Strategy: The adversary is not restricted to any specific strategies. We do not make any assumptions about adversarial strategy. Adversary can use its bounded computational resources however intelligently it likes.

Turing Machines can capture all types of computations that are possible. Hence, our adversary will be a computer program or an algorithm, modeled as a Turing Machine (see Section 2.7). Our adversary will also be efficient (captured via its running time). We will give the adversary additional power of randomness and non-uniformity and consider adversaries to by non-uniform PPT Turing Machines.

When you see a definition that talks about probability, always ask yourself what the probability is over and where the randomness in the probability comes from.

Notice above that we said the *probability of inverting $f(x)$ is small.* What is this probability even over? We've talked about modeling our adversaries as PPT machines with access to a random tape. So a natural idea would be to consider the probability over the adversary's random tape. But if you think about it a little, this isn't a good idea. A non-uniform adversary can find the best available random tape and just hardwire that into the description.

To avoid the adversary having control over the randomness, we instead take the probability over the choice of x. Now the intuition for f to be a one-way function is the following: take any non-uniform PPT adversary, with the best possible strategy, the probability that $\mathcal{A}$ inverts $f(x)$ for a randomly chosen x from the input should be small. So we sample an x at random, and then we compute $f(x)$ and give it to the adversary, and we observe the probability that it inverts that image.

Using this informal description, we will make an attempt to describe the 2 conditions in a more formal way.

3.2 Formal Definition of One-way Functions

With the intuition described in the previous section, let's write down our first attempt at a one-way function.

Definition 10 (One-way Function (1st Attempt)). *A function $f : \{0,1\}^* \mapsto \{0,1\}^*$ is a one way function (OWF) if it satisfies the following two conditions:*

1. **Easy to compute**: *There is a PPT algorithm C s.t. $\forall x \in \{0,1\}^*$,*

$$\Pr[r \leftarrow\$ \{0,1\}^m \ : \ C(x;r) = f(x)] = 1,$$

where the probability of success for C to output the correct value $f(x)$ on input x, is taken over the random coins used by C to compute $f(x)$. For simplicity, we denote here the number of random coins by m.

2. **Hard to invert**: *For every non-uniform PPT adversary $\mathcal{A}$, for any input length $n \in \mathbb{N}$.*

$$\Pr[x \leftarrow\$ \{0,1\}^n : \mathcal{A} \textit{ inverts } f(x)] \leq \textit{small}$$

We need to specify what exactly "small" means. To this end, we will define a fast decaying function $\nu(\cdot)$ s.t for any imput length $n \in \mathbb{N}$ this function decays asymptotically faster than any inverse polynomial. We will call this function *negligible*. Let us formalize this below.

Definition 11 (Negligible function). *A function $\nu(\cdot)$ is negligible if for $\forall c \in \mathbb{N}$, $\exists n_0 \in \mathbb{N}$, such that $\forall n \geq n_0$,*

$$\nu(n) < \frac{1}{n^c}$$

As we stated earlier, a negligible function decays faster than all "inverse-polynomial" functions $\left(n^{-\omega(1)}\right)$. Examples of an obviously negligible function are exponentially decaying functions 2^{-n} or $n^{-\log(n)}$.

Updating our previous definition of one-way functions in view of our definition of negligible functions, we have our second attempt.

Definition 12 (One-way Function (2nd Attempt)). *A function $f : \{0,1\}^* \mapsto \{0,1\}^*$ is a one way function (OWF) if it satisfies the following two conditions:*

1. **Easy to compute**: *There is a PPT algorithm C s.t. $\forall x \in \{0,1\}^*$,*

$$\Pr[r \leftarrow\$ \{0,1\}^m \ : \ C(x;r) = f(x)] = 1.$$

2. **Hard to invert**: *For every non-uniform PPT adversary $\mathcal{A}$, for any input length $n \in \mathbb{N}$, there exists a negligible function $\nu(\cdot)$ such that*

$$\Pr[x \leftarrow\$ \{0,1\}^n : \mathcal{A} \textit{ inverts } f(x)] \leq \nu(|x|)$$

Although this definition seems accurate, it has one small problem: What is $\mathcal{A}$'s input? Let's take a closer look at this, and let $y := f(x)$. If f is a one way function, the following two conditions have to be satisfied by $\mathcal{A}$:

- **Condition 1**: $\mathcal{A}$ on input y must run in $\mathsf{poly}(|y|)$.
- **Condition 2**: $\mathcal{A}$ cannot output x' s.t. $f(x') = y$.

However, if the size of y is much smaller than the size of the domain, $\mathcal{A}$ cannot write the inverse even if it can find it. For example, consider the function

$$f(x) = \text{first } \log|x| \text{ bits of } x.$$

Although it is trivial to invert this function,

$$f^{-1}(y) = y || \underbrace{0000...0}_{n-\log n}$$

where $n = 2^{|y|}$, it still satisfies the definition for one-way function we've proposed. Let's see why this is true. Although f is easy to compute, $\mathcal{A}$ cannot invert f in time $\mathsf{poly}(|y|)$ as it needs $2^{|y|}$ to write down the answer. But this clearly a function we want to rule out as a one-way function. In order to fix this issue, we adopt the convention to always pad y and write $\mathcal{A}(1^n, y)$, so that $\mathcal{A}$ has sufficient time to write the answer.

Putting this all together we have the following final definition.

Definition 13 (One-way Function). *A function $f : \{0,1\}^* \mapsto \{0,1\}^*$ is a one way function (OWF) if it satisfies the following two conditions:*

1. **Easy to compute**: *There is a PPT algorithm C s.t. $\forall x \in \{0,1\}^*$,*

$$\Pr[r \leftarrow\$ \{0,1\}^m \ : \ C(x;r) = f(x)] = 1.$$

2. **Hard to invert**: *For every non-uniform PPT adversary $\mathcal{A}$, for any input length $n \in \mathbb{N}$, there exists a negligible function $\nu(\cdot)$ such that*

$$\Pr[x \leftarrow\$ \{0,1\}^n : \widetilde{x} \leftarrow \mathcal{A}(1^n, f(x)) : f(\widetilde{x}) = f(x)] \leq \nu(n)$$

As we shall see, it is instructive to represent the above definition in terms of game between a challenger, and an adversary.

From the above diagram it is clear that f is a one-way function, if for every adversary $\mathcal{A}$, the challenger outputs 1 with only negligible probability $\nu(n)$. When clear from the context, we shall avoid clutter sometimes and drop the 1^n in the challenger-adversary diagrams.

An important exercise to write out what it means for a function f to *not* be a one-way function. Specifically, we say that a function f is not a one-way function

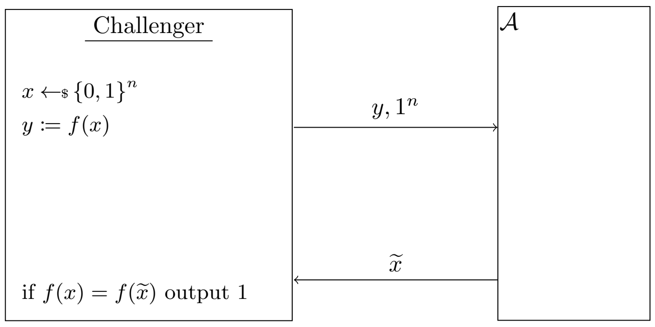

Challenger
$x \leftarrow_\$ \{0,1\}^n$
$y := f(x)$
$y, 1^n$
$\widetilde{x}$
if $f(x) = f(\widetilde{x})$ output 1
$\mathcal{A}$

Figure 3.1: One-way function challenge game

if there exists an adversary $\mathcal{A}$, a polynomial q such that for infinitely many values of n,

$$\Pr[x \leftarrow_\$ \{0,1\}^n : \widetilde{x} \leftarrow \mathcal{A}(1^n, f(x)) : f(\widetilde{x}) = f(x)] \geq \frac{1}{\mathsf{q}(n)}.$$

Remark 4. *Before we proceed, let's stop to look at the above phrasing which talks about* infinitely *many n. Why did we choose to phrase it as such. This goes back to our definition of negligible functions, for which we said must hold for all n greater than some n_0. If instead of infinitely many n, there were only a large, but finitely many n, we could always pick n_0 to be largest value in this set, and then this would not contradict the definition of a one-way function. So for a function to not be one-way, it must be the case that the above holds for infinitely many n.*

Extensions. In the chapter discussing preliminaries, we've already talked about injective or 1-1 functions, and permutations. The above definition for one-way functions extend naturally when the function f is additionally restricted to be an injective function, or a permutation.

3.3 Factoring Problem

Existence of one-way functions. Given that we're discussing one-way function, we should stop to ask if these objects even exist. It turns out that they don't exist unconditionally, and at the very least requires proving $\mathsf{P} \neq \mathsf{NP}$. However, by making certain assumptions about the hardness of some problems, we can construct conditional one-way functions also called "candidates". One such problem whose hardness is well studied is the Factoring Problem.

We will start with considering the **multiplication** function: $f_x : \mathbb{N} \times \mathbb{N} \mapsto \mathbb{N}$:

$$f_x(x, y) = \begin{cases} \perp & x = 1 \text{ or } y = 1 \\ x \cdot y & \text{otherwise} \end{cases}$$

This function is clearly not one-way, as the probability of xy being even and thus obviously factored into $(2, xy/2)$ is $\frac{3}{4}$ for random (x, y). In other words, the inversion succeeds 75% of time. We will then try to eliminate such trivial factors. We define $\prod_n$ be the set of all prime numbers $< 2^n$ and we randomly select two elements, p and q from this set and multiply them. Their product is thus unlikely to contain small trivial factors. Let's state our assumption formally below.

Assumption 1 (Factoring Assumption). *For every (non-uniform PPT) adversary $\mathcal{A}$, there exists a negligible function ν such that:*

$$\Pr[p \leftarrow\!\$\, \Pi_n ; q \leftarrow\!\$\, \Pi_n ; N := pq \ : \ \mathcal{A}(N) \in \{p, q\}] \leq \nu(n)$$

Of course, when we state an assumption, we must ask if it is a reasonable assumption. For the factoring assumption, up until now, there have been no "good attacks". The best known algorithms for breaking the Factoring Assumption are:

$$2^{O(\sqrt{n \log n})} \qquad \text{(provable)}$$
$$2^{O(\sqrt[3]{n \log^2 n})} \qquad \text{(heurestic)}$$

Looking back at our multiplication function, it is clear that if a random x and y happen to be prime, no $\mathcal{A}$ could invert it, which is the *good* case. If such a case happens with probability greater than ϵ, then every $\mathcal{A}$ must fail to invert the function with probability at least ϵ. If ϵ is a noticeable function, then $\mathcal{A}$ fails to invert the function with noticeable probability. This doesn't satisfy our requirement for a one-way function, but let us relax the notion slightly. This is what we shall call a *weak* one-way function. Before we define a *weak* one-way function, let us start by defining a noticeable function.

Definition 14 (Noticeable Function). *A function $\epsilon : \mathbb{N} \mapsto \mathbb{R}$ is noticeable if there exits $c \in \mathbb{N}$ and $n_0 \in \mathbb{N}$ such that $\forall n > n_0$:*

$$\epsilon(n) \geq \frac{1}{n^c}$$

Given the above definition, we have the following definition for a weak one-way function.

Definition 15 (Weak One-way Function). *A function $f : \{0,1\}^* \mapsto \{0,1\}^*$ is a one way function (OWF) if it satisfies the following two conditions:*

1. **Easy to compute**: *There is a PPT algorithm C s.t. $\forall x \in \{0,1\}^*$,*

$$\Pr[r \leftarrow\!\$ \{0,1\}^m \;:\; C(x;r) = f(x)] = 1.$$

2. **Somewhat hard to invert**: *There is a noticeable function $\epsilon : \mathbb{N} \mapsto \mathbb{R}$ s.t. for every non-uniform PPT adversary $\mathcal{A}$, for any input length $n \in \mathbb{N}$,*

$$\Pr[x \leftarrow\!\$ \{0,1\}^n : \widetilde{x} \leftarrow \mathcal{A}(1^n, f(x)) : f(\widetilde{x}) \neq f(x)] \geq \epsilon(n).$$

The reader should note the differences between this definition and our earlier definition of a *strong* one-way function.

Now we will try to show that $f_\times$ is indeed a weak OWF.

Theorem 2. *Assuming the factoring assumption, function $f_\times$ is a weak OWF.*

To prove this, we will show that the "good" case when x and y are prime occurs with noticeable probability, and to this end we will use Chebyshev's theorem to show that the fraction of prime numbers between 1 and 2^n is noticeable.

Theorem 3 (Chebyshev's theorem). *An n bit number is a prime with probability $\frac{1}{2n}$.*

Now we proceed to the proof.

Proof We're going to prove this using two different approaches. Our latter approach will be the more common approach of proof by contradiction.

Proof via definition : Let GOOD be the set of inputs (x,y), such that both x and y are prime. Then we have

$$\begin{aligned}\Pr[\mathcal{A} \text{ inverts } f_\times] = {} & \Pr[\mathcal{A} \text{ inverts } f_\times \mid (x,y) \in \mathsf{GOOD}] \cdot \Pr[(x,y) \in \mathsf{GOOD}] \\ & + \Pr[\mathcal{A} \text{ inverts } f_\times \mid (x,y) \notin \mathsf{GOOD}] \cdot \Pr[(x,y) \notin \mathsf{GOOD}]\end{aligned}$$

According to the Factoring Assumption, when $(x,y) \in \mathsf{GOOD}$, $\mathcal{A}$ could invert $f_\times$ with a probability no more than a negligible function $\nu(n)$. Using Chebyshev's theorem, an n bit number is a prime number with probability $\frac{1}{2n}$. Thus we get

$$\Pr[\mathcal{A} \text{ inverts } f_\times] \leq \nu(n) \cdot \frac{1}{4n^2} + 1 \cdot \left(1 - \frac{1}{4n^2}\right) = 1 - \frac{1}{4n^2}(1 - \nu(n))$$

Now we only need to prove that $\frac{1}{4n^2}(1 - \nu(n))$ is a noticeable function. Given that $\forall c > 0, \nu(n) \leq \frac{1}{n^c}$, we can conclude that for $n \geq 2$, $1 - \nu(n) \geq \frac{1}{n}$. Thus $\frac{1}{4n^2}(1 - \nu(n)) \geq \frac{1}{4n^3}$ is noticeable. Hence $f_\times$ is a weak OWF.

Now, let us prove the same statement via reduction to the factoring assumption.

Proof via reduction: Suppose that $f_\times$ is not a weak OWF, then we can construct an adversary to break the factoring assumption. Assume that there exists a non-uniform PPT algorithm $\mathcal{A}$ inverting $f_\times$ with probability at least $1 - \frac{1}{8n^2}$. That is

$$\Pr\Big[(x, y) \leftarrow\!\$\, \{0,1\}^n \times \{0,1\}^n, z := x \cdot y \;:\; \mathcal{A}(1^{2n}, z) \in f_\times^{-1}(z)\Big] \geq 1 - \frac{1}{8n^2}.$$

Now we construct a non-uniform adversary algorithm $\mathcal{B}$, which on input z (which is a product of two random n-bit prime numbers) uses $\mathcal{A}$ to break the factoring assumption. This is depicted pictorially below.

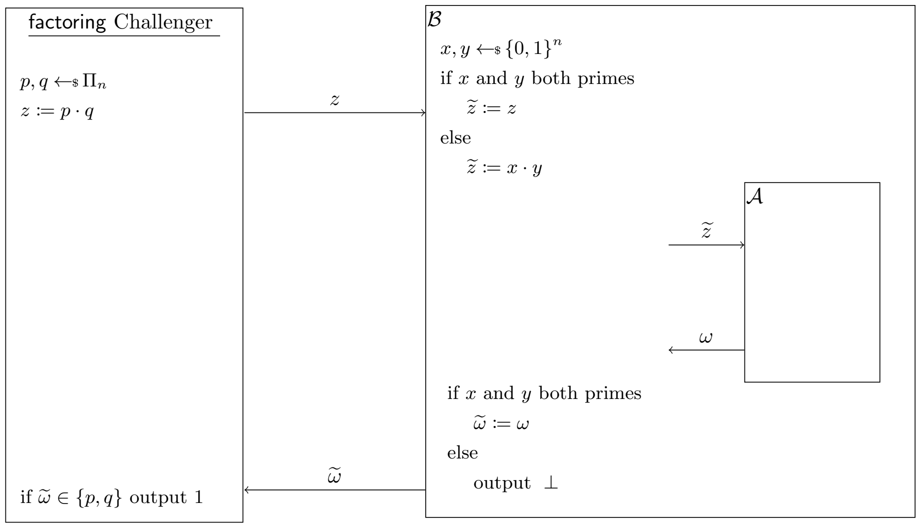

factoring Challenger
$p, q \leftarrow\!\$\, \Pi_n$
$z := p \cdot q$
z
$\mathcal{B}$
$x, y \leftarrow\!\$\, \{0,1\}^n$
if x and y both primes
$\widetilde{z} := z$
else
$\widetilde{z} := x \cdot y$
$\mathcal{A}$
$\widetilde{z}$
ω
if x and y both primes
$\widetilde{\omega} := \omega$
else
output $\perp$
$\widetilde{\omega}$
if $\widetilde{\omega} \in \{p, q\}$ output 1

Figure 3.2: Reduction to factoring assumption

Since this is our first proof by reduction, we've also written down the steps separately. In the future, we shall only use the above pictorial representation.

```
$\mathcal{B}(z)$
1 : Pick $(x, y)$ randomly from $\{0,1\}^n \times \{0,1\}^n$;
2 : if $x, y$ are both prime, let $\widetilde{z} = z$;
3 : else, let $\widetilde{z} = xy$;
4 : run $\omega = A(1^{2n}, \widetilde{z})$;
5 : if $x, y$ are both prime, set $\widetilde{\omega} := \omega$, and return $\widetilde{\omega}$.
6 : else output $\perp$ to indicate failure.
```

Remark 5. *The reason for randomly choosing (x, y) instead of passing the input directly to $\mathcal{A}$ is that, the input of $\mathcal{B}$ is a product of two random n-bit primes while that of $\mathcal{A}$ is the product of two random n-bit numbers. Passing the input directly to $\mathcal{A}$ would not emulate the uniform distribution of the inputs given to $\mathcal{A}$.*

Now we calculate the probability that $\mathcal{B}$ fails to break factoring assumption. We use the following notation:

$$\begin{aligned}\Pr[\mathcal{B}\text{ fails}] &= \Pr[\mathcal{B}\text{ passes input to }\mathcal{A}]\cdot\Pr[\mathcal{A}\text{ fails to invert }f_\times]\\ &\qquad+\Pr[\mathcal{B}\text{ fails to pass input to }\mathcal{A}]\\ &\le \Pr[\mathcal{A}\text{ fails to invert }f_\times]+\Pr[\mathcal{B}\text{ fails to pass input to }\mathcal{A}]\\ &\le \frac{1}{8n^2}+\left(1-\frac{1}{4n^2}\right)\le 1-\frac{1}{8n^2}\end{aligned}$$

Thus $\mathcal{B}$ breaks factoring assumption with a noticeable probability. And we get contraction. Thus $f_\times$ is a weak one-way function. □

3.4 Weak to strong One-way function

Once we have a weak one-way function, how do we get a strong one-way function? Can we transform the multiply function into a strong OWF? Can we do this generically? The answer is yes, and was proven by Yao.

Theorem 4 (Yao's Theorem). *Strong OWFs exist if and only if weak OWFs exist.*

In other words, if you have a weak one-way function, you can generically convert it to a strong one-way function.This is an example of a general phenomenon which is very well studied in complexity theory called *hardness amplification.*

Theorem 5. *For any weak OWF $f : \{0,1\}^n \mapsto \{0,1\}^n$, $\exists$ polynomial $N(\cdot)$ s.t. $F : \{0,1\}^{nN(n)} \mapsto \{0,1\}^{nN(n)}$ defined as*

$$F(x_1, ..., x_{N(n)}) = \Big(f(x_1), ..., f(x_{N(n)})\Big)$$

is a strong OWF.

Proof Since f is weak OWF, from Definition 15 we have $\mathsf{q} : \mathbb{N} \mapsto \mathbb{N}$ to be a polynomial function, such that for every non-uniform $\mathcal{A}$,

$$\Pr[x \leftarrow\!\$\, \{0,1\}^n, y \coloneqq f(x) \;:\; f(\mathcal{A}(1^n, y)) = y] \le 1 - \frac{1}{\mathsf{q}(n)}.$$

We want to find a N the right hand side of the above equation when repeatedly applied yields a negligible function. Specifically, we want an N such that $\left(1 - \frac{1}{\mathsf{q}(n)}\right)^N$ is negligible. One such value of N is $N \coloneqq 2n\mathsf{q}(n)$ which gives

$$\left(1 - \frac{1}{q(n)}\right)^N \approx \frac{1}{e^{2n}}.$$

Suppose that the defined F is not a strong OWF. Then by our definition of OWFs, $\exists$ polynomial function $\mathsf{p}'(\cdot)$ and a non-uniform adversary $\mathcal{A}$ s.t.

$$\begin{aligned}\Pr\Big[(x_1, ..., x_N) \leftarrow\!\$\, \{0,1\}^{nN}; y \coloneqq F(x_1, ..., x_N) \;:\; F\Big(\mathcal{A}(1^{nN}, y)\Big) = y\Big] \\ \ge \frac{1}{\mathsf{p}'(nN)} \\ = \frac{1}{\mathsf{p}(n)}\end{aligned}$$

This follows from the fact that N is a polynomial in n, therefore $\mathsf{p}'(nN)$ can be rewritten as a polynomial $\mathsf{p}(n)$ solely in n.

Now we shall use $\mathcal{A}$ to construct a non-uniform PPT $\mathcal{B}$ that breaks f with probability larger than $1 - \frac{1}{q(n)}$.

As a first step, we construct an intermediate adversary $\mathcal{B}_0$ on input $y = f(x)$ for random $x \in \{0,1\}^n$ as shown in Figure 3.3.

To improve the chance of inverting f, we construct $\mathcal{B}$ to run $\mathcal{B}_0$ several times using independently chosen random coins. We define $\mathcal{B} : \{0,1\}^n \mapsto \{0,1\}^n \cup \{\perp\}$ on input y to run $\mathcal{B}_0(y)$ for $2nN^2\mathsf{p}(n)$ times independently (i.e. choose x_j independently and randomly each time). $\mathcal{B}$ then outputs the first non-$\perp$ it receives. If all runs of $\mathcal{B}_0$ results in $\perp$, then $\mathcal{B}$ also outputs $\perp$.

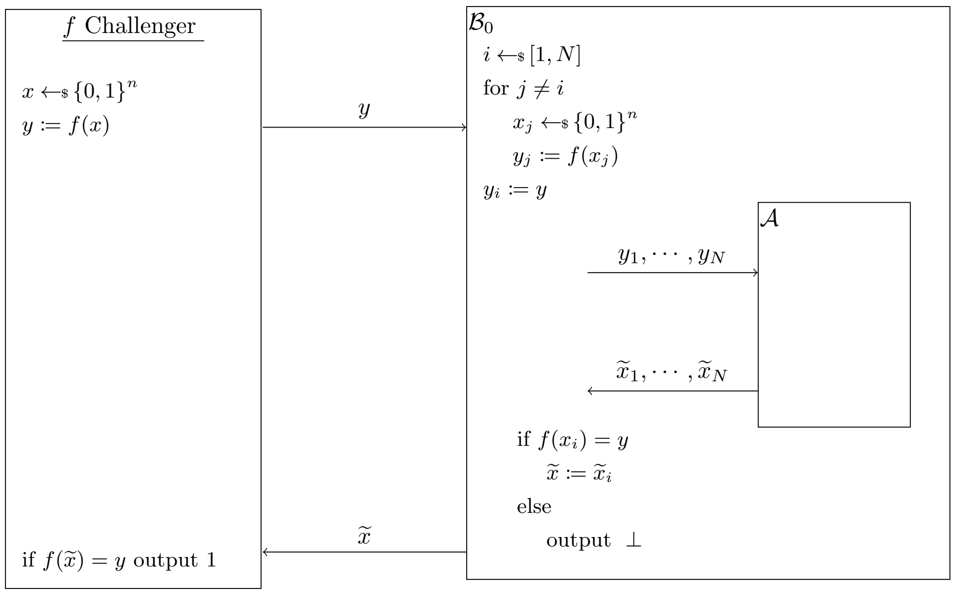


Figure 3.3: Construction of $\mathcal{B}_0$ from $\mathcal{A}$

Let GOOD be the set of xs that $\mathcal{B}_0$ inverts f with a probability at least $\frac{1}{2N^2\mathsf{p}(n)}$, i.e.,

$$\mathsf{GOOD} = \left\{ x \in \{0,1\}^n \,\middle|\, \Pr[f(\mathcal{B}_0(1^n, f(x))) = f(x)] \geq \frac{1}{2N^2\mathsf{p}(n)} \right\}.$$

Otherwise, we call x *bad.* Note that here the probability on the right side above is with respect to random coins used by $\mathcal{B}_0$.

The probability that $\mathcal{B}$ fails to invert f on $f(x)$ for x in GOOD is the probability that $\mathcal{B}$ fails on all $2nN^2\mathsf{p}(n)$ calls to $\mathcal{B}_0$, i.e.

$$\Pr[\mathcal{B}(f(x)) \text{ fails} \mid x \in \mathsf{GOOD}] \leq \left(1 - \frac{1}{2N^2\mathsf{p}(N)}\right)^{2nN^2\mathsf{p}(n)} \approx \frac{1}{e^n},$$

which is extremely small.
We prove that the fraction of GOOD set is noticeable.

Lemma 6. *There are at least* $2^n \cdot \left(1 - \frac{1}{2\mathsf{q}(n)}\right)$ *elements in* $\{0,1\}^n$ *in* GOOD.

Before we proceed to proving this Lemma, let's see how one would use it to complete the proof. We show that as long as the set GOOD is sufficiently large, $\mathcal{B}$ will succeed with high probability over the choice of x, alternatively that probability that $\mathcal{B}$ fails is low.

$$\begin{aligned}
\Pr[\mathcal{B}(f(x)) \text{ fails}] &= \Pr[\mathcal{B}(f(x)) \text{ fails} \mid x \in \mathsf{GOOD}] \cdot \Pr[x \in \mathsf{GOOD}] \\
&\quad + \Pr[\mathcal{B}(f(x)) \text{ fails} \mid x \notin \mathsf{GOOD}] \cdot \Pr[x \notin \mathsf{GOOD}] \\
&\leq \Pr[\mathcal{B}(f(x)) \text{ fails} \mid x \in \mathsf{GOOD}] + \Pr[x \notin \mathsf{GOOD}] \\
&\leq \left(1 - \frac{1}{2N^2\mathsf{p}(n)}\right)^{2N^2 n\mathsf{p}(n)} + \frac{1}{2\mathsf{q}(n)} \\
&\approx e^{-n} + \frac{1}{2\mathsf{q}(n)} \\
&< \frac{1}{\mathsf{q}(n)}
\end{aligned}$$

The first inequality comes from upper bounding the probability by 1. The second inequality comes from Lemma Lemma 6 and our earlier calculation of $\mathcal{B}$ failing to invert $f(x)$ when $x \in \mathsf{GOOD}$.

This is a contradiction to our assumption that f is $\mathsf{q}(n)$- weak. The only thing that remains to be proven is Lemma 6.

Proof (lemma 6) For the sake of contradiction, assume there are $> 2^n \cdot \left(\frac{1}{2q(n)}\right)$ elements not in GOOD. We shall show this violates the assumption that $\mathcal{A}$ inverts F with probability at least $\frac{1}{\mathsf{p}(n)}$. For simplicity of notation assume that that $\mathcal{A}$ outputs the symbol $\perp$ when it does not succeed, else it outputs the correct inverse. Also, we ignore the input 1^{nN} to $\mathcal{A}$.

$$\begin{aligned}
&\Pr\Big[(x_1, ..., x_N) \leftarrow\!\$\, \{0,1\}^{nN}; y := F(x_1, ..., x_N) \ : \ \mathcal{A}(y) \neq \perp\Big] \\
&\quad = \Pr\Big[(x_1, ..., x_N) \leftarrow\!\$\, \{0,1\}^{nN}; y := F(x_1, ..., x_N) \ : \ \mathcal{A}(y) \neq \perp \wedge (\forall i, x_i \in \mathsf{GOOD})\Big] \\
&\quad + \Pr\Big[(x_1, ..., x_N) \leftarrow\!\$\, \{0,1\}^{nN}; y := F(x_1, ..., x_N) \ : \ \mathcal{A}(y) \neq \perp \wedge (\exists i, x_i \notin \mathsf{GOOD})\Big]
\end{aligned}$$

For the first term, it suffices to bound the probability that *all* x_i are in GOOD. This is because $\mathcal{A}$ can at best invert with probability 1 in this case.

$$\begin{aligned}
&\Pr\Big[(x_1, ..., x_N) \leftarrow\!\$\, \{0,1\}^{nN}; y := F(x_1, ..., x_N) \ : \ \mathcal{A}(y) \neq \perp \wedge (\forall i, x_i \in \mathsf{GOOD})\Big] \\
&\qquad \leq \Pr\Big[(x_1, ..., x_N) \leftarrow\!\$\, \{0,1\}^{nN} \ : \ \forall i, x_i \in \mathsf{GOOD}\Big] \\
&\qquad \leq \left(1 - \frac{1}{2\mathsf{q}(n)}\right)^N = \left(1 - \frac{1}{2\mathsf{q}(n)}\right)^{2n\mathsf{q}(n)} \approx \frac{1}{e^n}
\end{aligned}$$

For the second term, fix some $j \in [N]$ and let us compute the probability that $\mathcal{A}$ inverts when $x_j \notin \mathsf{GOOD}$. Specifically, $\forall j \in [N]$:

$$\begin{aligned}
&\Pr\Big[(x_1, ..., x_N) \leftarrow\$ \{0,1\}^{nN}; y := F(x_1, ..., x_N) \ : \ \mathcal{A}(y) \neq \perp \wedge x_j \notin \mathsf{GOOD}\Big] \\
&\quad = \Pr\Big[(x_1, ..., x_N) \leftarrow\$ \{0,1\}^{nN}; y := F(x_1, ..., x_N) \ : \ \mathcal{A}(y) \neq \perp \ \Big| x_j \notin \mathsf{GOOD}\Big] \cdot \Pr[x_j \notin \mathsf{GOOD}] \\
&\quad \leq \Pr\Big[(x_1, ..., x_N) \leftarrow\$ \{0,1\}^{nN}; y := F(x_1, ..., x_N) \ : \ \mathcal{A}(y) \neq \perp \ \Big| x_j \notin \mathsf{GOOD}\Big] \\
&\quad \leq N \cdot \Pr[x_j \leftarrow\$ \{0,1\}^n \ : \ \mathcal{B}(f(x_j)) \neq \perp \ | \, x_j \notin \mathsf{GOOD}] \\
&\quad \leq \frac{N}{2N\mathsf{p}(n)} = \frac{1}{2\mathsf{p}(n)}
\end{aligned}$$

By taking union bound over all j, we have a bound for the second term,

$$\begin{aligned}
&\Pr\Big[(x_1, ..., x_N) \leftarrow\$ \{0,1\}^{nN}; y := F(x_1, ..., x_N) \ : \ \mathcal{A}(y) \neq \perp \wedge (\exists i, x_i \notin \mathsf{GOOD})\Big] \\
&\quad \leq \sum_{j=1}^{N} \Pr\Big[(x_1, ..., x_N) \leftarrow\$ \{0,1\}^{nN}; y := F(x_1, ..., x_N) \ : \ \mathcal{A}(y) \neq \perp \wedge x_j \notin \mathsf{GOOD}\Big] \\
&\quad \leq \frac{N}{2N\mathsf{p}(n)} = \frac{1}{2\mathsf{p}(n)}
\end{aligned}$$

Now putting everything together we have,

$$\begin{aligned}
&\Pr\Big[(x_1, ..., x_N) \leftarrow\$ \{0,1\}^{nN}; y := F(x_1, ..., x_N) \ : \ \mathcal{A}(y) \neq \perp\Big] \\
&\quad < \frac{1}{2\mathsf{p}(n)} + \frac{1}{e^n} \\
&\quad < \frac{1}{\mathsf{p}(n)}
\end{aligned}$$

This contradicts with the assumption that $\mathcal{A}$ is able to invert F with probability at least $\frac{1}{\mathsf{p}(n)}$. Thus it must be the case that the GOOD set has size at least $2^n \cdot \left(1 - \frac{1}{2\mathsf{q}(n)}\right)$.

□

This completes the proof. □

3.5 Final Remarks on OWFs

One-way functions are necessary for most of cryptography. Nevertheless, they are often not sufficient for most of cryptography. In particular, it is known that many advanced cryptographic primitives cannot be constructed by making *black-box* use of one-way functions; however, full separations are not known.

Also, recall that we don't know if one-way functions actually exist. (We only have candidates based on assumptions such as hardness of factoring.) Now, suppose someone told you one-way functions exist (perhaps by an existence proof, and not a constructive one). Then, simply using that knowledge, could you create an explicit one-way function? Surprisingly, it *can* be done! Levin gives a proof here [Lev03].

Chapter 4

Hard Core Predicate

4.1 Introduction

In this chapter, we will first examine what information one-way functions hide, and that will introduce us to the notion of hard core predicate.

The idea of a one-way function is very intuitive, but by themselves, they're often not very useful. Why? Because it only guarantees that $f(x)$ will hide the preimage x, but no more than that! For instance, if we have a one-way function $f : \{0,1\}^n \mapsto \{0,1\}^n$, then let us consider the following function $f' : \{0,1\}^{2n} \mapsto \{0,1\}^{2n}$,

$$f'(x_1||x_2) = f(x_1)||x_2,$$

where x_1 and x_2 are of size n. It is easy to see that f' is also a one-way function, but leaks half its input!

In fact, a function f may not hide any subset of the input bits, and still be a one-way function. More generally, for any non-trivial function $a(\cdot)$, there is no guarantee that $f(x)$ will hide $a(x)$. The question that naturally follows is,

> Are there non-trivial functions of x that $f(x)$ hides?

As a starting point, we'd be happy with such a function that outputs just a single bit.

Hard Core Predicate: Intuition. A hard core predicate for a one-way function f is a function over its inputs $\{x\}$ and its output is a single bit. We want this function to be easily computed given x, but "hard" to compute given $f(x)$.

Intuitively, this says that f can leak some (or many) bits of x, but does not leak the hard core bit. In other words, finding out the hard core bit, even *given* $f(x)$, is as difficult as inverting f. Of course, we need to be a little careful with "hard to compute" for a single bit, since it can be guessed correctly with probability $\frac{1}{2}$. So, intuitively, "hardness" should mean that it is impossible for any efficient adversary to gain any non-trivial advantage over guessing. We formalize this below:

Definition 16 (Hard Core Predicate). *Let $f : \{0,1\}^n \mapsto \{0,1\}^m$ be a one-way function. A predicate $h : \{0,1\}^* \mapsto \{0,1\}$ is a **hard core predicate** for $f(\cdot)$ if h is efficiently computable given x, and there exists a negligible function $\nu(\cdot)$ such that for every non-uniform PPT adversary $\mathcal{A}$, and $\forall n \in \mathbb{N}$,*

$$\Pr[x \leftarrow\$ \{0,1\}^n : \mathcal{A}(1^n, f(x)) = h(x)] \le \frac{1}{2} + \nu(n).$$

Here the randomness is over *both* the choice of x and the random coins used by $\mathcal{A}$. As before, we also present the above definition in the form of a game.

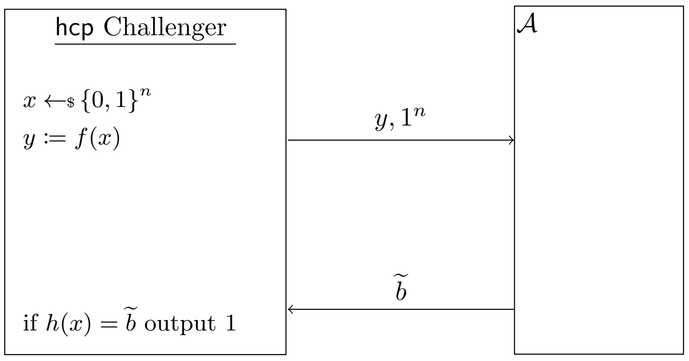


Figure 4.1: Hard-core predicate challenge game

In the above figure, we want it to hold for all non-uniform PPT $\mathcal{A}$ that hcp Challenger outputs 1 with some probability only negligible larger than $\frac{1}{2}$.

4.2 Hard Core Predicate via Inner Product

Ideally we would like every one-way function to have a hard core bit. But unfortunately, we do not know if this true. Instead, we settle for something slightly different. Namely, given *any* one-way function f, we show how to transform it into another function f' such that f' is one-way and has a hard core bit. We now describe this transformation using the inner product.

Theorem 7 (Goldreich-Levin). *Let $f : \{0,1\}^n \mapsto \{0,1\}^n$ be a one-way function (permutation). Then define $g : \{0,1\}^{2n} \mapsto \{0,1\}^{2n}$ as*

$$g(x) = f(x)||r$$

where $|x| = |r|$. Then, g is a one-way function (permutation) and

$$h(x,r) = \langle x, r \rangle$$

is a hard-core predicate for f, where $\langle x, r \rangle = (\sum_i x_i \cdot r_i) \mod 2$.

How should we prove this? If we use reduction, our main challenge is that our adversary $\mathcal{A}$ for h only outputs one bit, but our inverter $\mathcal{B}$ for f needs to output n bits. Amazingly, Goldreich and Levin proved that this can be done!

We start by considering two warmup cases, where we make assumptions on the adversary.

Assumption 2. *First, let's suppose that given $g(x,r) = f(x)||r$, adversary $\mathcal{A}$ will* always *(with probability 1) output $h(x,r)$ correctly.*

Proof We can use the property of the inner product to recover each bit of x one-by-one. For every $i \in [n]$, let e_i to be the ith standard basis vector for $\mathbb{R}^n$ (i.e., e_i is such that its ith bit is 1 but every other bit is 0). We construct an adversary $\mathcal{B}$ for f that works as follows: on input $f(x)$, It then runs the adversary $\mathcal{A}$ on input $(f(x), e_i)$ to recover x_i^*. Now, $\mathcal{B}$ simply outputs $x^* = x_1^* \cdots x_n^*$.

```
$\mathcal{B}(1^n, f(x))$
1 :  for $i = 1$ to $n$ do
2 :      $x_i^* \leftarrow \mathcal{A}(f(x), e_i)$
3 :  output $x^* = x_1^* \cdots x_n^*$
```

□

Okay, that was fairly straightforward. Let's see what happens when the adversary doesn't output $h(x,r)$ correctly with probability 1.

Assumption 3. *Now assume $\mathcal{A}$ outputs $h(x,r)$ with probability $\frac{3}{4} + \varepsilon(n)$.*

Proof (Informal Strategy) The main issue here is that our adversary $\mathcal{A}$ might not work on some specific inputs - such as e_i as in the previous case. We need the inputs to "look" random in order to mimic the expected distribution over the inputs of $\mathcal{A}$. So, we split our original single query into two queries such that each query looks random individually. Specifically, our inverter $\mathcal{B}$ computes $a := \mathcal{A}(f(x), e_i + r)$ and $b := \mathcal{A}(f(x), r)$ for $r \leftarrow\$ \{0,1\}^n$. Then, compute $c := a \oplus b$, where $\oplus$ is xor. By using a union bound, one can show that $c = x_i$ with probability $\frac{1}{2} + \varepsilon$, and we can then repeatedly compute x_i and take the majority to get x_i^* with $x_i^* = x_i$ with probability $1 - \mathsf{negl}(n)$.

By repeating this process for every i, $\mathcal{B}$ can learn every x_i^* and output $x^* = x_1^* \cdots x_n^*$.

□

The full proof of Theorem 7 follows the same ideas, but needs a more careful analysis and is skipped here. Note that this theorem is actually very important, even outside cryptography! It has applications to learning, list-decoding codes, etc.

Chapter 5

Pseudorandomness

5.1 Introduction

Our computers use randomness every day, but what exactly is randomness? How does your computer get this randomness? Some common sources of randomness are key-strokes, mouse movement, and power consumption, but the amount of randomness generated by these isn't a lot, and often, especially in the context of cryptography, a lot of randomness is required. We shall see concrete instances of its usage in the subsequent chapters.

This brings us to the fundamental question: Can we "expand" a few random bits into many random bits? There are many heuristic approaches to this, but this isn't good enough for cryptography. We need bits that are *"as good as truly random bits"* (to a PPT adversary). This isn't very precise, so let's define it formally.

5.2 Computational Indistinguishability and Prediction Advantage

Suppose we have n uniformly random bits, $x = x_1 \| \cdots \| x_n$, and we want to find a deterministic polynomial time algorithm G that outputs $n+1$ bits: $y = y_1 \| \cdots \| y_{n+1}$ and looks *"as good as"* a truly random string $r = r_1 \| \cdots \| r_{n+1}$. We call such a $G : \{0,1\}^n \mapsto \{0,1\}^{n+1}$ a **pseudorandom generator**, or PRG for short.

But what does *"as good as"* really mean? Intuitively it means that there should be no obvious patterns and that it should pass all statistical tests a truly random string would pass (all possible k-length substrings should occur equally). But the key point is that we only need to address our adversary, so as long as there is no efficient test that can tell $G(x)$ and r apart, then that is enough!

This gives us the notion of **computational indistinguishability** of $\{x \leftarrow\$ \{0,1\}^n : G(x)\}$ and $\{r \leftarrow\$ \{0,1\}^{n+1} : r\}$. We will use this notion and an equivalent one called prediction advantage in order to define pseudorandomness. Then, we will devise a complete test for pseudorandom distributions (next-bit prediction), and examine pseudorandom generators.

First, we will have to define some terms.

Definition 17 (Ensemble). *A sequence $\{X_n\}_{n\in\mathbb{N}}$ is called an **ensemble** if for each $n \in \mathbb{N}$, X_n is a probability distribution (Definition 3) over $\{0,1\}^*$.*

Typically, we will consider X_n to be a distribution over the binary strings $\{0,1\}^{\ell(n)}$, where $\ell(\cdot)$ is a polynomial.

Now, let us try to define computational indistinguishability. This captures what it means for distributions X, Y to "look alike" to any efficient test. In other words, no non-uniform PPT "distinguisher" algorithm $\mathcal{D}$ can tell X and Y apart, or the behavior of $\mathcal{D}$ on X and Y is the same.

We can try to think of this as a game of sorts. Let's say we give $\mathcal{D}$ a sample of X. Then, $\mathcal{D}$ gains a point if it says the sample is from X, and loses a point if it says the sample is from Y. Then we can encode $\mathcal{D}$'s output with one bit. In order for X and Y to be indistinguishable, $\mathcal{D}$'s average score on a sample of X should be basically the same as its average score on a sample of Y.

$$\Pr[x \leftarrow X;\ \mathcal{D}(1^n, x) = 1] \approx \Pr[y \leftarrow Y;\ \mathcal{D}(1^n, y) = 1]$$

or

$$\left| \Pr[x \leftarrow X;\ \mathcal{D}(1^n, x) = 1] - \Pr[y \leftarrow Y;\ \mathcal{D}(1^n, y) = 1] \right| \le \mu(n)$$

for negligible $\mu(\cdot)$.

It should be noted that in our definition, the distinguisher $\mathcal{D}$ will only get a *single* sample. This brings us to the formal definition of **computational indistinguishability**.

Definition 18 (Computational Indistinguishability). *Two ensembles of probability distributions $X = \{X_n\}_{n\in\mathbb{N}}$ and $Y = \{Y_n\}_{n\in\mathbb{N}}$ are said to be* computationally indistinguishable *if for every non-uniform PPT distinguisher $\mathcal{D}$ there exists a negligible function $\nu(\cdot)$ such that*

$$\left| \Pr[x \leftarrow X;\ \mathcal{D}(1^n, x) = 1] - \Pr[y \leftarrow Y;\ \mathcal{D}(1^n, y) = 1] \right| \le \nu(n).$$

We can see that this formalizes the notion of a PRG if we let X be the distribution over the PRG outputs and Y be the uniform distribution over strings of the same length as PRG outputs.

But, there is actually another model for the same idea! If we give $\mathcal{D}$ a sample from either X or Y, and ask it to identify which distribution it is from, if $\mathcal{D}$ is not right with probability better than $\frac{1}{2}$, then X and Y look the same to it! Since any adversary can trivially win with probability $\frac{1}{2}$ by guessing without looking at the sample received, we want to quantify how well an adversary does beyond simply guessing. It will be convenient to change notation a bit and set $X^{(1)} = X, X^{(0)} = Y$.

Definition 19 (Prediction Advantage). *Prediction Advantage is defined as*

$$\max_{\mathcal{A}} \left| \Pr\left[b \leftarrow_{\$} \{0,1\}, t \leftarrow X_n^b : \mathcal{A}(1^n, t) = b \right] - \frac{1}{2} \right|.$$

As before, we want the prediction advantage to be negligible. Let us depict the above game pictorially as a game between a *Challenger* and an adversary $\mathcal{A}$. We say that $\mathcal{A}$ wins if the *Challenger* outputs value 1. From the prior discussion,

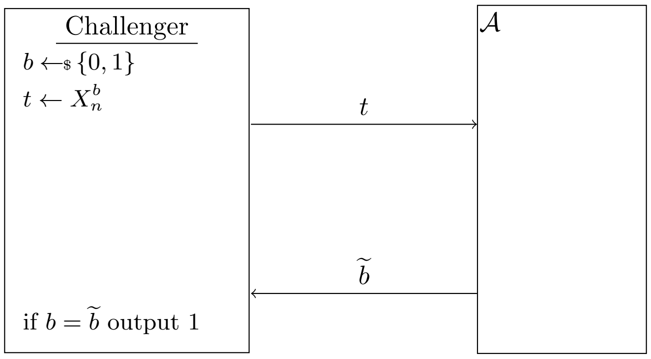


Figure 5.1: Indistinguishability Game between Challenger and adversary $\mathcal{A}$.

it is clear that probability that the challenger outputs 1 is,

$$\Pr[\text{ Challenger outputs 1}] = \Pr\left[b \xleftarrow{\$} \{0,1\}, t \leftarrow X_n^b : \mathcal{A}(t) = b \right].$$

Proposition 1. *Prediction advantage is equivalent to computational indistinguishability.*

Proof Let's write out the prediction advantage without taking the maximum over all $\mathcal{A}$,

$$\begin{aligned}
&\left|\Pr\left[b \leftarrow \{0,1\}; z \leftarrow X^{(b)}; \mathcal{D}(1^n, z) = b\right] - \frac{1}{2}\right| \\
&= \left|\Pr_{x \leftarrow X^b}\left[\mathcal{D}(x) = b \;\middle|\; b = 1\right] \cdot \Pr[b = 1] + \Pr_{x \leftarrow X^b}\left[\mathcal{D}(x) = b \;\middle|\; b = 0\right] \cdot \Pr[b = 0] - \frac{1}{2}\right| \\
&= \frac{1}{2}\left|\Pr_{x \leftarrow X^1}[\mathcal{D}(x) = 1] + \Pr_{x \leftarrow X^0}[\mathcal{D}(x) = 0] - 1\right| \\
&= \frac{1}{2}\left|\Pr_{x \leftarrow X^1}[\mathcal{D}(x) = 1] - \left(1 - \Pr_{x \leftarrow X^0}[\mathcal{D}(x) = 0]\right)\right| \\
&= \frac{1}{2}\left|\Pr_{x \leftarrow X^1}[\mathcal{D}(x) = 1] - \Pr_{x \leftarrow X^0}[\mathcal{D}(x) = 1]\right|
\end{aligned}$$

As before, a useful notion to consider for proofs is the notion of what is means for two distributions to *not* be computationally indistinguishable.

So they are equivalent within a factor of 2.

In the first step of the above calculations, we use the law of total probability (see Section 2.5 and the example for the law of total probability within). □

Lemma 8 (Prediction Lemma). *Let $\left\{X_n^{(0)}\right\}_{n\in\mathbb{N}}, \left\{X_n^{(1)}\right\}_{n\in\mathbb{N}}$ be ensembles of probability distributions. Let $\mathcal{D}$ be a non-uniform PPT adversary that $\varepsilon(\cdot)$-distinguishes $\{X_n^{(0)}\}, \{X_n^{(1)}\}$ for infinitely many $n \in \mathbb{N}$. Then $\exists$ a non-uniform PPT $\mathcal{A}$ such that*

$$\Pr\left[b \xleftarrow{\$} \{0,1\}, t \leftarrow X_n^b : \mathcal{A}(t) = b\right] - \frac{1}{2} \geq \frac{\varepsilon(n)}{2}$$

for infinitely many $n \in \mathbb{N}$.

Let's now look at some properties of computational indistinguishability.

Properties of Computational Indistinguishability.

1. First, we define the notation $\{X_n\} \approx_c \{Y_n\}$ to mean computational indistinguishability. Often, when clear from context we shall drop the subscript c and denote it by $\{X_n\} \approx \{Y_n\}$.
2. If we apply an efficient algorithm on X and Y, then their images under this operation are still indistinguishable. Formally, $\forall$ non-uniform PPT M, $\{X_n\} \approx_c \{Y_n\} \implies \{M(X_n)\} \approx_c \{M(Y_n)\}$. If this were not the case, then a distinguisher could simply use M to tell $\{X_n\}$ and $\{Y_n\}$ apart!

3. If X, Y are indistinguishable with advantage at most μ_1 and Y, Z are indistinguishable with advantage at most μ_2, then X, Z are indistinguishable with advantage at most $\mu_1 + \mu_2$. This follows from the triangle inequality, and we sketch the proof below.
   We are given, for all PPT $\mathcal{D}$

   $$\left|\Pr[x \leftarrow X;\ \mathcal{D}(1^n, x) = 1] - \Pr[y \leftarrow Y;\ \mathcal{D}(1^n, y) = 1]\right| \leq \mu_1(n)$$

   and

   $$\left|\Pr[y \leftarrow Y;\ \mathcal{D}(1^n, y) = 1] - \Pr[z \leftarrow Z;\ \mathcal{D}(1^n, z) = 1]\right| \leq \mu_2(n).$$

   Then we compute,

   $$\begin{aligned}
   &\left|\Pr[x \leftarrow X;\ \mathcal{D}(1^n, x) = 1] - \Pr[z \leftarrow Z;\ \mathcal{D}(1^n, z) = 1]\right| \\
   &= \Big|\Pr[x \leftarrow X;\ \mathcal{D}(1^n, x) = 1] - \Pr[y \leftarrow Y;\ \mathcal{D}(1^n, y) = 1] \\
   &\quad + \Pr[y \leftarrow Y;\ \mathcal{D}(1^n, y) = 1] - \Pr[z \leftarrow Z;\ \mathcal{D}(1^n, z) = 1]\Big| \\
   &\leq \left|\Pr[x \leftarrow X;\ \mathcal{D}(1^n, x) = 1] - \Pr[y \leftarrow Y;\ \mathcal{D}(1^n, y) = 1]\right| \\
   &\quad + \left|\Pr[y \leftarrow Y;\ \mathcal{D}(1^n, y) = 1] - \Pr[z \leftarrow Z;\ \mathcal{D}(1^n, z) = 1]\right| \\
   &= \mu_1(n) + \mu_2(n)
   \end{aligned}$$

   where we use the triangle inequality that $|a + b| \leq |a| + |b|$.

This last property is actually quite nice, and we would like to generalize it a bit. The proof follows from a simple application of the pigeonhole principle and triangle inequality.

Lemma 9 (Hybrid Lemma). *Let $X^1, \ldots, X^m$ be distribution ensembles for $m = \text{poly}(n)$. Suppose $\mathcal{D}$ distinguishes X^1 and X^m with advantage ε. Then $\exists i \in \{1, \cdots, m-1\}$ such that $\mathcal{D}$ distinguishes X^i, X^{i+1} with advantage $\geq \frac{\varepsilon}{m}$.*

The pigeonhole principle states that if there are n pigeons and m holes such that $n > m$, then there must be a hole with more than one pigeon. While this may

Try to see where the Hybrid Lemma uses the pigeonhole principle.

seem like an obvious fact, this principle has found extensive use in combinatorics and computer science.

Returning to pseudorandomness, we define a bit more notation. We call the uniform distribution over $\{0,1\}^{\ell(n)}$ by $U_{\ell(n)}$. Intuitively, a distribution is pseudorandom if it looks like a uniform distribution to any efficient test. We have the tools now to formulate this:

Definition 20 (Pseudorandom Ensembles). *An ensemble $\{X_n\}$, where X_n is a distribution over $\{0,1\}^{\ell(n)}$ is said to be pseudorandom if*

$$\{X_n\} \approx_c \{U_{\ell(n)}\}.$$

This is relevant for PRGs, as their outputs should be pseudorandom.

5.3 Next-Bit Test

In this section, we consider an interesting way to characterize pseudorandomness. For a string to be considered pseudorandom, we know that at the very least it must pass all efficient tests a true random string would pass. Let's consider one such natural test, which we shall call the "next bit unpredictability". For a truly random string, given any prefix of this string, it is not possible to predict the "next bit" with probability better than $\frac{1}{2}$. Let us extend this notion to an arbitrary string. We say a sequence of bits *passes the next-bit test* if no *efficient* adversary can predict "the next bit" in the sequence with probability better than $\frac{1}{2}$ even given all previous bits of the sequence. Let us formalize this below.

Definition 21 (Next-bit Unpredictability). *An ensemble of distributions $\{X_n\}$ over $\{0,1\}^{\ell(n)}$ is next-bit unpredictable if, $\forall\ 0 \leq i < \ell(n)$, $\forall$ non-uniform PPT adversaries $\mathcal{A}$, $\exists$ negligible function $\nu(\cdot)$ such that $\forall n \in \mathbb{N}$,*

$$\Pr\Big[t = t_1 \cdots t_{\ell(n)} \leftarrow X_n : \mathcal{A}(t_1 \cdots t_i) = t_{i+1}\Big] \leq \frac{1}{2} + \nu(n).$$

When presented with such a definition, an instructive exercise is to consider what it means for an ensemble to *not* be next-bit unpredictable. This will be useful when we discuss proof by reduction.

An ensemble $\{X_n\}$ is *not* next-bit unpredictable if there exists an index i, an adversary $\mathcal{A}_{\textsf{next-bit}}$ and a polynomial n such that for infinitely many n,

$$\Pr\Big[t = t_1 \cdots t_{\ell(n)} \leftarrow X_n : \mathcal{A}_{\textsf{next-bit}}(t_1 \cdots t_i) = t_{i+1}\Big] \geq \frac{1}{2} + \frac{1}{\mathsf{p}(n)}.$$

We will refer to $\frac{1}{\mathsf{n}}$ as the advantage $\mathcal{A}_{\mathsf{next\text{-}bit}}$ with respect to randomly guessing.

As before, the figure below illustrates a game for the above definition.

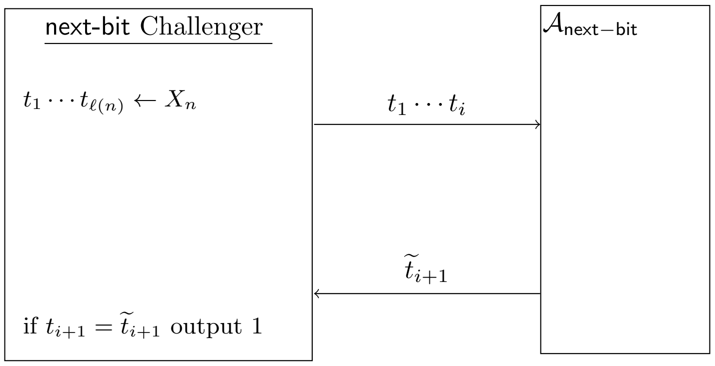

next-bit Challenger
$t_1 \cdots t_{\ell(n)} \leftarrow X_n$
$t_1 \cdots t_i$
$\widetilde{t}_{i+1}$
if $t_{i+1} = \widetilde{t}_{i+1}$ output 1
$\mathcal{A}_{\mathsf{next-bit}}$

Figure 5.2: Next bit unpredictability

This intuitive test turns out to be really powerful, as we show below. As we show, the next-bit test can in fact be used to test *pseudorandomness*.

Theorem 10 (Completeness of the Next-bit Test). *If $\{X_n\}$ is next-bit unpredictable then $\{X_n\}$ is pseudorandom.*

Proof This will be our first example where we use the Hybrid Lemma (Lemma 9) in a proof. This will be an invaluable tool through this course, and therefore we shall go over this proof carefully. In subsequent proofs, we might skip some of the details with the expectation that the reader will complete them.

Let us assume to the contrary that $\{X_n\}$ is pseudorandom. From Definition 20, we have that there exists a non uniform PPT distinguisher $\mathcal{D}$, and a polynomial $p(\cdot)$ s.t. for infinitely many $n \in \mathbb{N}$, $\mathcal{D}$ distinguishes X_n and $U_{l(n)}$ with probability $\frac{1}{p(n)}$. Our goal will be to use the above distinguisher D to construct an adversary $\mathcal{A}_{\mathsf{next-bit}}$ for the next-bit unpredictability.

Let us consider the following distributions, that we shall refer to as **hybrid distributions** or **hybrids** for short. For $i \in [\ell(n)]$, we define

$$\mathsf{H}_n^i := \left\{ x \leftarrow X_n, u \leftarrow U_{\ell(n)} : x_1 ... x_i u_{i+1} u_{i+2} ... u_{\ell(n)} \right\}$$

where $U_{\ell(n)}$ is the uniform distribution.

Note that the first hybrid H_n^0 is the uniform distribution $U_{\ell(n)}$, and the last hybrid $\mathsf{H}_n^{\ell(n)}$ is the distribution X_n. We show the transition of the hybrids pictorially in Figure 5.3, where the darker color indicates a random bit, and the lighter a pseudorandom bit.

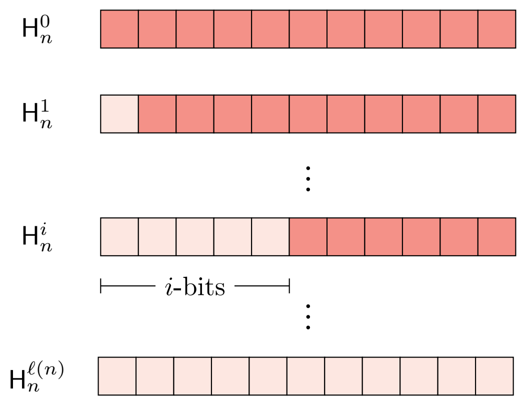


Figure 5.3: Progression of Hybrids

Think of why this is true. What would happen if it for all values of $i \in [\ell(n)]$, the probability was smaller than $\frac{1}{p(n)\ell(n)}$?

Thus, rewriting our initial claim regarding distinguisher $\mathcal{D}$ with respect to the hybrid distributions, $\mathcal{D}$ distinguishes between H_n^0 and $\mathsf{H}_n^{\ell(n)}$ with probability $\frac{1}{p(n)}$.

By the hybrid lemma (Lemma 9), $\exists$ some $i \in [0, \ell(n)]$ s.t. $\mathcal{D}$ distinguishes between H_n^i and H_n^{i+1} with probability at least $\frac{1}{p(n)\ell(n)}$.

The only difference between H^{i+1} and H^i is that in H^{i+1}, $(i+1)^{th}$ bit is x_{i+1}, and in H^i, it is u_{i+1}.

Let's define another distribution $\tilde{\mathsf{H}}_n^i$:

$$\tilde{\mathsf{H}}_n^i = \left\{ x \leftarrow X_n, u \leftarrow U_{\ell(n)} : x_1 ... x_{i-1} \bar{x}_i u_{i+1} u_{i+2} ... u_{\ell(n)} \right\},$$

where $\bar{x_i} = 1 - x_i$. This is identical to H_n^{i+1} with the $i+1$-th bit flipped.

Since H_n^{i+1} can be sampled from either H_n^i or $\tilde{\mathsf{H}}_n^i$ with equal probabilities,

$$\begin{aligned}
&\left|\Pr\left[t \leftarrow \mathsf{H}_n^i : \mathcal{D}(t) = 1\right] - \Pr[t \leftarrow \mathsf{H}_n^{i+1} : \mathcal{D}(t) = 1]\right| \\
&= \left|\Pr\left[t \leftarrow \mathsf{H}_n^{i+1} : \mathcal{D}(t) = 1\right] - \right. \\
&\qquad \left. \left(\frac{1}{2}\Pr\left[t \leftarrow \mathsf{H}_n^i : \mathcal{D}(t) = 1\right] + \frac{1}{2}\Pr\left[t \leftarrow \tilde{\mathsf{H}}_n^i : D(t) = 1\right]\right)\right| \\
&= \frac{1}{2}\left|\Pr\left[t \leftarrow \mathsf{H}_n^i : \mathcal{D}(t) = 1\right] - \Pr\left[t \leftarrow \tilde{\mathsf{H}}_n^i : \mathcal{D}(t) = 1\right]\right|.
\end{aligned}$$

The observation that $\mathcal{D}$ distinguishes H_n^i and H_n^{i+1} with probability $\frac{1}{p(n)\ell(n)}$ implies that $\mathcal{D}$ distinguishes H_n^{i+1} and $\tilde{\mathsf{H}}_n^{i+1}$ with probability $\frac{2}{p(n)\ell(n)}$. We shall use $\mathcal{D}$ to construct an adversary $\mathcal{A}_{\mathsf{next-bit}}$ for the next-bit Challenger. $\mathcal{A}_{\mathsf{next-bit}}$

on receiving input $t_1 \cdots t_i$ from the next-bit Challenger, needs to prepare the input for $\mathcal{D}$. It uses $t_1 \cdots t_i$, samples a random bit b, and the rest of the string is randomly generated. If $\mathcal{D}$ responds with 1, $\mathcal{A}_{\mathsf{next-bit}}$ guesses the next bit as b, else it guesses $1 - b$. This strategy is presented in Figure 5.4. Now we need to calculate the

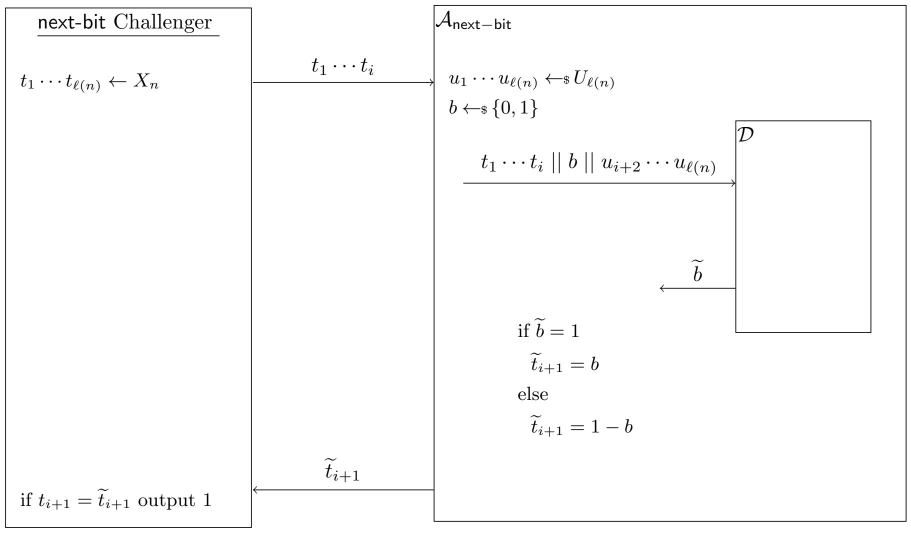

next-bit Challenger
$t_1 \cdots t_{\ell(n)} \leftarrow X_n$
$t_1 \cdots t_i$
$\mathcal{A}_{\mathsf{next-bit}}$
$u_1 \cdots u_{\ell(n)} \leftarrow\$ U_{\ell(n)}$
$b \leftarrow\$ \{0,1\}$
$\mathcal{D}$
$t_1 \cdots t_i \,||\, b \,||\, u_{i+2} \cdots u_{\ell(n)}$
$\widetilde{b}$
if $\widetilde{b} = 1$
$\widetilde{t}_{i+1} = b$
else
$\widetilde{t}_{i+1} = 1 - b$
$\widetilde{t}_{i+1}$
if $t_{i+1} = \widetilde{t}_{i+1}$ output 1

Figure 5.4: Completeness of Next-bit unpredictability

probability that $\mathcal{A}_{\mathsf{next-bit}}$ wins.

$$\begin{aligned}
\Pr[\mathcal{A}_{\mathsf{next-bit}} \text{ wins }] &= \frac{1}{2} \cdot \Pr\Big[t \leftarrow \mathsf{H}_n^i : \mathcal{D}(t) = 1\Big] + \frac{1}{2} \cdot \Pr\Big[t \leftarrow \widetilde{\mathsf{H}}_n^i : \mathcal{D}(t) \neq 1\Big] \\
&= \frac{1}{2} \cdot \Pr\Big[t \leftarrow \mathsf{H}_n^i : \mathcal{D}(t) = 1\Big] + \frac{1}{2} \cdot \Big(1 - \Pr\Big[t \leftarrow \widetilde{\mathsf{H}}_n^i : \mathcal{D}(t) = 1\Big]\Big) \\
&= \frac{1}{2} + \frac{1}{2} \cdot \Big(\Pr\Big[t \leftarrow \mathsf{H}_n^i : \mathcal{D}(t) = 1\Big] - \Pr\Big[t \leftarrow \widetilde{\mathsf{H}}_n^i : \mathcal{D}(t) = 1\Big]\Big) \\
&= \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{\mathsf{p}(n)\ell(n)}
\end{aligned}$$

This follows from the fact that with probability $\frac{1}{2}$ we will provide $\mathcal{D}$ with input either from H_n^i or $\widetilde{\mathsf{H}}_n^i$. We have constructed an adversary $\mathcal{A}_{\mathsf{next-bit}}$ that breaks our assumption that $\{X_n\}$ is next-bit unpredictable. Therefore it must be the case that $\{X_n\}$ is pseudorandom. □

We can now rely on next bit unpredictability to prove distributions are pseudorandom.

5.4 Pseudorandom Generators (PRG)

We have defined the notion of pseudorandomness and next-bit unpredictability. Now, we turn to our goal of constructing pseudorandom generators, that will take a small amount of randomness and expand it into a large amount of pseudorandomness.

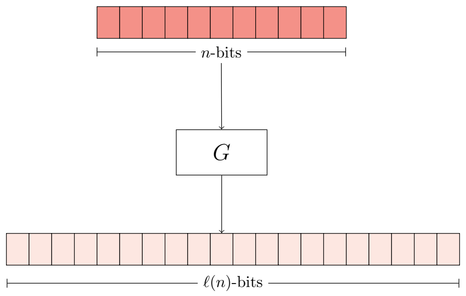


Figure 5.5: Pseudorandom Generator G

Before we can proceed with a construction, we need to define the properties we need from a pseudrandom generator.

Definition 22 (Pseudorandom Generator). *A deterministic algorithm* $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ *is called a pseudorandom generator (PRG) if:*

1. *(efficiency): the output of* G *can be computed in polynomial time*
2. *(expansion):* $|G(x)| > |x|$
3. *(pseudorandomness):* $\{x \leftarrow\!\$\, \{0,1\}^n : G(x)\} \approx_c \{U_{\ell(n)}\}$ *where* $\ell(n) = |G(0^n)|$

A few remarks about the above definition are in order:

- **Deterministic:** It is clear that the above definition is only meaningful if G is deterministic since any additional randomness can otherwise be directly output by G.
- **Seed:** The input to the PRG is referred to as the *seed* of the PRG.
- **Randomness of the seed:** Similar to the one-way functions, the security of a PRG is defined only when the seed is chosen uniformly at random.

To evaluate the "effectiveness" of a PRG, we define the *stretch* of the PRG below.

Definition 23 (Stretch). *The stretch of* G *is defined as:* $|G(x)| - |x|$

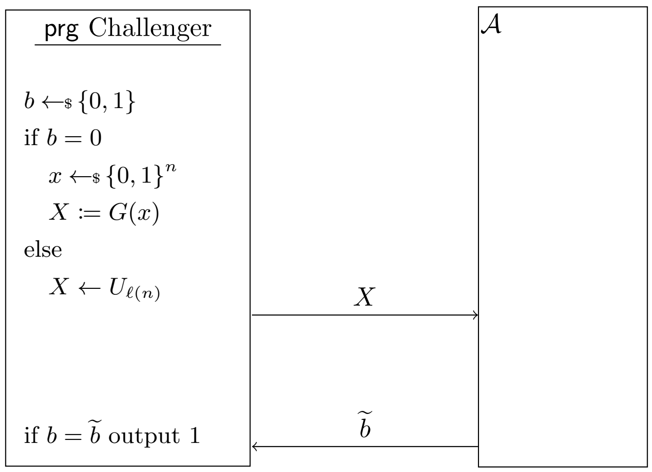

prg Challenger
$\mathcal{A}$
$b \leftarrow_\$ \{0,1\}$
if $b = 0$
$x \leftarrow_\$ \{0,1\}^n$
$X := G(x)$
else
$X \leftarrow U_{\ell(n)}$
X
$\widetilde{b}$
if $b = \widetilde{b}$ output 1

Figure 5.6: Indistinguishability Game for PRG.

We will first construct a PRG by 1-bit stretch, which already seems non trivial. Then, we will show how to generically transform a PRG with 1-bit stretch into one that achieves polynomial-bit stretch.

5.5 PRG with 1-bit Stretch

Armed with what we've learned so far, a natural candidate for our construction of PRG is

$$G(s) = f(s)||h(s)$$

where f is a one-way function and h is the hardcore predicate associated with f.

While $h(s)$ is indeed unpredictable even given $f(s)$, this construction is not really a PRG because of the following reasons:

- $|f(s)|$ might be less than $|s|$.
- $f(x)$ may always start with a prefix, which is not random. Indeed, OWF doesn't promise random outputs.

It turns out that PRGs can in fact be constructed from one-way functions, but the construction is quite involved. Instead, we'll settle on a slightly easier problem of constructing PRG from a one-way *permutation* (OWP) over $\{0,1\}^n$.

A one-way permutation clearly address both of the above issues:

- Since f is a permutation, the domain and range have the same number of bits, i.e., $|f(s)| = |s| = n$.

– $f(s)$ is uniformly random (not just pseudorandom) over $\{0,1\}^n$ if s is randomly chosen. In particular $\forall y$:

$$\Pr[s \leftarrow\$ \{0,1\}^n : f(s) = y] = \Pr\Big[s \leftarrow\$ \{0,1\}^n : s = f^{-1}(y)\Big] = \frac{1}{2^n}.$$

Thus, if s is uniform, $f(s)$ is uniform.

As it turns out, our initial proposed construction for works when f is a one-way permutation. Let us now prove the following theorem:

Theorem 11 (PRG based on OWP). *Let f be a one-way permutation, and h be a hard-core predicate for f. Then $G(s) = f(s)||h(s)$ is a PRG with 1-bit stretch.*

Proof Let us assume, to the contrary, that G is not a PRG. Then from Definition 22, we know that the output of G is not pseudorandom. Since we've established that the next-bit unpredictability is complete (Theorem 10), if the output of G is not pseudorandom, it must not satisfy next-bit unpredictability.

Putting this all together, if G is not a PRG, there exists an index i, a non-uniform PPT adversary $\mathcal{A}_{\mathsf{next-bit}}$, a polynomial $\mathsf{p}(\cdot)$ such that for infinitely many values of n,

$$\Pr[s \leftarrow\$ \{0,1\}^n, y_1 \cdots y_n y_{n+1} := G(s) : \mathcal{A}_{\mathsf{next-bit}}(y_1 \cdots y_i) = y_{i+1}] = \frac{1}{2} + \frac{1}{\mathsf{p}(\cdot)}.$$

Let us take a closer look at what values i can take. From our construction, we know that if s is random, the first n bits are random and no adversary can guess any of its bits with probability better than $\frac{1}{2}$. So it must be the case that $i = n$. We will use this information to construct an adversary $\mathcal{A}_{\mathsf{hcp}}$ for the hard-core predicate. The strategy is quite simple, when $\mathcal{A}_{\mathsf{hcp}}$ is given $y := f(x)$, this is passed along to $\mathcal{A}_{\mathsf{next-bit}}$ and the corresponding response from $\mathcal{A}_{\mathsf{next-bit}}$ is used as the response to the hcp Challenger. We describe this pictorially below:

From the description above it is clear that $\mathcal{A}_{\mathsf{hcp}}$ succeeds if and only if $\mathcal{A}_{\mathsf{next-bit}}$ succeeds. Therefore, the probability $\mathcal{A}_{\mathsf{hcp}}$ succeeds is given by

$$\Pr[\mathcal{A}_{\mathsf{hcp}} \text{ wins }] = \Pr[\mathcal{A}_{\mathsf{next-bit}} \text{ wins }] \geq \frac{1}{2} + \frac{1}{\mathsf{p}(n)}.$$

This contradicts the assumption that h is a hard-core predicate for f. Thus, it must be the case that G is a PRG. □

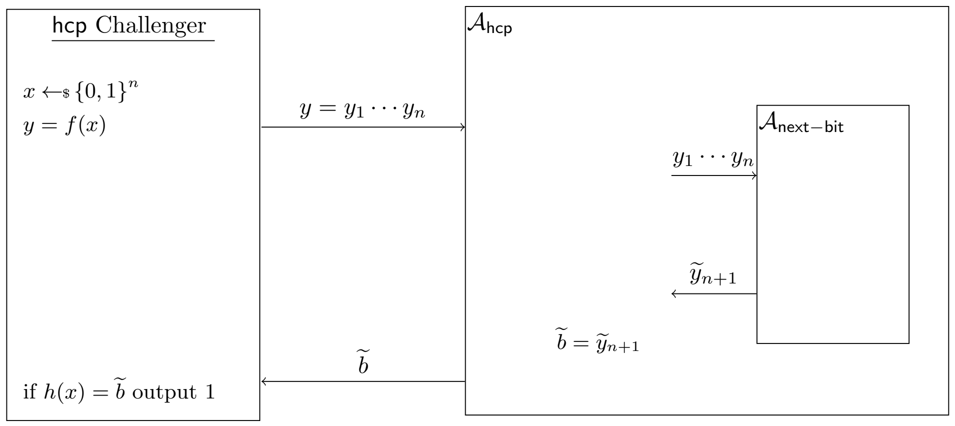

hcp Challenger
$x \leftarrow_\$ \{0,1\}^n$
$y = f(x)$
if $h(x) = \widetilde{b}$ output 1
$y = y_1 \cdots y_n$
$\widetilde{b}$
$\mathcal{A}_{\mathsf{hcp}}$
$\mathcal{A}_{\mathsf{next-bit}}$
$y_1 \cdots y_n$
$\widetilde{y}_{n+1}$
$\widetilde{b} = \widetilde{y}_{n+1}$

Figure 5.7: 1-bit Stretch PRG

5.6 PRG with Poly-Stretch

To be useful, we will need to go beyond a 1-bit stretch, to any arbitrary polynomial. From our previous section, why stop at 1 bit? Why don't we repeatedly apply the same procedure. Indeed, this will be how we will construct our PRG. See Figure 5.8 for visual representation of this idea, where we collect the last bit from each iteration and this forms the output of the PRG G. The first n bits from each output are fed into the next iteration of G.

We formally prove the lemma below.

Lemma 12. *Let $G : \{0,1\}^n \to \{0,1\}^{n+1}$ be a PRG. For any polynomial ℓ, $G' : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ is defined as:*

$$\begin{aligned} G'(s) &= b_1 ... b_{\ell(n)} \text{ where} \\ X_0 &:= s \\ X_{i+1} || b_{i+1} &:= G(X_i) \end{aligned}$$

Then, G' is a PRG.

Proof We first establish some notation. Let $G'(s) = G^{\ell(n)}(s)$, where

$$\begin{aligned} G^0(x) &= \epsilon \\ G^i(x) &= b || G^{i-1}(x') \text{ where } x' || b := G(x) \end{aligned}$$

and ϵ denotes the empty string.

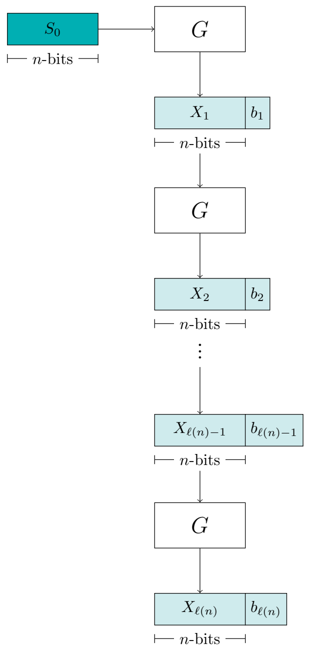


Figure 5.8: PRG with polynomial Stretch from PRG with a 1-bit Stretch.

Let us assume, to the contrary, that G' is not a PRG. Then, from Definition 22, there exists a distinguisher $\mathcal{D}$, and a polynomial $\mathsf{p}(\cdot)$ such that for infinitely many n, $\mathcal{D}$ distinguishes

$$\left\{U_{\ell(n)}\right\} \text{ and } \{G'(U_n)\}$$

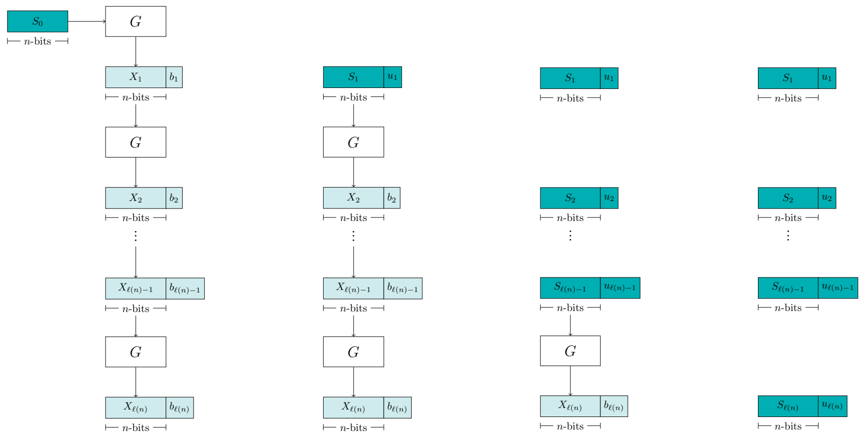
with non-negligible probability $\frac{1}{p(n)}$.

We shall use the Hybrid Lemma (Lemma 9) as the main tool for our proof. To do so, let us define the following hybrid distributions $\forall i \in [\ell(n)]$:

$$\mathsf{H}_n^i = U_i \| G^{\ell(n)-1}(U_n),$$

i.e. in hybrid H_n^i, the first i bits are random, while the rest are obtained as in the process above. Figure 5.9 gives a visual depiction of these hybrids. Then, $\mathsf{H}_n^0 = G^{\ell(n)}(U_n)$ and $\mathsf{H}_n^{\ell(n)} = U_{\ell(n)}$ and $\mathcal{D}$ distinguishes H_n^0 and $\mathsf{H}_n^{\ell(n)}$ with probability $\frac{1}{\mathsf{p}(n)}$.

By the hybrid lemma, for infinitely many n, there exists index i such that $\mathcal{D}$ distinguishes H_n^i and H_n^{i+1} with probability $\frac{1}{\ell(n)\mathsf{p}(n)}$. Since $\ell(n)$ is polynomial, so is $\ell(n)\mathsf{p}(n)$



(a) H_0 (b) H_1 (c) $\mathsf{H}_{\ell(n)-1}$ (d) $\mathsf{H}_{\ell(n)}$

Figure 5.9: Hybrids in

Note that the only difference between H_n^i and H_n^{i+1} can be seen by writing out the hybrids explicitly

$$\begin{aligned}
\mathsf{H}_n^i &= U_i \| G^{\ell(n)-i}(U_n) \\
&= U_i \| b \| G^{\ell(n)-i-1}(x) \\
\mathsf{H}_n^{i+1} &= U_{i+1} \| G^{\ell(n)-i-1}(U_n) \\
&= U_i \| U_1 \| G^{\ell(n)-i-1}(U_n)
\end{aligned}$$

where $x \| b := G(U_n)$. Thus, only the $i+1$-th bit either comes from a PRG output or is truly random. Now we shall use the adversary $\mathcal{D}$ to build an adversary $\mathcal{A}_{\mathsf{prg}}$ that breaks the pseudorandomness of G. $\mathcal{A}_{\mathsf{prg}}$ gets as input either a random $n+1$ bit string, or an output of a PRG of the same length. The strategy is to provide to $\mathcal{D}$ as input something from either H_n^i or H_n^{i+1} using the inputs $\mathcal{A}_{\mathsf{prg}}$ receives. From the construction, and the description of the hybrids, it is clear that we should use the input to $\mathcal{A}_{\mathsf{prg}}$ for the $i+1$-th bit for the input to $\mathcal{D}$. Formally, this is shown in the diagram below, If $b = 0$, then the input constructed by $\mathcal{A}_{\mathsf{prg}}$ for $\mathcal{D}$ is identical to H_n^i, while if $b = 1$, then it is identical to H_n^{i+1}. Therefore, the value $\widetilde{b}$ returned by $\mathcal{D}$ also indicates whether the input to $\mathcal{A}_{\mathsf{prg}}$ was output of a PRG or random. In fact $\mathcal{A}_{\mathsf{prg}}$ has the same probability of success as $\mathcal{D}$. Therefore,

$$\Pr[\mathcal{A}_{\mathsf{prg}} \text{ wins }] = \Pr[\mathcal{D} \text{ wins }] \geq \frac{1}{2} + \frac{1}{\ell(n)\mathsf{p}(n)}.$$

Since G runs in polynomial time, so does $\mathcal{A}_{\mathsf{prg}}$. Therefore our constructed $\mathcal{A}_{\mathsf{prg}}$

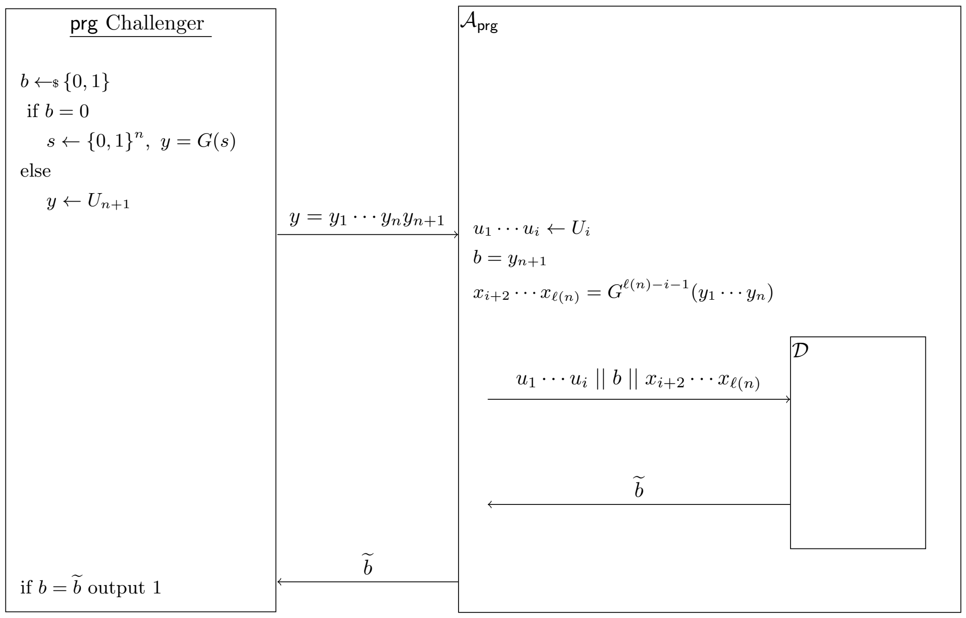

prg Challenger
$b \leftarrow_{\$} \{0,1\}$
if $b = 0$
$s \leftarrow \{0,1\}^n,\ y = G(s)$
else
$y \leftarrow U_{n+1}$
if $b = \tilde{b}$ output 1
$y = y_1 \cdots y_n y_{n+1}$
$\tilde{b}$
$\mathcal{A}_{\mathsf{prg}}$
$u_1 \cdots u_i \leftarrow U_i$
$b = y_{n+1}$
$x_{i+2} \cdots x_{\ell(n)} = G^{\ell(n)-i-1}(y_1 \cdots y_n)$
$\mathcal{D}$
$u_1 \cdots u_i \mid\mid b \mid\mid x_{i+2} \cdots x_{\ell(n)}$
$\tilde{b}$

Figure 5.10: 1-bit to poly-bit stretch PRG

What happened if we got greedy and tried to output say X_1? Would it still be secure?

contradicts the assumption that G is a PRG. It must then be the case that G' is a PRG.

□

5.7 Pseudorandom Functions (PRF)

Going beyond Poly Stretch. PRGs can only generate polynomially long pseudorandom strings. What if we want exponentially long pseudorandom strings? How can we efficiently generate them? One way to do this is by using functions that index exponentially long pseudorandom strings.

Towards that end, let us start by looking at the notion of a random function. Consider a function $f : \{0,1\}^n \mapsto \{0,1\}^n$. As we've seen in Example 2, the total number of functions that map n bits to n bits is 2^{n2^n}.
To define a random function, we can use one of the two methods:

1. Select a random function F *uniformly at random* from all 2^{n2^n} functions that map n bits to n bits
2. Use a randomized algorithm to describe the function. This is called *lazy sampling*, and often more convenient implementation for proofs. Specifically, we describe below a randomized program M that maintains a table

$\mathcal{T}$ to mimic a random function.

```
M(1^n)
1 : $\mathcal{T} = \emptyset$
2 : **input** $x$
3 : **if** $\exists y'$ s.t. $(x, y') \in \mathcal{T}$
4 :     **output** $y'$
5 : **else**
6 :     $y \leftarrow\$ \{0,1\}^n$
7 :     **add** $(x, y)$ **to** $\mathcal{T}$
8 :     **output** $y$
```

Note that the distribution of M's output is identical to that of a random function F.

Truly random functions are huge random objects. Neither of the methods allows us to store the entire function efficiently. But with the second method, M will only need polynomial space and time to store and query $\mathcal{T}$, if one is limited to making only *polynomial* calls to the random function.

We said we would use these functions to index exponentially long pseudorandom strings. We shall come to pseudorandom aspect shortly, but these terms make sense even for exponentially long random strings. We've seen that the size of function itself was large, it may not be immediately obvious what the exponentially long string or index corresponds to with respect to the functions. We will think of the exponentially long string to be all the concatenation of the output strings of the random function F, i.e. the string is $F(0)||F(1)|| \cdots ||F(2^n)$. Given that each output is n-bits, the size of the string is $n \cdot 2^n$. With this view, it is easy to see what the index corresponds to, it is simply the input to the function F, which returns an n bit chunk of the long string.

Now that we've seen a random function, what is a pseudorandom function (PRF). A PRF "looks" like a random function, but unlike a random function it can be described using only *polynomially* many bits. At first, it seems like a good idea to use computational indistinguishability to make PRF "look like" a random function. However, there are issues to this approach that we expand upon below:

Issue: Since the description of a random function is of exponential size, $\mathcal{D}$ might not even be able to read the input efficiently in case of random function and can easy tell the difference just be looking at the size of input.

Suggested Solution: What if $\mathcal{D}$ is not given the description of functions as input, but is instead allowed to only query the functions on inputs of its choice, and view the output. But the question here is whether the entire implementation of the PRF is hidden from the distinguisher or only a part of it.

Keeping the entire description of PRF secret from $\mathcal{D}$, is similar to providing security by obscurity which in general is not a good idea according to Kerchoff's principle. Therefore in accordance with the principle, we use the notion of keyed functions. It is better to define PRFs as keyed functions. So only the key remains secret while the PRF evaluation algorithm can be made public.

Since this is the first time we are talking about a keyed function/function family, we can think of functions $F(\cdot,\cdot)$ that take in two inputs, a key (or index of the family) k and the input x and outputs $F(k,x)$. To separate out the two types of inputs, we will find it convenient to represent keyed functions as $F_k(x)$. Once the key or index is fixed, the function $F_k(\cdot)$ behaves like the functions we have encountered so far. We will often interchangeably use the term 'keyed function' and function family.

Now we are finally ready to talk about the security of PRFs.

5.7.1 Security of PRFs via Game Based Definition

As we have seen previously, the security is described via a game between the PRF challenger and an Adversary/Distinguisher $\mathcal{D}$. The game is described in Figure 5.11. Note that unlike previous security games, here the adversary $\mathcal{D}$ can repeatedly ask queries x_i to the challenger before it decides to output the bit $\widetilde{b}$.

We say that $\mathcal{D}$ wins the game if the PRF Challenger outputs 1. Intuitively, we want that any adversary $\mathcal{D}$ wins with probability only negligibly greater than $\frac{1}{2}$. We now define the PRFs below.

Definition 24 (Pseudorandom Functions). *A family $\{F_k\}_{k\in\{0,1\}^n}$ of functions, where $F_k:\{0,1\}^n \to \{0,1\}^n$ for all k is* pseudorandom *if, it is:*

- ***Easy to Compute:*** *There is an efficient algorithm M such that $\forall k, x$: $M(k,x) = F_k(x)$.*
- ***Hard to Distinguish:*** *For every non-uniform PPT $\mathcal{D}$, there exists a negligible function ν such that $\forall n \in N$:*

$$Pr[\mathcal{D}\ wins] \leq \frac{1}{2} + \nu(n) \tag{5.1}$$

Remark 6. *We are ensuring that the challenger is efficient by implementing the random function (when $b = 1$) using the second method -* lazy sampling. *It is im-*

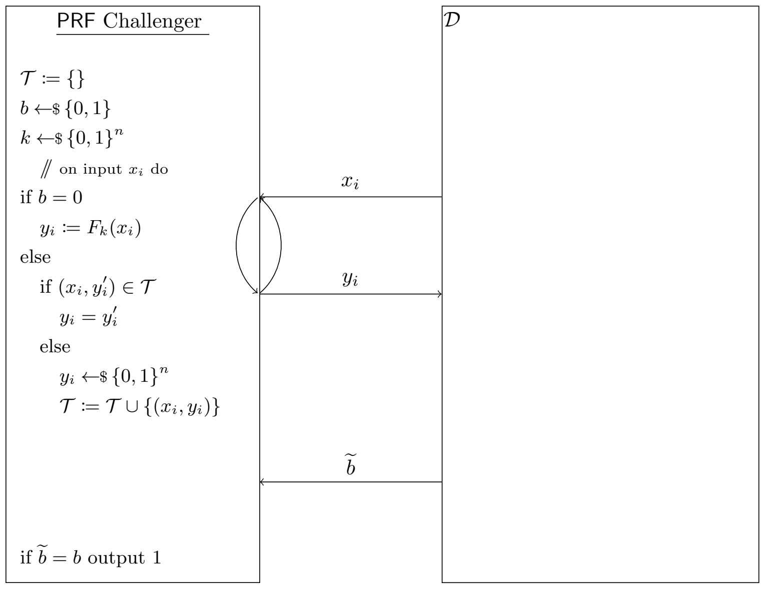

PRF Challenger
$\mathcal{T} \coloneqq \{\}$
$b \leftarrow\$ \{0,1\}$
$k \leftarrow\$ \{0,1\}^n$
// on input x_i do
if $b = 0$
$y_i \coloneqq F_k(x_i)$
else
if $(x_i, y_i') \in \mathcal{T}$
$y_i = y_i'$
else
$y_i \leftarrow\$ \{0,1\}^n$
$\mathcal{T} \coloneqq \mathcal{T} \cup \{(x_i, y_i)\}$
if $\widetilde{b} = b$ output 1
$\mathcal{D}$
x_i
y_i
$\widetilde{b}$

Figure 5.11: PRF Challenger Game

portant to construct challengers that are efficient. As we have seen while building reductions, the outer adversary acts like a challenger to the inner adversary. And if the challenger is not efficient, the outer adversary would not be efficient, and would thus lead to the construction of an inefficient *reduction. All of our definitions are defined to be secure only against non-uniform* PPT *adversaries, therefore in our reduction, to arrive at a contradiction the outer adversary needs to run in polynomial. So an inefficient outer adversary, i.e. one whose running time is not polynomial, would not give rise to the necessary contradiction.*

Now that we have our definition, the next step is construct a PRF using the tools we've developed so far. Let's start off with a simple warm-up example of PRF whose input is a single bit.

5.7.2 PRF with 1-bit input

Intuitively, PRFs with 1- bit input, can be constructed using PRGs. Since PRFs are keyed functions, the key of PRF can be used as the random seed for PRG. We can construct a PRF $F_k : \{0,1\} \mapsto \{0,1\}^n$ using a length doubling PRG G as follows: Let G be the length-doubling PRG such that $G(s) = y_0||y_1$ where $|y_0| = |y_1| = n$.

```
$F_k(x)$
1: $s := k$
2: $y_0||y_1 := G(s)$
3: **if** $x = 0$
4:     **output** $y_0$
5: **else**
6:     **output** $y_1$
```

Let's argue the security of this PRF construction by relying on the security of the underlying PRG.

Proof Let us assume that the above construction is *not* a PRF. Then there exists an adversary $\mathcal{D}_{\mathsf{PRF}}$ that can distinguish between the above construction and a random function F. We will use $\mathcal{A}_{\mathsf{PRF}}$ to construct a distinguisher $\mathcal{D}_{\mathsf{PRG}}$ that is able to distinguish between the output of the PRG G and a random string. The reduction is presented in Figure 5.12.

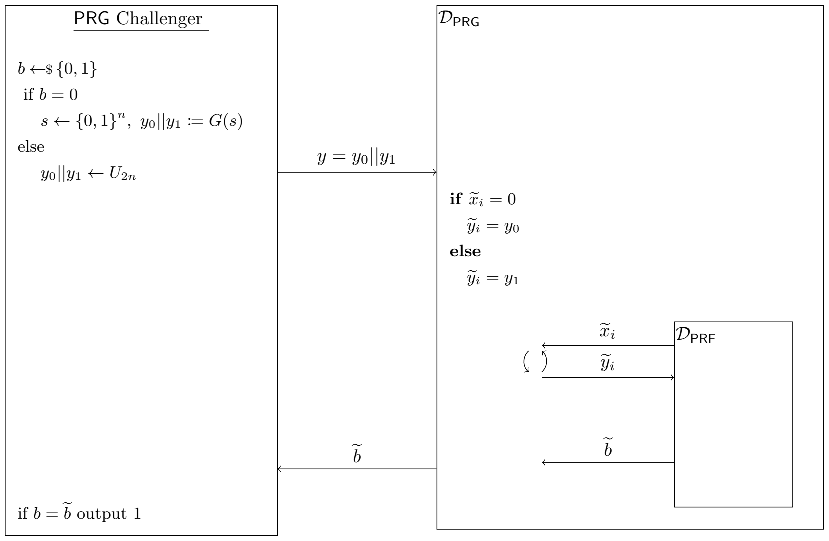

PRG Challenger
$b \leftarrow\$ \{0,1\}$
if $b = 0$
$s \leftarrow \{0,1\}^n,\ y_0||y_1 := G(s)$
else
$y_0||y_1 \leftarrow U_{2n}$
if $b = \widetilde{b}$ output 1
$\mathcal{D}_{\mathsf{PRG}}$
$y = y_0||y_1$
if $\widetilde{x}_i = 0$
$\widetilde{y}_i = y_0$
else
$\widetilde{y}_i = y_1$
$\widetilde{x}_i$
$\widetilde{y}_i$
$\mathcal{D}_{\mathsf{PRF}}$
$\widetilde{b}$
$\widetilde{b}$

Figure 5.12: Security of 1-bit PRF

Since the distinguisher $\mathcal{D}_{\mathsf{PRF}}$ can distinguish between a random function F and the constructed PRF with noticeable probability. From the described reduction (Figure 5.12), it follows that $\mathcal{D}_{\mathsf{PRG}}$ can also distinguish between a random string and the output of the PRG with the same noticeable probability. But from the security of the PRG G, we know that no such distinguisher exists and therefore no such distinguisher for the PRF can exist. □

5.7.3 PRF with n-bit input

Now that we have a PRF construction with a 1-bit input, can we extend the same idea to n-bit outputs? Let's think of the naive extension where instead of partitioning the output of the PRG G into two parts, we start with a PRG G with polynomial size output and partition the output into 2^ℓ parts where ℓ is the length of the PRF input. Since G can only output $\mathsf{poly}(n)$ bits, ℓ is limited to being $\log(n)$. This yields a PRF $F_k : \{0,1\}^{\log(n)} \mapsto \{0,1\}^n$ from a PRG $G(s) = y_1||y_2||\cdots||y_{\mathsf{poly}(n)}$, where each y_i is n-bits.

This is a useful start, but doesn't give us what we want: going beyond polynomial stretch. Instead, for n-bit inputs, we will use the "double and choose" policy used in out 1-bit PRF construction repeatedly. This wonderful construction was first described by Golreich, Goldwasser and Micali; and is referred to as the *GGM* construction.

Theorem 13 (Goldreich-Goldwasser-Micali (GGM) [GGM84]). *If pseudorandom generators exist, then pseudorandom functions exist.*

Proof Let us fix some terminology, we start with a length-doubling PRG $G : \{0,1\}^n \mapsto \{0,1\}^{2n}$. For convenience, we define two functions G_0 and G_1 that refer to either the left or right half of G's output, i.e. $G_0(s) \coloneqq G(s)[1:n]$, and $G_1(s) \coloneqq G(s)[n+1:2n]$. The construction, in a concise manner, is given below.

```
F_k(x)
1 :  $s \coloneqq k$
2 :  output $G_{x_n}(G_{x_{n-1}}(.....(G_{x_1}(k))..)$
```

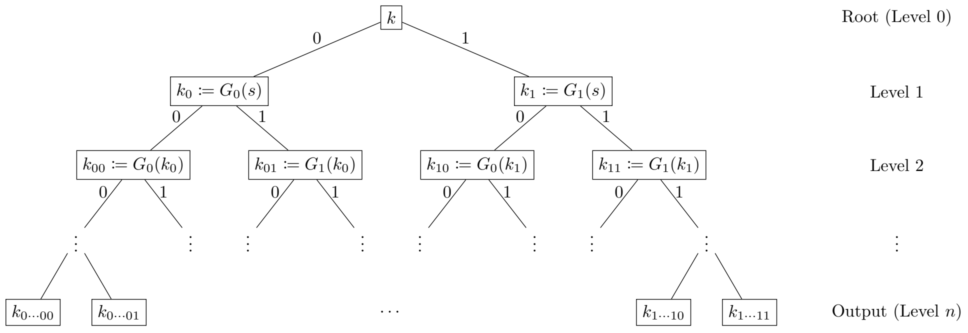


Figure 5.13: PRF Construction

Let us look at the construction in more detail. It is convenient to think of the construction of F_k as a binary tree of size 2^n, see Figure 5.13. Here k denotes the root (chosen randomly); the left and right child at level 1 of the tree are denoted as $k_0 := G_0(k)$ and $k_1 := G_1(k)$. The output of the PRG, k_0 and k_1, now being fed as the seed to the PRG to generate the next level of the tree. It is easy to see from Figure 5.13 how this extends to higher levels. Specifically, at level ℓ, there are 2^ℓ nodes; and each of these nodes are labeled $k_{x_1,...,x_\ell}$ for $x_1, \cdots, x_\ell \in \{0, 1\}$.

The evaluation of function F_k on an input string $x_1x_2....x_n$ can be thought of as a walk down the branches of the tree up till the leaf nodes. Starting from the root, based on the value of x_1 (0 or 1), either the path traverses through the left branch ($x_1 = 0$) or the right branch ($x_1 = 1$). Subsequently, the choice of branches is based on individual bits of the input string. Every input would correspond to a unique path and thus a unique leaf node, since at least one bit (and correspondingly, one edge in the tree) would be different.

Efficiency. One of the requirements for a PRF (Definition 24) is that it must be efficient. While the tree itself is exponentially sized, there is no need to store the entire tree and the output value can be computed on the fly. From our description of F_k above, we note that we execute G n-times, one for each bit of the input. Therefore,

$$\text{RunningTime}(F_k(x)) = n \cdot \text{ RunningTime}(G(\cdot)) = n \cdot \mathsf{poly}(n) = \mathsf{poly}'(n),$$

satisfying the efficiency requirement.

Security. The natural intuition is to prove using Hybrid arguments. The first idea to construct hybrids, is to replace each node in the tree, from the output of a PRG to a random string one by one as was done in the proof of Lemma 12. But since there are exponential number of nodes in the tree, this would result in exponential number of hybrids. Unfortunately, the Hybrid Lemma (Lemma 9) only works for a polynomial number of hybrids. To overcome this barrier, we make use of the following interesting observation: any PPT adversary can only make polynomial queries to F_k. By construction, each query corresponds to a unique path (of n nodes), therefore with a single query the adversary learns some information about these n nodes. Does it learn information about any of the other nodes? Since any node in the tree is half the output of a PRG, the query could conceivably leak information about the siblings of each of these n nodes - corresponding to the *other half* of the PRG output. Therefore, with a single query, the adversary learns some information about $2n$ nodes. By making $\mathsf{poly}(n)$ many queries, it therefore learns information about at most $2n \cdot \mathsf{poly}(n) =$

$\mathsf{poly}(n)$ nodes. Therefore, to argue security we only need to change polynomially many nodes in the tree, since the adversary is oblivious to all other nodes that are *untouched* by its queries to F_k. We achieve this by using 2 layers of nested hybrids.

Make sure you understand this observation; and why the siblings are inlcuded.

Let's look at the **first level of hybrids**.

H_0 :Level 0 nodes are random strings
Level $i > 0$ nodes are pseudorandom strings

H_1 :Level 0,1 nodes are random strings
Level $i > 1$ nodes are pseudorandom strings

$\vdots$

H_i :Level $\leq i$ nodes are random strings
Level $> i$ nodes are pseudorandom strings

$\vdots$

H_n :Nodes at all levels are random strings

First, note that when we say that the nodes of level i are random or pseudorandom,we mean all nodes of level i that are affected by the adversaries queries that are random or pseudorandom respectively. Next, note the hybrid H_0 corresponds to the adversary's view of the actual PRF construction, while H_n corresponds to the view of a random function (since the outputs are simply random strings).

Let us assume that the constructed function F_k is distinguishable from a truly random function F, then by Definition 24 there exists an adversary $\mathcal{D}_{\mathsf{PRF}}$ that can distinguish the two cases with noticeable probability $\varepsilon(n)$. From the construction of our hybrids, this corresponds to $\mathcal{D}_{\mathsf{PRF}}$ distinguishing between hybrids H_0 and H_n with the same probability. Then by the Hybrid Lemma (Lemma 9), there must exist $i \in [n]$ such that $\mathcal{D}_{\mathsf{PRF}}$ distinguishes between hybrids H_i and H_{i+1} with probability $\geq \frac{\varepsilon(n)}{n}$. Note, that from our description of the hybrids, the only difference between H_i and H_{i+1} is that:

– in H_i, nodes at level $i+1$ are pseudorandom strings.
– while in H_{i+1}, nodes at level $i+1$ are random strings.

Since there are multiple nodes at level $i+1$ that are affected by the adversary's queries, we need to create another set of hybrids between H_i and H_{i+1}. To create these hybrids, we only need to replace the nodes that are affected by the queries of the adversary. Since the number of queries are polynomial, changing polynomial number of nodes is sufficient from adversary's point of view.

Let S be the set of nodes at level i that are affected by the adversary/distinguisher's input queries. We know that $|S| = \mathsf{poly}(n)$. We now define the **second level of hybrids below**, where we assume that all node in S are in lexicographic order: $\forall j \in [|S|]$

$$\begin{aligned}\mathsf{H}_{i,j} \text{ :} & \text{same as } \mathsf{H}_i\text{, except that all nodes at level } i+1 \\ & \text{that are children of nodes} \leq j\text{, are random strings.}\end{aligned}$$

From the description above, we see that $\mathsf{H}_{i,0}$ corresponds to the hybrid H_i while $\mathsf{H}_{i,|S|}$ corresponds to the string H_{i+1}.

Again, given $\mathcal{D}_{\mathsf{PRF}}$ distinguishing between hybrids $\mathsf{H}_{i,0}$ and $\mathsf{H}_{i,|S|}$ (from above), by Hybrid Lemma there must exist $j \in [|S|]$ such that $\mathcal{D}_{\mathsf{PRF}}$ distinguishes between hybrids H_i and H_{i+1} with probability $\geq \frac{\varepsilon(n)}{n \cdot |S|} \geq \frac{1}{\mathsf{poly}(n)}$ (recall that $\varepsilon(n)$ is noticeable (i.e., $\geq 1/\mathsf{poly}(n)$)).

- in $\mathsf{H}_{i,j}$, node j in level $i+1$ is a pseudorandom string.
- while $\mathsf{H}_{i,j+1}$, node j in level $i+1$ is a random string.

Given $\mathcal{D}_{\mathsf{PRF}}$ that distinguishes between hybrids $\mathsf{H}_{i,j}$ and $\mathsf{H}_{i,j+1}$, we will construct an adversary $\mathcal{D}_{\mathsf{PRG}}$ that can efficiently distinguish between the output of a PRG and a random string. See Figure 5.14 for the reduction. This is slightly more complex than the reductions you've seen so far. Primarily because $\mathcal{D}_{\mathsf{PRG}}$ has to simulate $\mathsf{H}_{i,j}$ and $\mathsf{H}_{i,j+1}$ relying only on the string $y_0||y_1$ to introduce the difference between the two hybrids. In order to do so, $\mathcal{D}_{\mathsf{PRG}}$ chooses random n-bit strings for nodes that are affected by the adversary's queries, up till level i and for nodes at level $i+1$ that are children of nodes $< j$. The children of node j are substituted by y_0 and y_1 respectively. The remaining nodes in the tree that are affected by the adversary's queries as computed using the chosen random values and y_0 and y_1. $\mathcal{D}_{\mathsf{PRG}}$ replies to the queries of $\mathcal{D}_{\mathsf{PRF}}$ using this tree.

Since the distinguisher $\mathcal{D}_{\mathsf{PRF}}$ can distinguish between hybrids $\mathsf{H}_{i,j}$ and $\mathsf{H}_{i,j+1}$ with noticeable probability. From the described reduction (Figure 5.12), it follows that $\mathcal{D}_{\mathsf{PRG}}$ can also distinguish between a random string and the output of the PRG with the same noticeable probability. But from the security of the PRG G, we know that no such distinguisher exists and therefore no such distinguisher for the $\mathcal{D}_{\mathsf{PRF}}$ can exist. Therefore, following through all our assumptions we conclude that the constructed PRF F_k must be secure. □

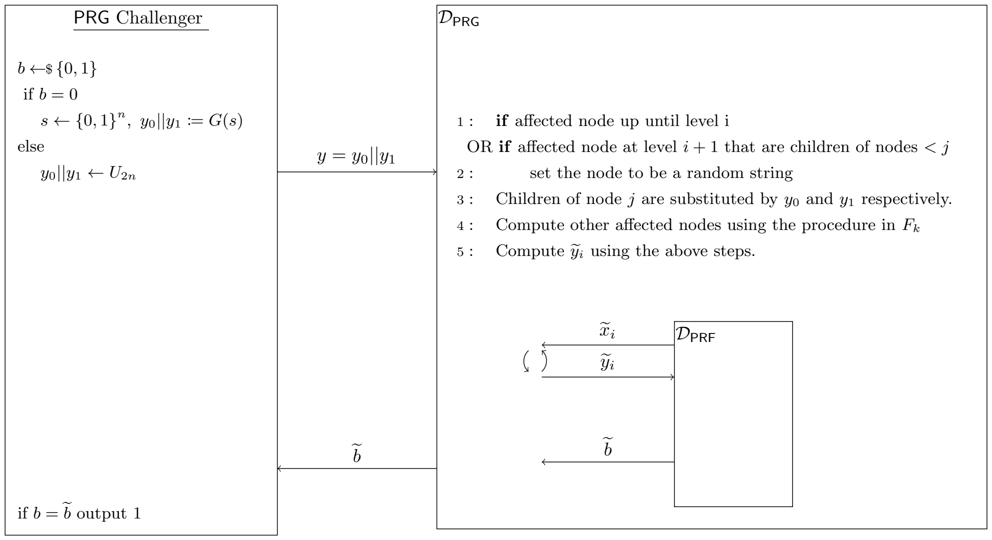

PRG Challenger
$b \leftarrow\$ \{0,1\}$
if $b = 0$
$s \leftarrow \{0,1\}^n,\ y_0||y_1 := G(s)$
else
$y_0||y_1 \leftarrow U_{2n}$
if $b = \widetilde{b}$ output 1
$y = y_0||y_1$
$\widetilde{b}$
$\mathcal{D}_{\mathsf{PRG}}$
1 : if affected node up until level i
OR if affected node at level $i+1$ that are children of nodes $< j$
2 : set the node to be a random string
3 : Children of node j are substituted by y_0 and y_1 respectively.
4 : Compute other affected nodes using the procedure in F_k
5 : Compute $\widetilde{y}_i$ using the above steps.
$\widetilde{x}_i$
$\widetilde{y}_i$
$\widetilde{b}$
$\mathcal{D}_{\mathsf{PRF}}$

Figure 5.14: Security of 1-bit PRF

Chapter Notes

We would like to note some other interesting results pertaining to PRFs and extensions to the definition

Efficient PRFs from concrete assumptions: [NR97, BPR12].

Constrained PRFs: PRFs with “punctured” keys that are disabled on certain inputs [BW13, KPTZ13, BGI14, SW14].

Related-key Security: Evaluation of $F_k(x)$ does not help in predicting $F_{k'}(x)$ [BC10].

Key-Homomorphic PRF: Given $F_k(x)$ and $F_{k'}(x)$ compute $F_{g(k,k')}(x)$ [BLMR13].

Chapter 6

Key Exchange

Going forward, we shall see that one of the fundamental requirements will be for Alice and Bob to be in possession of a shared secret. In this chapter we shall see how Alice and Bob can do so even if there is an eavesdropper Eve listening in on their conversation.

We shall start with some preliminary definitions and tools before we formally define key agreement, and finally provide a construction building on the tools.

6.1 Groups

We start by defining the notion of a group and its associated properties. A group $\mathbb{G}$ is a non-empty set with an associated binary operation $\bullet$ that takes in two elements, and outputs a third. The binary operator needs to satisfy additional properties, that are listed below.

Definition 25. $(\mathbb{G}, \bullet)$ *is a group if it satisfies the following conditions:*

Closure: *For all* $a, b \in \mathbb{G}$*, we have* $a \bullet b \in \mathbb{G}$.
Associativity: *For all* $a, b, c \in \mathbb{G}$*, we have* $(a \bullet b) \bullet c = a \bullet (b \bullet c)$.
Identity *There exists an identity element* e *such that for all* $a \in \mathbb{G}$*, we have* $e \bullet a = a$.
Inverse *For every* $a \in \mathbb{G}$*, there exists* $b \in \mathbb{G}$ *such that* $a \bullet b = e$.

Note that the above definition doesn't require the group operator to be commutative, i.e. does not require $a \bullet b = b \bullet a$. Try to think of an example of a group for which this is not true. Groups that additionally satisfy the commutative property are called *Abelian Groups*. In this course, we will restrict our attention to abelian groups.

While the above generic definition of a group is useful, we shall find it convenient to consider concrete examples of groups.

Example 15. *A simple example is the group $(\mathbb{Z}, +)$ of integers with the addition operator. It is easy to that it satisfies the properties listed in the definition. One should consider whether the same is true if we replace addition with multiplication, i.e. is $(\mathbb{Z}, \times)$ a group?*

6.1.1 Cyclic Groups

For our course we shall primarily consider groups that are additionally structured, and are referred to as cyclic groups.

Definition 26. *A group $(\mathbb{G}, \cdot)$ is a cyclic group if is generated by a single element, i.e. there exists a $g \in \mathbb{G}$ such that $\mathbb{G} = \{g^0, g^1, \cdots, g^{n-1}\}$ where $g^i = \underbrace{g \cdot g \cdots g}_{i \text{ times}}$. Such a cyclic group $\mathbb{G}$ is denoted as $\mathbb{G} = \langle g \rangle$ and g is called the* generator *of the group.*

From the above definition, it is clear that the order (size) of $\mathbb{G}$ is $|\mathbb{G}| = n$. When clear from context, we shall often drop the group operator to avoid cluttered notation.

6.2 Hard Problems in Cyclic Groups

In this section, we will look at some problems that are believed to be hard for certain cyclic groups. We will then leverage the hardness of one of the problems to construct a key agreement scheme. For each of the problems below we consider a cyclic group $\mathbb{G}$ of order p, where p is an n-bit *safe prime* number (i.e. $p = 2q + 1$ for some large prime q).

6.2.1 Discrete Log (DL)

Since $\mathbb{G}$ is cyclic, every element can be represented uniquely as g^a for some $a \mod p$. It turns out that given an x, it is hard to find the corresponding a such that $g^{a \mod p} = x$. Of course this can't hold true for all x, and we formalize this below requiring the computation of a *discrete log* for a random x. Unless explicitly required, we drop the $\mod p$ from the exponent and assume values in the exponent are always from $\{0, \cdots, p-1\}$.

We say $a \equiv b \mod p$ if there exists an integer k such that $a = kp + b$, i.e. b is the remained when a is divided by p and takes values only in $\{0, \cdots, p-1\}$.

Definition 27 (Discrete Log (DL) Assumption). *Let $\mathbb{G}$ be a cyclic group of order p (where p is a safe prime) with generator g, then for every non-uniform* PPT *adversary $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(n)$ *such that*

$$\Pr[a \leftarrow\$ \{0, \cdots, p-1\}, \widetilde{a} \leftarrow \mathcal{A}(\mathbb{G}, p, g, g^a) : \widetilde{a} = a] \leq \mathsf{negl}(n)$$

The DL *problem* is to find a, while the *assumption* is that the DL problem is hard.

As we've seen throughout, we shall find it convenient to describe the above definition as a game the DL problem in Figure 6.1 between the DL Challenger and adversary $\mathcal{A}_{\mathsf{DL}}$, where the adversary wins if the challenger outputs 1. We require that the probability $\mathcal{A}_{\mathsf{DL}}$ wins is bounded by $\mathsf{negl}(n)$.

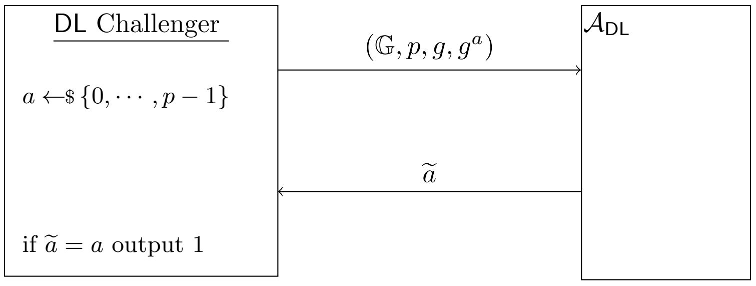

DL Challenger
$a \leftarrow\$ \{0, \cdots, p-1\}$
if $\widetilde{a} = a$ output 1
$(\mathbb{G}, p, g, g^a)$
$\widetilde{a}$
$\mathcal{A}_{\mathsf{DL}}$

Figure 6.1: DL Challenge Game

6.2.2 Computational Diffie-Hellman (CDH)

It turns out that while DL is a natural problem, we do not know how to leverage it get primitives we want. Instead, we define two other related problem. We start with the first, the *computational Diffie-Hellman* assumption (CDH). Here, an adversary is given g^a and g^b and asked to compute g^{ab}. Intuition would suggest that this is a more difficult problem than DL, and we will indeed formalize this intuition in Section 6.2.4. We formally define the problem below.

Definition 28 (Computational Diffie-Hellman Problem (CDH) Assumption). *Let $\mathbb{G}$ be a cyclic group of order p (where p is a safe prime)* *with generator g, then for every non-uniform* PPT *adversary $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(n)$ *such that*

$$\Pr\Big[a, b \leftarrow\$ \{0, \cdots, p-1\}, y \leftarrow \mathcal{A}(\mathbb{G}, p, g, g^a, g^b) : y = g^{ab}\Big] \leq \mathsf{negl}(n)$$

The corresponding game the CDH problem between the CDH Challenger and adversary $\mathcal{A}_{\mathsf{CDH}}$ is given in Figure 6.2.

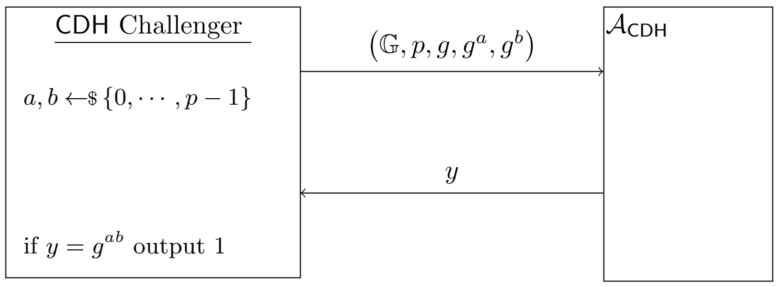

CDH Challenger
$a, b \leftarrow\$ \{0, \cdots, p-1\}$
if $y = g^{ab}$ output 1
$(\mathbb{G}, p, g, g^a, g^b)$
y
$\mathcal{A}_{\mathsf{CDH}}$

Figure 6.2: CDH Challenge Game

6.2.3 Decisional Diffie-Hellman (DDH)

The above definitions have a flavor similar to one-way functions, where we ask the adversary to compute some value given some information. The next definition is of the indistinguishability flavor, where we simply ask the adversary to distinguish between two distributions. Specifically, we give the adversary either (g^a, g^b, g^{ab}) or (g^a, g^b, g^r), and ask that it distinguish the two for a random r. This is formalized below.

Definition 29 (Decisional Diffie-Hellman (DDH) Assumption). *Let $\mathbb{G}$ be a cyclic group of order p (where p is a safe prime) with generator g, then the following two distributions are computationally indistinguishable:*

- $\Big\{a, b \leftarrow\$ \{0, \cdots, p-1\} : \Big(\mathbb{G}, p, g, g^a, g^b, g^{ab}\Big)\Big\}$
- $\Big\{a, b, r \leftarrow\$ \{0, \cdots, p-1\} : \Big(\mathbb{G}, p, g, g^a, g^b, g^r\Big)\Big\}$

The corresponding game for the DDH problem between the DDH Challenger and adversary $\mathcal{A}_{\mathsf{DDH}}$ is given in Figure 6.3.

Remark 7. *It should be noted that the* DDH *assumption as stated above is not*

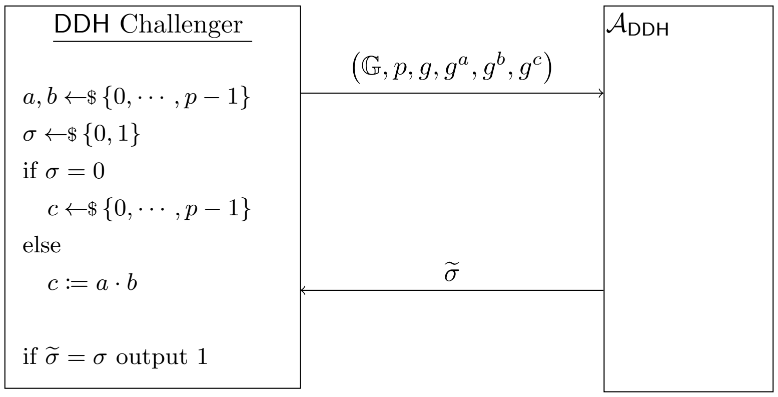

DDH Challenger
$a, b \leftarrow\$ \{0, \cdots, p-1\}$
$\sigma \leftarrow\$ \{0,1\}$
if $\sigma = 0$
$c \leftarrow\$ \{0, \cdots, p-1\}$
else
$c := a \cdot b$
if $\widetilde{\sigma} = \sigma$ output 1
$\mathcal{A}_{\mathsf{DDH}}$
$(\mathbb{G}, p, g, g^a, g^b, g^c)$
$\widetilde{\sigma}$

Figure 6.3: DDH Challenge Game

necessarily true in all cyclic groups and care must be taken when we pick the group $\mathbb{G}$ to work with.

6.2.4 Relation Between the Hard Problems

From the ordering of the problems above, intuitively it seems like we are requiring stronger assumptions to hold at each step. We formalize this below by showing,

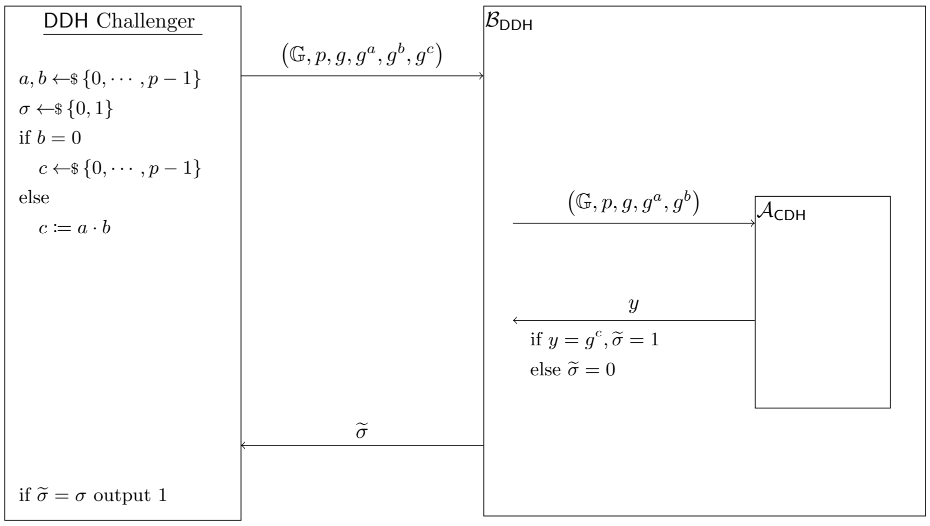
$$\mathsf{DDH} \implies \mathsf{CDH} \implies \mathsf{DL}$$

$\mathsf{DDH} \implies \mathsf{CDH}$. We show that DDH is a stronger requirement than CDH. Specifically, we show that any adversary $\mathcal{A}_{\mathsf{CDH}}$ breaking CDH can be used to construct an adversary $\mathcal{B}_{\mathsf{DDH}}$ that breaks DDH. The sketch of the reduction is provide in Figure 6.4.

$\mathsf{CDH} \implies \mathsf{DL}$. We show that CDH is a stronger requirement than DL. Specifically, we show that any adversary $\mathcal{A}_{\mathsf{DL}}$ breaking DL can be used to construct an adversary $\mathcal{B}_{\mathsf{CDH}}$ that breaks DL. The sketch of the reduction is provide in Figure 6.5.

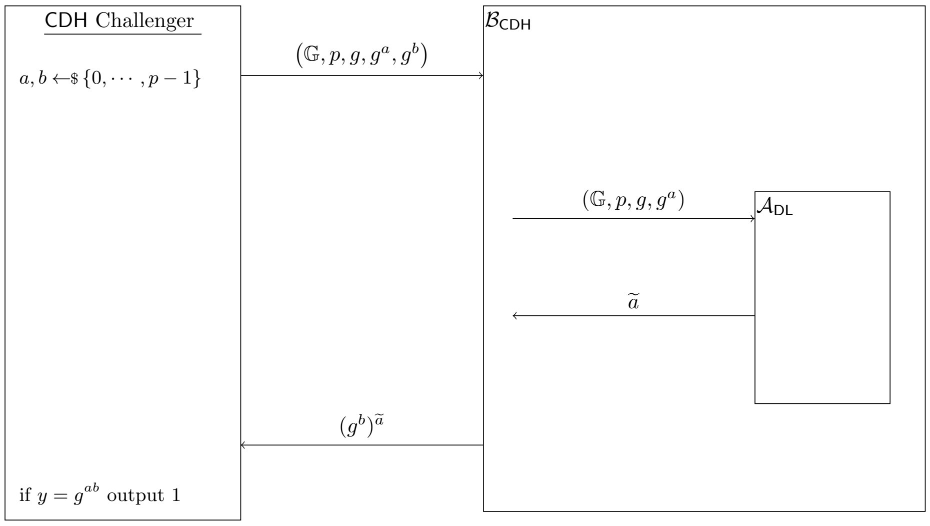
6.3 Key Agreement

Now that we have the tools, let us formally define a key agreement (or exchange) protocol.



DDH Challenger
$a, b \leftarrow\$ \{0, \cdots, p-1\}$
$\sigma \leftarrow\$ \{0,1\}$
if $b = 0$
$c \leftarrow\$ \{0, \cdots, p-1\}$
else
$c := a \cdot b$
if $\widetilde{\sigma} = \sigma$ output 1
$\mathcal{B}_{\mathsf{DDH}}$
$(\mathbb{G}, p, g, g^a, g^b, g^c)$
$(\mathbb{G}, p, g, g^a, g^b)$
$\mathcal{A}_{\mathsf{CDH}}$
y
if $y = g^c, \widetilde{\sigma} = 1$
else $\widetilde{\sigma} = 0$
$\widetilde{\sigma}$

Figure 6.4: DDH $\implies$ CDH



CDH Challenger
$a, b \leftarrow\$ \{0, \cdots, p-1\}$
if $y = g^{ab}$ output 1
$\mathcal{B}_{\mathsf{CDH}}$
$(\mathbb{G}, p, g, g^a, g^b)$
$(\mathbb{G}, p, g, g^a)$
$\mathcal{A}_{\mathsf{DL}}$
$\widetilde{a}$
$(g^b)^{\widetilde{a}}$

Figure 6.5: CDH $\implies$ DL

Definition 30 (Key Agreement Protocol). *A key agreement protocol is an efficient protocol executed by two parties Alice and Bob. During protocol execution, Alice uses randomness r_A while Bob uses randomness r_b.*

The protocol transcript of the execution is $\tau := \tau(r_A, r_B)$; Alice's output is $k_A := k_A(\tau, r_A)$ and Bob's output is $k_B := k_B(\tau, r_B)$. The protocol satisfies the following properties:

Correctness

$$\Pr_{r_A, r_B}[k_A = k_B] = 1 - \mathsf{negl}(n)$$

Security

$$(k_A, \tau) \equiv (k_B, \tau) \approx_c (r, \tau)$$

where the eavesdropper Eve's view is simply the transcript τ, and r is a uniformly sampled string independent of τ.

The definition states that even given the transcript, an adversarial Eve who simply eavesdrops cannot tell the difference between the established key k_A (or k_B) from a truly random string independent of τ.

6.4 Construction of Key Agreement Protocol based on DDH

In this section, we finally construct a key agreement protocol based on the hardness of the Decisional Diffie-Hellman problem (Section 6.2.3)[DH76].

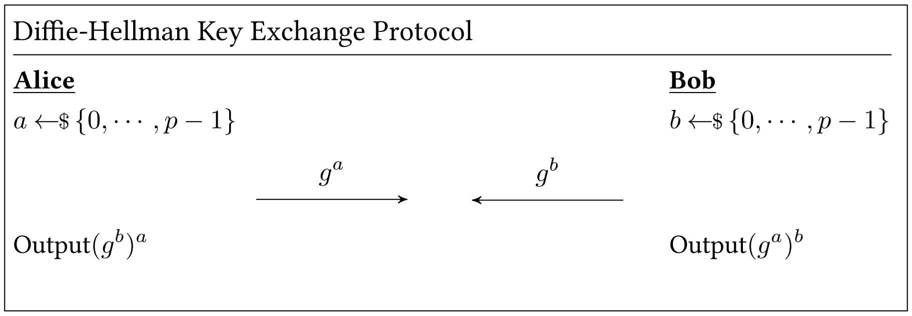

Diffie-Hellman Key Exchange Protocol
Alice
Bob
$a \leftarrow\$ \{0, \cdots, p-1\}$
$b \leftarrow\$ \{0, \cdots, p-1\}$
g^a
g^b
Output$(g^b)^a$
Output$(g^a)^b$

Figure 6.6: Diffie-Hellman Key Exchange Protocol

Make sure to map the protocol to the algorithms from Defintion 30 and convince yourself why the security holds. Note that g^r for a random r can be considered to be uniformly random.

The security follows directly from the definition of the key agreement protocol and the DDH assumption.

Remark 8. *A few remarks regarding the above protocol are in order.*

1. *Note that we only prove the above protocol secure against eavesdropping adversaries. These are the mildest form of adversaries, since one might expect an*

adversary to modify the messages sent between Alice and Bob. We call such adversaries active, and one consider whether the above protocol remains secure against active adversaries.

2. *We needed the stronger assumption of* DDH *since a weaker assumption, say, like* CDH *doesn't prevent half the bits of the shared key from being leaked.*

Chapter 7

Secret-Key Encryption

In this chapter we shall talk about encryption, the most commonly associated term when one thinks of cryptography. Historically, cryptography was used almost exclusively for encryption. Over this and subsequent chapters we shall see many wonderful applications of cryptography using the tools we've discussed so far, but also developing further tools along the way..

7.1 Setting

We assume that Alice and Bob share a secret $s \in \{0,1\}^n$. In Chapter 6 we have seen how Alice and Bob can establish such a shared secret. Alice wants to send a private message to Bob, and we assume the "worst case" that the communication channel is public such that a third party Eve can read all messages sent on the channel. Alice will use an encryption scheme to communicate over the public channel with Bob.

7.2 Secret-key Encryption

We establish first the syntax, then talk about correctness and security of the encryption scheme.

Syntax A secret-key encryption consists of three algorithms described below:

Key Generation. $\mathsf{KGen}(1^n)$ is a randomized algorithm that takes no input, and outputs randomly sampled key s.

Encryption. $\mathsf{Enc}(s, m)$ may either be a deterministic or randomized algorithm that takes as input a key s, message m and

outputs a ciphertext c.
When the encryption is randomized, we will sometimes explicitly add an additional input to include randomness as $\mathsf{Enc}(s, m; r)$ where r is the randomness. Note that when the randomness is explicit, Enc is a deterministic algorithm, i.e. randomness has already been accounted for in the input.

Decryption. $\mathsf{Dec}(s, c)$ is a deterministic algorithm that takes as input a key s, ciphertext c and outputs a message m.

Each of these algorithms must run in polynomial time. In the above scenario, to send a message m to Bob, Alice uses the encryption algorithm to compute $c \leftarrow \mathsf{Enc}(s, m)$ and sends c on the public channel to Bob. Bob then uses the decryption algorithm $m := \mathsf{Dec}(s, c)$ to recover the message.
A secret-key encryption scheme must satisfy the two properties described below:

Correctness We require that the scheme satisfies correctness, where decrypting an encrypted message should yield the message.

$$\Pr[s \leftarrow \mathsf{KGen}(1^n), c \leftarrow \mathsf{Enc}(s, m)\ :\ \mathsf{Dec}(s, c) = m] = 1$$

where the probability is over the randomness used in the key generation and encryption algorithms.
For simplicity, since the probability is 1, this can be restated as

$$\forall m,\ \mathsf{Dec}(s, \mathsf{Enc}(s, m)) = m,$$

where $s \leftarrow \mathsf{KGen}(1^n)$.

Security Intuitively an eavesdropper Eve must not be able to decipher the message m from the ciphertext c. More specifically, we require that she cannot distinguish between ciphertexts of two different messages m and m'. To formalize this we introduce the notion of $\mathsf{IND\text{-}CPA}$ Security (Indistinguishability under Chosen Plaintext Attack). Note here that n is known as the "security parameter" which expresses the degree of security of the scheme.

Definition 31 ($\mathsf{IND\text{-}CPA}$)**.** *A secret-key encryption scheme* $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *is* $\mathsf{IND\text{-}CPA}$ *secure if for all non uniform* PPT *ad-*

versaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ *such that:*

$$\Pr\left[\begin{array}{r} s \leftarrow \mathsf{KGen}(1^n), \\ (m_0, m_1) \leftarrow \mathcal{A}(1^n), \\ b \leftarrow\!\!\$ \{0,1\} \end{array} : \mathcal{A}(\mathsf{Enc}(s, m_b)) = b \right] \leq \frac{1}{2} + \mathsf{negl}(n)$$

This is a game based definition as you've seen before, but importantly the adversary is allowed to pick the messages m_0 and m_1 of its choice. The game is described in Figure 7.1.

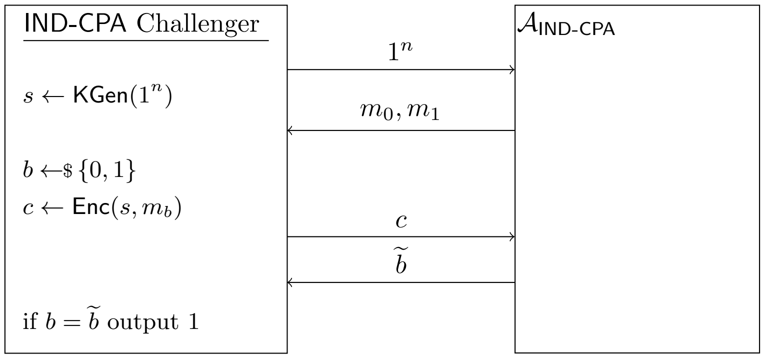

IND-CPA Challenger
$\mathcal{A}_{\text{IND-CPA}}$
1^n
$s \leftarrow \mathsf{KGen}(1^n)$
m_0, m_1
$b \leftarrow\!\!\$ \{0,1\}$
$c \leftarrow \mathsf{Enc}(s, m_b)$
c
$\widetilde{b}$
if $b = \widetilde{b}$ output 1

Figure 7.1: IND-CPA security of secret key encryption

Remark 9. *Note the lengths of the messages must be the same to prevent a simple attack that inspects the length of the ciphertext in order to differentiate.*

Often it is easier to think of this definition as requiring computationally indistinguishability of the ciphertexts:

$$\mathsf{Enc}(m_0) \approx_c \mathsf{Enc}(m_1)$$

7.2.1 One-Time Pads

Let's start off constructing a simple scheme satisfying the above definition. Consider the following scheme

```
KGen(1^n)
1 :  $s \leftarrow\!\!\$ \{0,1\}^n$
2 :  Return $s$

Enc(s, m)
1 :  $c := s \oplus m$
2 :  Return $c$

Dec(s, c)
1 :  $m := s \oplus c$
2 :  Return $m$
```

Correctness follows trivially from the properties of XOR ($\oplus$). For security since s is random, $s \oplus m$ is also random so m is hidden from an informational theoretic perspective. That is to say $\mathsf{Enc}(s, m) \equiv U_n$, which is to say that they are identically distributed. Thus two ciphertexts are more than just indistinguishable, they are in fact identically distributed.

$$\{s \leftarrow \mathsf{KGen}(1^n) : \mathsf{Enc}(s, m_0)\} \equiv \{s \leftarrow \mathsf{KGen}(1^n) : \mathsf{Enc}(s, m_1)\}$$

Think of why even leaking the XOR of two messages is a bad idea.

Unfortunately, if we try to use the same key to encrypt multiple messages, the security no longer holds. This too follows from the properties of XOR, consider two ciphertexts c_1 and c_2 encrypted under the same key s, $c_1 \oplus c_2 = (s \oplus m_1) \oplus (s \oplus m_1) = m_1 \oplus m_2$ which can break the security.

7.2.2 Encryption using PRG

In the simple scheme that we saw above, we noted that a key can never be reused, meaning that the length of the secret-key grows with the length of the message being encrypted. We now discuss an encryption scheme where a secret-key of a fixed length can be used to encrypt a polynomially long message.

We will construct such an encryption using pseudorandom generators (PRG) (Definition 22) by relying on the fact that the output of a PRG is computationally indistinguishable from a uniformly random string. The intuition is that we can use a PRG to convert a random key of a fixed length into a pseudorandom key of the necessary length to encrypt messages that are of arbitrary polynomial length.

Consider the following encryption scheme where the length of the message is $\ell(n)$ and $G : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ is the PRG:

```
KGen(1^n)
1 :  $s \leftarrow\$ \{0,1\}^n$
2 :  Return $s$
```

```
Enc(s, m)
1 :  $c := G(s) \oplus m$
2 :  Return $c$
```

```
Dec(s, c)
1 :  $m := G(s) \oplus c$
2 :  Return $m$
```

For security we do not have the identical distribution to uniform random as we did with one-time pads, instead we show security via indistinguishability.

Proposition 2 (Security of Encryption using PRG).

$$\{s \leftarrow \mathsf{KGen}(1^n) : \mathsf{Enc}(s, m_0)\} \approx_c \{s \leftarrow \mathsf{KGen}(1^n) : \mathsf{Enc}(s, m_1)\}$$

Proof Security is proven via a hybrid argument. Rather than using the hybrid lemma though we prove in the forward direction by using the fact that indistinguishability is transitive over polynomial number of hybrids. Consider then the following list of hybrids, where the change between hybrids have been highlighted:

```
$\mathsf{H}_0$
1 : $s \leftarrow \mathsf{KGen}(1^n)$
2 : $r := G(s)$
3 : $c := m_0 \oplus r$
```

```
$\mathsf{H}_1$
1 : $s \leftarrow \mathsf{KGen}(1^n)$
2 : $r \leftarrow\$ \{0,1\}^{\ell(n)}$
3 : $c := m_0 \oplus r$
```

```
$\mathsf{H}_2$
1 : $s \leftarrow \mathsf{KGen}(1^n)$
2 : $r \leftarrow\$ \{0,1\}^{\ell(n)}$
3 : $c := m_1 \oplus r$
```

```
$\mathsf{H}_3$
1 : $s \leftarrow \mathsf{KGen}(1^n)$
2 : $r := G(s)$
3 : $c := m_1 \oplus r$
```

Note that the output of each hybrid is simply the ciphertext c. From the description of the hybrids, it is clear that $\mathsf{H}_0 = \{s \leftarrow \mathsf{KGen}(1^n) : \mathsf{Enc}(s, m_0)\}$ and $\mathsf{H}_3 = \{s \leftarrow \mathsf{KGen}(1^n) : \mathsf{Enc}(s, m_1)\}$.

First, we show that H_0 and H_1 are computationally indistinguishable by relying on the security of the PRG. For contradiction, let $\mathcal{A}_{\mathsf{H}_0,\mathsf{H}_1}$ be the adversary that distinguishes between H_0 and H_1, using $\mathcal{A}_{\mathsf{H}_0,\mathsf{H}_1}$ we will construct an adversary $\mathcal{A}_{\mathsf{PRG}}$ that we will use to win the PRG game. The reduction is presented in Figure 7.2.

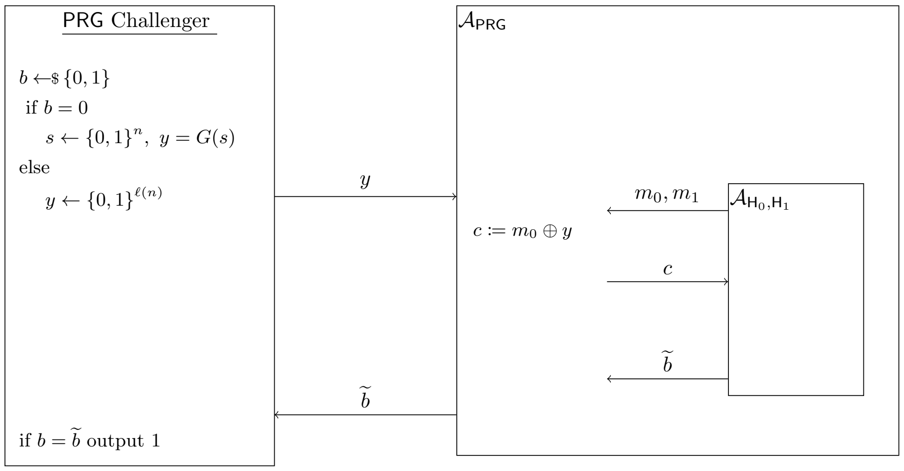

PRG Challenger
$b \leftarrow\$ \{0,1\}$
if $b = 0$
$s \leftarrow \{0,1\}^n,\ y = G(s)$
else
$y \leftarrow \{0,1\}^{\ell(n)}$
if $b = \widetilde{b}$ output 1
y
$\widetilde{b}$
$\mathcal{A}_{\mathsf{PRG}}$
$c := m_0 \oplus y$
m_0, m_1
c
$\widetilde{b}$
$\mathcal{A}_{\mathsf{H}_0,\mathsf{H}_1}$

Figure 7.2: Indistinguishability of H_0 and H_1

The next pair of hybrids H_1 and H_2 are (information theoretically) indistinguishable by the indistinguishability of one-time pads as we've seen before. Finally H_2 and H_3 are indistinguishable by a symmetric argument to H_0 and H_1. Thus by transitivity of the Hybrid Lemma (Lemma 9), H_0 and H_3 are indistinguishable, which establishes that the scheme is IND-CPA secure. □

Remark 10. *Note that in Hybrid* H_1 *we are creating a ciphertext that cannot be decrypted correctly since it doesn't follow the encryption procedure. Here correctness does not matter since the adversary does not have access to the decryption procedure to check if the ciphertext is indeed computed correctly. It suffices that the first and last hybrids compute ciphertexts that are* correctly formed.

7.3 Multi-message Secure Encryption

So far, we have only discussed encryption schemes where a key can be used to encrypt a *single* message. In practice we want to be able to encrypt multiple messages. We extend our definition to account for multiple messages.

How does one extend the definition to consider adversaries that may *adaptively* choose the message pairs?

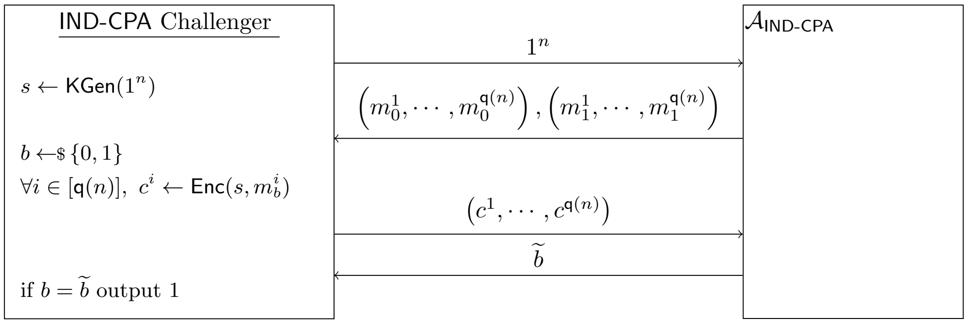
Definition 32 (Multi-message Secure Encryption). *A secret-key encryption scheme* (KGen, Enc, Dec) *is multi-message* IND-CPA *secure if for all non-uniform* PPT *adversaries* $\mathcal{A}$, *for all polynomials* $\mathsf{q}(\cdot)$ *there exists a negligible function* negl *such that:*

$$\Pr\left[\begin{array}{c} s \xleftarrow{\$} \mathsf{KGen}(1^n), \\ \{(m_0^i, m_1^i)\}_{i=1}^{\mathsf{q}(n)} \leftarrow \mathcal{A}(1^n), \\ b \xleftarrow{\$} \{0,1\} \end{array} : \mathcal{A}\left(\{\mathsf{Enc}(m_b^i)\}_{i=1}^{\mathsf{q}(n)}\right) = b \right] \leq \frac{1}{2} + \mathsf{negl}(n)$$

This definition is very similar to our first definition, but now in the security game the adversary sends two arrays of messages,$(m_0^1, \cdots, m_0^{\mathsf{q}(n)})$ and $(m_1^1, \cdots, m_1^{\mathsf{q}(n)})$ and the challenger encrypts one of the arrays and returns an array of ciphertexst $(c^1, \ldots, c^{\mathsf{q}(n)})$. We let $\mathsf{q}(\cdot)$ be any arbitrary polynomial chosen by the adversary $\mathcal{A}$. The game is shown in Figure 7.3.

Theorem 14 (Stateful Multi-message Encryption). *There exists a multi-message secret-key encryption scheme based on* PRGs *where the encryption algorithm is stateful.*

The proof of the above theorem is straightforward and left as an exercise. Very roughly, the idea is that we can expand the key to a sufficiently long pseudorandom string using a PRG (as in the previous construction) and then use different *chunks* of the PRG output to encrypt different messages. We need to keep track



IND-CPA Challenger
$s \leftarrow \mathsf{KGen}(1^n)$
$b \leftarrow\$ \{0,1\}$
$\forall i \in [\mathsf{q}(n)],\ c^i \leftarrow \mathsf{Enc}(s, m_b^i)$
if $b = \widetilde{b}$ output 1
$\mathcal{A}_{\text{IND-CPA}}$
1^n
$\left(m_0^1, \cdots, m_0^{\mathsf{q}(n)}\right), \left(m_1^1, \cdots, m_1^{\mathsf{q}(n)}\right)$
$\left(c^1, \cdots, c^{\mathsf{q}(n)}\right)$
$\widetilde{b}$

Figure 7.3: Multi-message Secure Encryption

of which chunk is used to encrypt which message, and therefore the encryption algorithm is stateful.

In practice, however, having a stateful encryption algorithm is not very desirable. Instead, we would like to construct multi-message encryption schemes where the encryption algorithm is stateless. The theorem below states that such an encryption scheme must also have a randomized encryption procedure.

Theorem 15 (Randomized Encryption). *A multi-message secure encryption scheme cannot be deterministic and stateless.*

Proof Suppose such a scheme existed. Then an adversary $\mathcal{A}$ could send (m_0^1, m_0^2) and (m_1^1, m_1^2) such that $m_0^1 = m_0^2$ and $m_1^1 \neq m_1^2$. Since no state is kept and the algorithm is entirely deterministic, this gives $\mathsf{Enc}(m_0^1) = \mathsf{Enc}(m_0^2)$ as $m_0^1 = m_0^2$. $\mathcal{A}$ could then just check if $c_1 = c_2$ thus breaking security (as this will be true if $b = 0$ and false with high probability otherwise). □

7.3.1 Encryption using PRFs

To construct a stateless multi-message secure encryption scheme, we will use a family of PRFs (Definition 24). Consider the following encryption scheme where the length of the message is n and $f_s : \{0,1\}^n \mapsto \{0,1\}^n$ is the PRF family:

```
KGen(1^n)
1 :  $s \leftarrow\$ \{0,1\}^n$
2 :  Return $s$

Enc(s, m)
1 :  $r \leftarrow\$ \{0,1\}^n$
2 :  $c := f_s(r) \oplus m$
3 :  Return $(r, c)$

Dec(s, (r, c))
1 :  $m := f_s(r) \oplus c$
2 :  Return $m$
```

Theorem 16. *Let* (KGen, Enc, Dec) *be as constructed based on a secure* PRF *family, then it is a multi-message secure encryption scheme.*

Proof The proof is done via a forward hybrid argument. Let F be a truly random function. Consider the following list of hybrids:

```
H_0
1 : $s \leftarrow \mathsf{KGen}(1^n)$
2 : $\forall i \in [\mathsf{q}(n)]$
3 :     $r^i \leftarrow\$ \{0,1\}^n$
4 :     $R^i := f_s(r^i)$
5 :     $c^i := m_0^i \oplus R^i$
```

```
H_1
1 : $s \leftarrow \mathsf{KGen}(1^n)$
2 : $\forall i \in [\mathsf{q}(n)]$
3 :     $r^i \leftarrow\$ \{0,1\}^n$
4 :     $R^i := F(r^i)$
5 :     $c^i := m_0^i \oplus R^i$
```

```
H_2
1 : $s \leftarrow \mathsf{KGen}(1^n)$
2 : $\forall i \in [\mathsf{q}(n)]$
3 :     $r^i \leftarrow\$ \{0,1\}^n$
4 :     $R^i \leftarrow\$ \{0,1\}^n$
5 :     $c^i := m_0^i \oplus R^i$
```

```
H_3
1 : $s \leftarrow \mathsf{KGen}(1^n)$
2 : $\forall i \in [\mathsf{q}(n)]$
3 :     $r^i \leftarrow\$ \{0,1\}^n$
4 :     $R^i \leftarrow\$ \{0,1\}^n$
5 :     $c^i := m_1^i \oplus R^i$
```

```
H_4
1 : $s \leftarrow \mathsf{KGen}(1^n)$
2 : $\forall i \in [\mathsf{q}(n)]$
3 :     $r^i \leftarrow\$ \{0,1\}^n$
4 :     $R^i := F(r^i)$
5 :     $c^i := m_1^i \oplus R^i$
```

```
H_5
1 : $s \leftarrow \mathsf{KGen}(1^n)$
2 : $\forall i \in [\mathsf{q}(n)]$
3 :     $r^i \leftarrow\$ \{0,1\}^n$
4 :     $R^i := f_s(r^i)$
5 :     $c^i := m_1^i \oplus R^i$
```

The output of each hybrid is the array $((r^1, c^1), \cdots, (r^{\mathsf{q}(n)}, c^{\mathsf{q}(n)}))$. From the description of the hybrids, it is clear that H_0 corresponds to the encryption of the array $(m_0^1, \cdots, m_0^{\mathsf{q}(n)})$ while H_5 corresponds to the encryption of the array $(m_1^1, \cdots, m_1^{\mathsf{q}(n)})$.

The proof follows in a similar manner to that of the one-message security of PRG. We sketch in Figure 7.4 how one proves that hybrids H_0 and H_1 are computationally indistinguishable by relying on the security of the PRF. Specifically we use an adversary $\mathcal{A}_{\mathsf{H}_0,\mathsf{H}_1}$ that distinguishes between H_0 and H_1 to construct an adversary $\mathcal{A}_{\mathsf{PRF}}$ that breaks the security of the PRF.

Sketch out the full proof to make sure you understand it.

□

7.4 Semantic Security

We look at the final notion of security for encryption, *semantic security* that we define below.

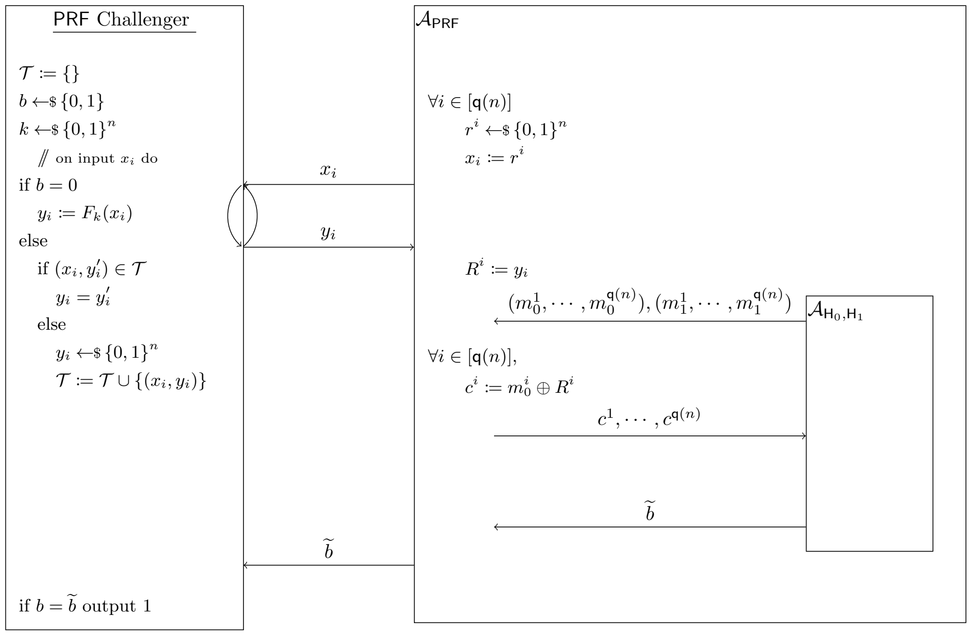

PRF Challenger
$\mathcal{T} := \{\}$
$b \leftarrow\$ \{0,1\}$
$k \leftarrow\$ \{0,1\}^n$
// on input x_i do
if $b = 0$
$y_i := F_k(x_i)$
else
if $(x_i, y_i') \in \mathcal{T}$
$y_i = y_i'$
else
$y_i \leftarrow\$ \{0,1\}^n$
$\mathcal{T} := \mathcal{T} \cup \{(x_i, y_i)\}$
if $b = \widetilde{b}$ output 1
x_i
y_i
$\widetilde{b}$
$\mathcal{A}_{\mathsf{PRF}}$
$\forall i \in [\mathsf{q}(n)]$
$r^i \leftarrow\$ \{0,1\}^n$
$x_i := r^i$
$R^i := y_i$
$(m_0^1, \cdots, m_0^{\mathsf{q}(n)}), (m_1^1, \cdots, m_1^{\mathsf{q}(n)})$
$\forall i \in [\mathsf{q}(n)]$,
$c^i := m_0^i \oplus R^i$
$c^1, \cdots, c^{\mathsf{q}(n)}$
$\widetilde{b}$
$\mathcal{A}_{\mathsf{H}_0,\mathsf{H}_1}$

Figure 7.4: Indistinguishability of H_0 and H_1

Definition 33 (Semantic Security). *A secret-key encryption scheme* $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *is* semantically secure *if there exists a* PPT *simulator algorithm* Sim *s.t. the following two experiments generate computationally indistinguishable outputs:*

$$\left\{ \begin{array}{r} (m, z) \leftarrow M(1^n), \\ s \leftarrow \mathsf{KGen}(1^n), \\ (\mathsf{Enc}(s, m), z) \end{array} \right\} \approx_c \left\{ \begin{array}{r} (m, z) \leftarrow M(1^n), \\ \mathsf{Sim}(1^n, z) \end{array} \right\}$$

where M is a machine that randomly samples a message from the message space and arbitrary auxiliary information z.

Intuitively, semantic security promises that a PPT adversary does not learn any “new” information about a message m by simply looking at its ciphertext. To capture the fact that it might already have some information about the underlying message, we denote by z any arbitrary auxiliary information. This intuition is captured by the fact that the adversary’s view (ciphertext and auxiliary information z) can be efficiently simulated by Sim given only z and no other information on m.

We now show that semantic security and $\mathsf{IND\text{-}CPA}$ security are, in fact, equivalent security notions.

Theorem 17. *Semantic security and* $\mathsf{IND\text{-}CPA}$*security are equivalent.*

Proof We prove each case separately:

Semantic security $\Rightarrow$ $\mathsf{IND\text{-}CPA}$: From semantic security, we have that for any m_1, $(\mathsf{Enc}(m_0), z) \approx_c \mathsf{Sim}(1^n, z)$. Also from semantic security, for any m_1, we have that $(\mathsf{Enc}(m_1), z) \approx_c \mathsf{Sim}(1^n, z)$. Thus, by transitivity, we get that semantic security implies $\mathsf{IND\text{-}CPA}$ security.

$\mathsf{IND\text{-}CPA}$ $\Rightarrow$ **semantic security** : From $\mathsf{IND\text{-}CPA}$ security, we have that encryptions of any two messages are indistinguishable. Then, we simply construct a simulator that encrypts the all zeros string, i.e., $\mathsf{Sim}(1^n, z) = (\mathsf{Enc}(0^n), z)$. Semantic security then immediately follows from $\mathsf{IND\text{-}CPA}$ security.

□

Chapter 8

Public-Key Encryption

Recall that in the setting of private-key encryption, Alice and Bob were allowed to share a secret before communication. In the public-key setting, Alice and Bob don't share a secret any more. Our goal is to be able to have Alice still send a message to Bob in such a manner that no eavesdropper can distinguish between encryptions of m and m'. We formalize the notion of public-key encryption that allows us to do this.

The syntax is similar to the secret-key encryption, with a few crucial differences.

Syntax A secret-key encryption consists of three algorithms described below:

Key Generation. $\mathsf{KGen}(1^n)$ is a randomized algorithm that takes no input, and outputs a sampled (public-key,secret-key) pair pk, sk.

Encryption. $\mathsf{Enc}(\mathsf{pk}, m)$ is a randomized algorithm takes as input a public-key pk, message m and outputs a ciphertext c.

Decryption. $\mathsf{Dec}(\mathsf{sk}, c)$ is a deterministic algorithm that takes as input a secret-key sk, ciphertext c and outputs a message m or a special symbol $\perp$ indicating failure to decrypt.

After looking at the security definition, consider if Enc can be a determinstic algorithm in the public-key setting.

As before, we require that all the algorithms run in polynomial time.
A public-key encryption scheme must satisfy the two properties described below:

Correctness We require that the scheme satisfies correctness, where decrypting an encrypted message should yield the message.

$$\forall m,\ \mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m,$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^n)$.

Security Again, we want to the encryption scheme to have security against an eavesdropper Eve. We now provide our definition of security. We start with a weak definition (also called selective security).

Definition 34. *A public key encryption scheme is weakly-indistinguishably secure under chosen plaintext attack* ($\mathsf{IND\text{-}CPA}$) *if for all non uniform* PPT *adversaries* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\Pr\left[\begin{array}{r} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^n), \\ (m_0, m_1) \leftarrow \mathcal{A}(1^n), \\ b \leftarrow\$ \{0,1\} \end{array} : \mathcal{A}(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, m_b)) = b\right] \leq \frac{1}{2} + \mathsf{negl}(n)$$

This definition is very similar to the $\mathsf{IND\text{-}CPA}$ definition (Definition 31) for secret-key encryption. The primary difference being that that $\mathcal{A}$ is additionally given the public-key pk along with the ciphertext. The corresponding game is provided in Figure 8.1.

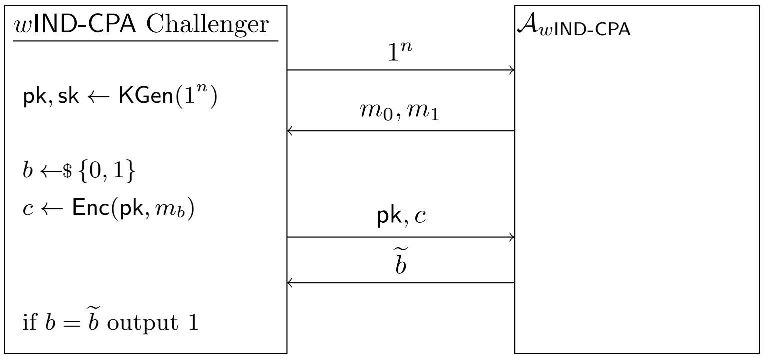

$w\mathsf{IND\text{-}CPA}$ Challenger
$\mathcal{A}_{w\mathsf{IND\text{-}CPA}}$
1^n
$\mathsf{pk}, \mathsf{sk} \leftarrow \mathsf{KGen}(1^n)$
m_0, m_1
$b \leftarrow\$ \{0,1\}$
$c \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$
pk, c
$\widetilde{b}$
if $b = \widetilde{b}$ output 1

Figure 8.1: Weakly-indistinguishably PKE scheme under chosen plaintext attack ($\mathsf{IND\text{-}CPA}$).

We think of this as a weaker notion since the adversary has to choose its message pair prior to looking at the public-key, which as the name suggests is public. We now adapt the above definition to allow the adversary to choose its messages adaptively based on the public-key. send us messages.

Definition 35. *A public key encryption scheme is indistinguishably secure under chosen plaintext attack* ($\mathsf{IND\text{-}CPA}$) *if for all non uniform* PPT *adver-*

saries $\mathcal{A}$ there exists a negligible function negl *such that*

$$\Pr\left[\begin{array}{r}(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KGen}(1^n),\\ (m_0, m_1) \leftarrow \mathcal{A}(1^n, \mathsf{pk}),\\ b \leftarrow\!\$\, \{0,1\}\end{array} : \mathcal{A}(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, m_b)) = b\right] \le \frac{1}{2} + \mathsf{negl}(n)$$

The corresponding game is provided in Figure 8.2.

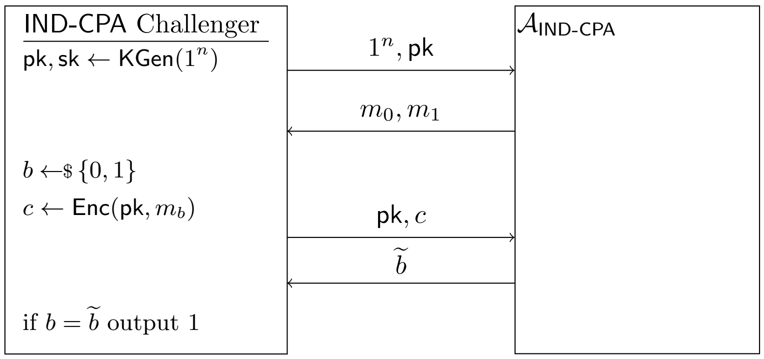

IND-CPA Challenger
$\mathsf{pk}, \mathsf{sk} \leftarrow \mathsf{KGen}(1^n)$
$b \leftarrow\!\$\, \{0,1\}$
$c \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$
if $b = \widetilde{b}$ output 1
$\mathcal{A}_{\text{IND-CPA}}$
$1^n, \mathsf{pk}$
m_0, m_1
pk, c
$\widetilde{b}$

Figure 8.2: Indistinguishably PKE scheme under chosen plaintext attack (IND-CPA).

Just as in the secret key encryption schemes we want to obtain multiple message security as well. However examining our definition we can realize that one-message security already implies multi-message security. Multi-message security is defined analogously to the secret-key encryption setting, and we omit the definition here.

Lemma 18. *One-message security implies multi-message security for public-key encryption*

Proof We now briefly sketch the proof. For simplicity, we only show that one-message security implies two-message security. The general case can be derived in a similar manner.

Suppose that (m_0^1, m_1^1) is the first message pair and (m_0^2, m_1^2) is the second message pair. Now, consider the following sequence of hybrid experiments:

H_0	H_1	H_2
1 : $c^1 \leftarrow \mathsf{Enc}(\mathsf{pk}, m_0^1)$	1 : $c^1 \leftarrow \mathsf{Enc}(\mathsf{pk}, m_1^1)$	1 : $c^1 \leftarrow \mathsf{Enc}(\mathsf{pk}, m_1^1)$
2 : $c^1 \leftarrow \mathsf{Enc}(\mathsf{pk}, m_0^2)$	2 : $c^1 \leftarrow \mathsf{Enc}(\mathsf{pk}, m_0^2)$	2 : $c^1 \leftarrow \mathsf{Enc}(\mathsf{pk}, m_1^2)$

H_0 corresponds to the case in the multi-message game when $b = 0$, and H_2 corresponds to the case that $b = 1$. We start by showing that hybrids H_0 and H_1 are indistinguishable from the one-message security of the public-key encryption scheme. Specifically, if there is an adversary $\mathcal{A}_{\mathsf{H}_0,\mathsf{H}_1}$ that distinguishes between hybrids H_0 and H_1, we use $\mathcal{A}_{\mathsf{H}_0,\mathsf{H}_1}$ to construct an adversary $\mathcal{A}_{\text{1-msg-IND-CPA}}$ that breaks the one-message security of the encryption scheme. The reduction is presented in Figure 8.3.

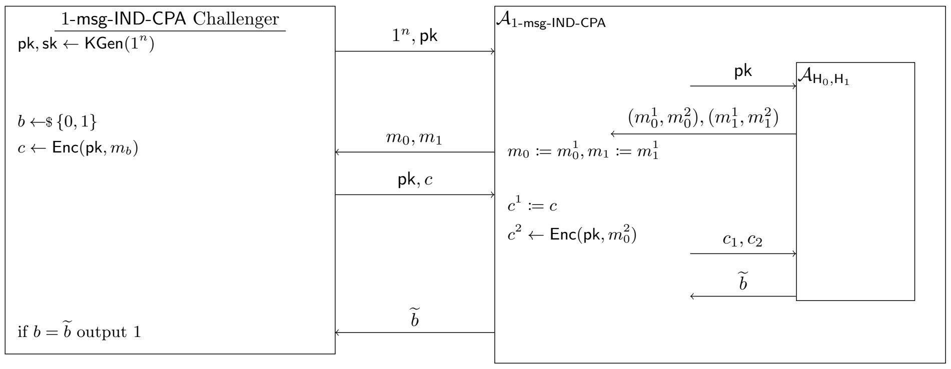

1-msg-IND-CPA Challenger
$\mathsf{pk}, \mathsf{sk} \leftarrow \mathsf{KGen}(1^n)$
$b \leftarrow\$ \{0,1\}$
$c \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$
if $b = \widetilde{b}$ output 1
$1^n, \mathsf{pk}$
m_0, m_1
pk, c
$\widetilde{b}$
$\mathcal{A}_{\text{1-msg-IND-CPA}}$
pk
$(m_0^1, m_0^2), (m_1^1, m_1^2)$
$m_0 := m_0^1, m_1 := m_1^1$
$c^1 := c$
$c^2 \leftarrow \mathsf{Enc}(\mathsf{pk}, m_0^2)$
c_1, c_2
$\widetilde{b}$
$\mathcal{A}_{\mathsf{H}_0,\mathsf{H}_1}$

Figure 8.3: Indistinguishability of hybrids H_0 and H_1.

In the reduction note that if c is an encryption of $m_0 = m_0^1$, then the above corresponds to H_0, otherwise, it corresponds to H_1. Therefore, if $\mathcal{A}_{\mathsf{H}_0,\mathsf{H}_1}$ distinguishes with noticeable probability, then so does$\mathcal{A}_{\text{1-msg-IND-CPA}}$. This contradicts the one-message security of the encryption scheme.

Think of why this approach doesn't work in the secret-key setting. Can you compute the other ciphertexts in the hybrids there?

The indistinguishability of hybrids H_1 and H_2 follow in an identical manner. Therefore, by the Hybrid Lemma (Lemma 9) we get that H_0 and H_2 are indistinguishable, as desired.

□

8.1 Trapdoor Permutations

Before we can construct a public-key encryption scheme, we shall need some new tools that we define now. We start by defining a collection of one-way functions and permutations, and then proceed to give a definition of trapdoor permutations.

The following definition extends the notion of one-way function to a family of functions. Note that we discussed a family of functions when we talked about pseudorandom functions (PRFs).

Definition 36 (Collection of one-way functions). *A collection of one way functions is a family of functions*

$$\mathcal{F} = \{f_i : D_i \to R_i\}_{i \in \mathcal{I}}$$

that satisfy the following conditions:

Sampling Function: *There exists a* PPT *algorithm* Gen *such that* $\mathsf{Gen}(1^n)$ *outputs* $i \in \mathcal{I}$.

Sampling from Domain: *There exists a* PPT *algorithm that on input* i *outputs a uniformly random element of* D_i.

Evaluation: *There exists a* PPT *algorithm that on input* $i, x \in D_i$ *outputs* $f_i(x)$.

Hard to invert: *For every non-uniform* PPT *adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *s.t.*

$$\Pr[i \leftarrow \mathsf{Gen}(1^n), x \leftarrow D_i : f_i(\mathcal{A}(1^n, f_i(x), i)) = f_i(x)] \leq \mathsf{negl}(n).$$

Let us see how this notion relates to our familiar notion of one-way functions (Definition 13).

Theorem 19. *There exists a collection of one-way functions if and only if there exists a strong one-way function.*

Proof The forward direction is obvious. If there exists a strong one-way function we can have the collection that is just the single function, and therefore collections of one way functions exist.

For the other direction we must construct a single one-way function given a collection $\mathcal{F}$. To do this define the one-way function g as

$$g(r_1, r_2) \coloneqq (i, f_i(x)),$$

where $f_i \in \mathcal{F}$, $i \coloneqq \mathsf{Gen}(1^n; r_1)$ and x is sampled randomly from D_i using r_2. This is a strong one way function because the ability to invert any single element of it would mean you could invert some $f_i \in \mathcal{F}$, which is not possible. □

This notion extends naturally to a collection of one-way permutations that we define below.

Definition 37 (Collection of one-way permutations). *A collection*

$$\mathcal{F} = \{f_i : D_i \to R_i\}_{i \in \mathcal{I}}$$

is a collection of one-way permutations if $\mathcal{F}$ is a collections of OWF, and for every $i \in \mathcal{I}$, f_i is a one-way permutation.

While the above definitions were similar to tools we've already looked at, we are now ready to define to define a completely new tool: trapdoor permutations.

Definition 38. *A collection of trapdoor permutations is a family of permutations*

$$F = \{f_i : D_i \to R_i\}_{i \in I}$$

satisfying the following conditions:

Sampling Function: *There exists a* PPT *algorithm* Gen *such that* $\mathsf{Gen}(1^n)$ *outputs* $(i, t) \in \mathcal{I}$.

Sampling from Domain: *There exists a* PPT *algorithm that on input* i *outputs a uniformly random element of* D_i.

Evaluation: *There exists a* PPT *algorithm that on input* $i, x \in D_i$ *outputs* $f_i(x)$.

Hard to invert: *For every non-uniform* PPT *adversary* $\mathcal{A}$, *there exists a negligible function* $\mathsf{negl}(\cdot)$ *s.t.*

$$\Pr[(i, t) \leftarrow \mathsf{Gen}(1^n), x \leftarrow D_i : f_i(\mathcal{A}(1^n, f_i(x), i)) = f_i(x)] \leq \mathsf{negl}(n).$$

Inversion with a trapdoor: *There exists a* PPT *algorithm that given* (i, t, y) *will output* $f_i^{-1}(y)$.

Roughly speaking, a trapdoor permutation is essentially a one way permutation that has a "trapdoor" t, that allows you to invert. We will often find it convenient to drop the index where convenient and sample a function and its corresponding trapdoor as $(f_i, f_i^{-1}) \leftarrow \mathsf{Gen}(1^n)$.

Trapdoor Permutations from RSA

Before we proceed to our construction of a public-key encryption scheme, we show that one can construct trapdoor permutations based on the RSA assumption. Before we can specify the assumption, we need to define the RSA collection.

Definition 39 (RSA **collection).** $\mathsf{RSA} := \{f_i : D_i \to R_i\}$ *where:*

- $\mathcal{I} := \left\{(N, e) \mid N = p \cdot q \text{ s.t. } p, q \in \Pi_n, e \in \mathbb{Z}^*_{\Phi(N)}\right\}$
- $D_i := \{x \mid x \in \mathbb{Z}^*_N\}$
- $R_i := \mathbb{Z}^*_N$
- $((N, e), d) \leftarrow \mathsf{Gen}(1^n)$ *where* $(N, e) \in \mathcal{I}$ *and* $ed \mod \Phi(N) = 1$
- $f_{N,e}(x) = x^e \mod N$
- $f^{-1}_{N,e}(y) = y^d \mod N$

Recall that Π_n is the set of all n-bit primes. It is easy to verify that that $f_{N,e}$ is a permutation.

We can now state the RSA assumption:

Definition 40 (RSA **assumption).** *For any n.u. PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *s.t.:*

$$\Pr\left[\begin{array}{c} p, q \leftarrow\$\, \Pi_n, N = pq,\ e \leftarrow\$\, \mathbb{Z}^*_{\Phi(N)} \\ y \leftarrow\$\, \mathbb{Z}^*_N; x \leftarrow \mathcal{A}(N, e, y) \end{array} : x^e = y \mod N\right] \leq \mathsf{negl}(n).$$

Finally, we note that based on the RSA assumption, we can establish that the RSA collection is a family of trapdoor permutations. The proof is left as an exercise.

Theorem 20. *Assuming the* RSA *assumption, the* RSA *collection is a family of trapdoor permutations.*

8.2 Public-key Encryption from Trapdoor Permutations

We will now show how to build a public key encryption scheme from a family of one way trapdoor permutations.

Let $\mathcal{F} = \{f_i : D_i \to R_i\}\ i \in I$ be a family of trapdoor permutations and let h_i be the hardcore predicate associated with f_i:

```
KGen(1^n)
1 :  $(f_i, f_i^{-1}) \leftarrow \mathsf{Gen}(1^n)$
2 :  $\mathsf{pk} := f_i, h_i$
3 :  $\mathsf{sk} := f_i^{-1}$
4 :  Return $(\mathsf{pk}, \mathsf{sk})$
```

```
Enc(pk, m)
1 :  $r \leftarrow\$ \{0,1\}^n$
2 :  $c_1 := f_i(r)$
3 :  $c_2 := h_i(r) \oplus m$
4 :  Return $(c_1, c_2)$
```

```
Dec(sk, (c_1, c_2))
1 :  $r := f_i^{-1}(c_1)$
2 :  $m := h_i(r) \oplus c_2$
3 :  Return $m$
```

Theorem 21. *Let* (KGen, Enc, Dec) *be as constructed based on a secure family of trapdoor permutations, then it is an* IND-CPA *secure public-key encryption scheme.*

Proof The proceeds by a hybrid argument. Consider the following sequence of hybrids:

```
H_0
1 :  $r \leftarrow\$ \{0,1\}^n$
2 :  $c_1 := f_i(r)$
3 :  $z := h_i(r)$
4 :  $c_2 := m_0 \oplus z$
```

```
H_1
1 :  $r \leftarrow\$ \{0,1\}^n$
2 :  $c_1 := f_i(r)$
3 :  $z \leftarrow\$ \{0,1\}$
4 :  $c_2 := m_0 \oplus z$
```

```
H_2
1 :  $r \leftarrow\$ \{0,1\}^n$
2 :  $c_1 := f_i(r)$
3 :  $z \leftarrow\$ \{0,1\}$
4 :  $c_2 := m_1 \oplus z$
```

```
H_3
1 :  $r \leftarrow\$ \{0,1\}^n$
2 :  $c_1 := f_i(r)$
3 :  $z := h_i(r)$
4 :  $c_2 := m_1 \oplus z$
```

The output of each hybrid is c_1, c_2. From the description of the hybrids, it is clear that H_0 corresponds to the encryption of m_0 while H_3 corresponds to the encryption of m_1.

Make sure to work out the details of the reduction and understand why $\mathcal{A}_{\mathsf{hcp}}$ flips the value of its guess b depending on the value $\widetilde{b}$.

The proof follows in a similar manner to those we have seen before. We sketch in Figure 8.4 how one proves that hybrids H_0 and H_1 are computationally indistinguishable by relying on the security of the hardcore predicate. Specifically we use an adversary $\mathcal{A}_{\mathsf{H}_0,\mathsf{H}_1}$ that distinguishes between H_0 and H_1 to construct an adversary $\mathcal{A}_{\mathsf{hcp}}$ that breaks the security of the hardcore predicate of f_i. Note that extending the notion of hardcore predicates to a collection, the challenger provides the predicate to the adversary. The analysis follows as in the proof of Theorem 10.

The indistinguishability of H_1 and H_2 follows immediately from the security of one-time pad. Finally, the indistinguishability of H_2 and H_3 can be argued in an analogous manner as for H_0 and H_1.

Combining the above, we obtain that H_0 and H_3 are indistinguishable, thereby proving the IND-CPA security of our constructed scheme. □

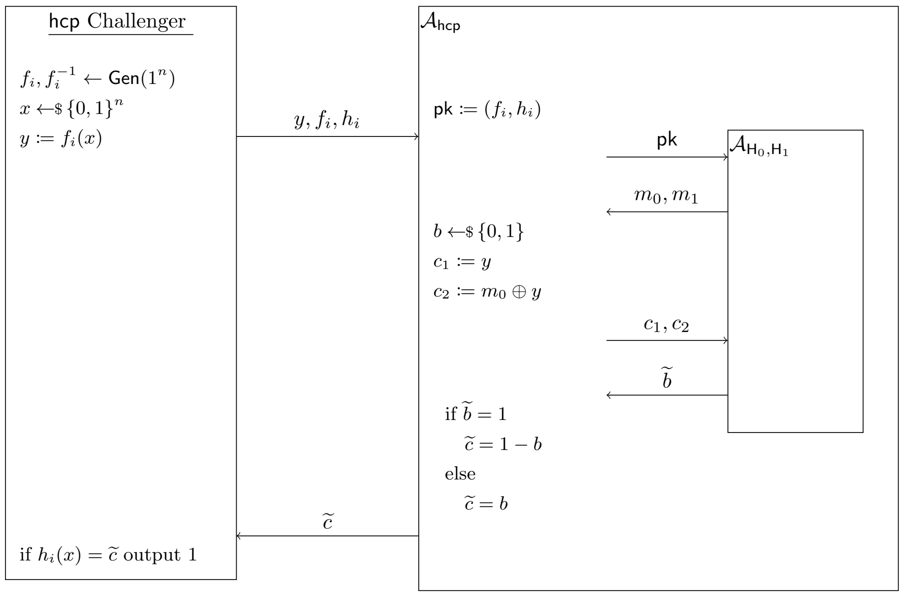

hcp Challenger
$f_i, f_i^{-1} \leftarrow \mathsf{Gen}(1^n)$
$x \leftarrow\$ \{0,1\}^n$
$y \coloneqq f_i(x)$
y, f_i, h_i
$\mathcal{A}_{\mathsf{hcp}}$
$\mathsf{pk} \coloneqq (f_i, h_i)$
pk
$\mathcal{A}_{\mathsf{H}_0,\mathsf{H}_1}$
m_0, m_1
$b \leftarrow\$ \{0,1\}$
$c_1 \coloneqq y$
$c_2 \coloneqq m_0 \oplus y$
c_1, c_2
$\widetilde{b}$
if $\widetilde{b} = 1$
$\widetilde{c} = 1 - b$
else
$\widetilde{c} = b$
$\widetilde{c}$
if $h_i(x) = \widetilde{c}$ output 1

Figure 8.4: Indistinguishability of H_0 and H_1

Chapter 9

Authentication

9.1 Introduction

We begin by describing the setting. There are two parties, Alice and Bob. Alice wants to send a message to Bob over a public channel such that an active adversary Eve cannot tamper with the message without being detected. In particular, Eve should not be able to replace the message sent by Alice with another message and trick Bob into thinking that it actually came from Alice.

We will discuss two methods using which Bob will be able to verify whether or not the message he receives is sent by Alice. The first method, called *message authentication codes*, requires Alice and Bob to share a secret key. The second method, called *digital signatures*, is the public-key analogue, where Alice will use a secret key and Bob will use a corresponding public key.

Note: We will abbreviate Alice = A, Bob = B, Eve = E.

9.2 Private Key: MAC

We first consider the private key scenario. We will discuss message authenticated codes, or MAC, where A and B share a private key that is used both for signing and verifying.

9.2.1 Algorithm overview

MACs have three relevant algorithms. First, a key generator Gen() that outputs a private key. Second, we have a tag generator Tag() that takes as input a signing key and a message and outputs a signature. Finally, there is a verification algorithm that takes as input a verification key, a message and a signature and outputs 1 if the signature is valid, and 0 otherwise.

- Key Generation: $k \leftarrow$ Gen(1^n), generates a secret key
- Signing: $\sigma \leftarrow$ Tag$_k$(m), computes a tag for the message m
- Verification: Ver$_k$(m, σ) = 1 iff σ is a valid tag of m over k, else 0.

MACs also have the following two properties: Correctness, and Security. With respect to security, we will discuss Unforgability under Chosen-Message Attack (UF-CMA).

Definition 41 (MAC). *Given a key k and a signature σ, MACs have the following properties:*

- ***Correctness:*** *Pr[$k \leftarrow$ Gen(1^n), $\sigma \leftarrow$ Tag$_k(m)$: Ver$_k(m, \sigma)$ = 1] = 1*
- ***Security:*** *$\forall$ nu PPT adversary A, $\exists$ a negligible function $\mu(\cdot)$ such that: Pr[$k \leftarrow$ Gen(1^n), $(m, \sigma) \leftarrow A^{Tag_k(\cdot)}$($1^n$) : A did not query $m \wedge$ Ver$_k(m, \sigma)$ = 1] $\leq \mu(n)$*

Another way of looking at the security of MACs is via the following security game. Let Ch = Challenger, A = Adversary.

1. Ch generates a key: $k \leftarrow$ Gen(1^n)
2. A sends a message m_i to Ch
3. Ch generates a tag: $\sigma_i \leftarrow$ Tag$_k(m_i)$
4. Ch sends σ_i to A
5. A and Ch repeat steps 2 to 4 multiple times
6. A finally sends Ch (m^*, σ^*)

A wins if, for all i, $m^* \neq m_i$, and Ver$_k(m^*, \sigma^*) = 1$. Unforgeability under Chosen-Message Attack (UF-CMA) security states that the probability that A wins the above game is negligible.

9.3 Construction of MAC

We now construct a MAC scheme based on pseudorandom functions (PRFs).

Theorem 22. *PRF $\Rightarrow$ MAC*

In order to prove the above theorem, we start by giving a construction of MAC scheme based on PRFs.

Construction. Let $\{f_k\}$ be a family of PRFs. The MAC scheme is described below:

1. Gen(1^n) : Output k $\overset{\$}{\leftarrow}$ {0, 1}^{n}
2. Tag$_k$(m) : Output f$_k(m)$
3. Ver$_k$(m, σ) : Output f$_k(m) \overset{?}{=} \sigma$

We now prove the security of the above construction.

Proof of Security. Let A_{MAC} be a PPT adversary that breaks the security of MAC. We want to construct an adversary A_{PRF} that uses A_{MAC} to break the security of PRFs and thus reach a contradiction and conclusion.

Whenever A_{MAC} queries A_{PRF} with a message m_i, A_{PRF} queries its challenger oracle with x_i = m_i. A_{PRF} will receive the oracle's output y_i. A_{PRF} will then forward σ_i = y_i to A_{MAC}. This process repeats several times. Now, A_{MAC} outputs (m^*, σ^*). At this point, A_{PRF} queries its challenger on $x^* = m^*$ and obtains y^*. If $\sigma^* = y^*$, then A_{PRF} outputs "PRF", else it outputs "RF".

If the Challenger oracle is random, then A_{MAC} has no information on the PRF key, and therefore the probability that it can output a valid signature σ^* is negligible.

If the Challenger oracle is pseudo-random, then our A_{MAC} above will be able to output a valid σ with non-negligible probability. This is a contradiction on the security of PRFs. □

9.3.1 One Time MACs

We can also consider MACs with a weaker security definition where the adversary is only allowed one signing query to the challenger. The advantage is that we can construct one-time MACs with unconditional security. This is the analogue to One Time Pads for authentication.

9.4 Public Key: Digital Signature

Now, we discuss the public key scenario, or digital signatures. The intuition here is that "A uses a private key to sign, and B uses a public key to verify."

9.4.1 Algorithm overview

Like MACs, we have three relevant algorithms: a key generator, a sign generator, and a verifier. Note the differences below.

Our key generator now produces a pair containing a secret key sk and a public key pk. The tag generator produces a signature using the secret key. Our verifier verifies the message using the public key.

- Key Generation: $(sk, pk) \leftarrow$ Gen(1^n)
- Signing: $\sigma \leftarrow$ Sign$_{sk}(m)$, a signature
- Verification: Ver$_{pk}(m, \sigma)$: $M \times S \to \{0, 1\}$, where M is the message space and S the signature space

Definition 42 (Digital Signatures). *Given a key (sk, pk) and a signature σ, Digital Signatures have the following properties:*

- ***Correctness:*** *Pr[$(sk, pk) \leftarrow$ Gen(1^n), $\sigma \leftarrow$ Sign$_{sk}(m)$: Ver$_{pk}(m, \sigma)$ = 1] = 1*
- ***Security:*** *UF-CMA as described above.*
  $\forall$ nu PPT adversaries A, $\exists$ a negligible function $\mu(\cdot)$ such that:
  Pr[$(sk, pk) \leftarrow$ Gen(1^n), $(m, \sigma) \leftarrow A^{Sign_{sk}(\cdot)}(1^n)$: A did not query m $\wedge$ Ver$_{pk}(m, \sigma)$ = 1] $\leq \mu(n)$

9.5 One Time Signatures

Next, we move on into a discussion of One Time Signatures, which is a digital signature scheme where the adversary can only make one signature query. Below we discuss Lamport's one-time signature scheme based on one-way functions.

Lamport's Construction of OTS. Let f be a OWF. We make the assumption that we will only sign n-bit messages.

- Key Generation:

$$sk = \begin{bmatrix} x_1^0 & x_2^0 & \dots & x_n^0 \\ x_1^1 & x_2^1 & \dots & x_n^1 \end{bmatrix}$$

  Where $x_i^b \xleftarrow{\$} \{0, 1\}^n, \forall i \in [n]$ and $b \leftarrow \{0, 1\}$.

$$pk = \begin{bmatrix} y_1^0 & y_2^0 & \dots & y_n^0 \\ y_1^1 & y_2^1 & \dots & y_n^1 \end{bmatrix}$$

  Where $y_i^b = f(x_i^b)$.
- Signing: Sign$_{sk}(m) : \sigma := (x_1^{m_1}, x_2^{m_2}, \dots, x_n^{m_n})$
  Where $\mid \sigma \mid= n$. Basically, for every column of sk, select one of the two x_i^b depending on m_i.
- Verification: Ver$_{pk}(m, \sigma) : \wedge_{i \in [n]} f(\sigma_i) \overset{?}{=} y_i^{m_i}$.

Proof of Security. We will prove security w.r.t. a weaker security definition called selective security. We describe the security game below:

1. Adversary A states what message m^* it is going to forge and sends that to Challenger Ch.
2. Ch generates and sends back a public key pk.
3. A queries Ch with a message m_i.
4. Ch returns with a sign σ_i.
5. A and Ch repeat steps 3 and 4 multiple times.
6. A finally sends Ch a σ^*.

We now prove that Lamport's construction is selectively secure.

For the sake of contradiction, suppose there is a PPT adversary A_{OTS} that can break the OTS security scheme. We construct a PPT adversary A_{OWF} that uses A_{OTS} to invert a OWF.

1. A_{OWF} receives $y = f(x)$ as input. This input will be embedded into an input for A_{OTS}. This latter input will be a public key that we will generate.
2. A_{OWF} runs A_{OTS} and gets $m^* = m_1^* \dots m_n^*$ where each m_i^* is one bit.
3. Choose $i \xleftarrow{\$} [n]$, and set $b^* = m_i^*$.
4. Set $y_i^{b^*} = y$. For all $j \neq i, b \in \{0,1\}$, choose an $x_j^b \xleftarrow{\$} \{0,1\}^n$ and calculate $y_j^b = f(x_j^b$. Further, choose $x_i^{1-b^*} \xleftarrow{\$} \{0,1\}^n$ and calculate $y_i^{1-b^*} = f(x_i^{1-b^*})$.
   Now, set pk to be the matrix of all y's as computed above.
5. Send this pk to A$_{OTS}$.
6. Now, A_{OTS} may send a signing query m. If $m_i = m_i^*$, then halt. Otherwise, answer the signing query as $\sigma = (x_1^{m_1}, \dots, x_n^{m_n})$.
7. A$_{OTS}$ outputs some σ^* as its forgery. A$_{OWF}$ will then set its $x = \sigma_i^*$ as its inversion.

Thus, if A$_{OTS}$ generates a valid forgery with noticeable probability ϵ, then A$_{OWF}$ can invert f with probability at least $\frac{\epsilon}{n}$, which is still noticeable. This is a contradiction. □

Note that the adversary A$_{OTS}$ declared the message m^* that it was going to forge at the beginning. The reduction didn't know the signing query m that A will send after that, therefore it guessed i such that $m_i \neq m_i^*$. We want this guess to be correct with noticeable probability, since otherwise, our reduction will fail almost always. If we choose i at random, then our guess will be correct with probability $\frac{1}{n}$. This is because A has no clue what i is, which is necessary - otherwise he may choose $m_i = m_i^*$ every time and thus the reduction will always fail.

So far, we have discussed authentication schemes for fixed length messages. But we continue our discussion for the case of arbitrary length messages. In order to construct signature schemes for arbitrary length messages, we first study the notion of collision-resistant hash functions (CRHFs).

9.6 Collision-Resistant Hash Function

A hash function is a compression function that shrinks a long message to a fixed length message. For our purposes, we need a hash function that ensures minimal collisions when shrinking messages. In particular, we want a hash function h where it is computationally hard to find two different inputs x and x' having the same outputs $h(x)$ and $h(x')$.

Definition 43 (CRHF Family). *A family $H = \{h_i : D_i \to R_i\}_{i \in I}$ is a collision resistant hash function family if:*

- *Easy to Sample: $\exists$ a PPT Gen algo s.t. $i \leftarrow Gen(1^n), i \in I$*
- *Compression: $\mid R_i \mid < \mid D_i \mid$*
- *Easy to Evaluate: $\exists$ a polytime algorithm $Eval$ s.t. given $x \in D_i$, $i \in I$, Eval(x, i) = $h_i(x)$*
- *Collision Resistance: $\forall$ nu PPT adversary A, $\exists$ negligible function μ s.t.*
  Pr[$i \xleftarrow{\$} Gen(1^n), (x, x') \leftarrow A(1^n, i) : x \neq x' \wedge h_i(x) = h_i(x')] \leq \mu(n)$

A couple of remarks on hash functions: first, hash functions with one-bit compression can be transformed to hash functions with arbitrary compression. One such mechanism is called Merkle trees. Note, however, that the output size of the hash must be large enough to prevent easy collisions.

One-time Signatures for Long Messages. Using CRHFs, we can transform Lamport's OTS scheme into a new OTS scheme where we can sign arbitrarily long messages. To sign a message m, we first hash it using $h_i(m)$ and then use the signing algorithm in Lamport signing scheme. We rely on the collision resistance of h to argue security.

Universal One-way Hash Functions. While CRHFs are not known based on one-way functions, there is a weaker notion of hash function that can be based on one-way functions and turns out to be sufficient for constructing digital signature schemes for long messages. This notion is called a universal one-way hash function (UOWHF), and it is defined in the same manner as a CRHF, except that the collision-resistance property is described as follows:

Definition 44 (Universal One-Way Hash Function (UOWHF)).

$$Pr[(x, state) \leftarrow A(1^n), i \overset{\$}{\leftarrow} Gen(1^n), x' \leftarrow A(i, state) : x \neq x' \wedge h_i(x) = h_i(x')] \leq \mu(n)$$

9.7 Multi-message Signatures

We end with a discussion on multi-message signatures. The definition of multi-message signatures is in the class lecture slides, as are relevant extra readings. A stateful construction of multi-message signatures can be constructed by creating a "chain" of one-time signatures (the construction is given in the class slides). Further, even stateless construction of multi-message signatures are possible, however they will not be covered in this course.

Chapter 10

Zero-Knowledge Proofs

10.1 What is a Proof?

Generally speaking, a proof is a demonstration of the veracity of statement through a line of deductive reasoning. In the realm of mathematics, this involves reducing the statement into a series of axioms and assumptions that are known to be valid.

Properties of Proofs. Here are two natural properties that a proof must satisfy: first, a verifier should accept the proof if the statement is true. This is known as Correctness. Second, if the statement is false, then any proof should be rejected by the verifier. This is known as Soundness.

It is also important that a proof can be efficiently verified. In particular, a proof that would clearly show if a statement is true, but cannot be verified efficiently is not acceptable. For example, consider the following proof for the existence of infinite primes: a list containing all primes. Clearly, both the generation of the proof and the verification would take an undefined amount of time, so this proof is not useful.

To ensure that proofs can be efficiently verified, we require that the verifier must be polynomial time in the length of the statement.

Must a proof be non-interactive? In exploring the structure of a proof, an important question that arises is whether a proof must be non-interactive? Or, can can a proof be in the form of a conversation between a prover and verifier, where at the end of the conversation the verifier is convinced.

Indeed, we will now formalize the notion of interactive proofs and show that they are extremely powerful!

10.2 Interactive Protocols

We start by establishing some notation and definitions related to interactive protocols.

Definition 45 (Interactive Turing Machine). *An interactive Turing machine (ITM) is a Turing machine with two additional tapes: a read-only tape to receives messages and a write-only one to send messages.*

Definition 46 (Interactive Protocol). *An interactive protocol is a pair of ITMs such that the read tape of the first ITM is the send tape of the second and vice-versa. An interactive protocol proceeds in rounds, where in each round only one ITM is active, while the other is idle. The protocol is finished when both machines halt.*

Definition 47 (Protocol execution). *A (randomized) execution of an interactive protocol (M_1, M_2) refers to the ITMs executing all the rounds of the protocol until they halt. An execution of (M_1, M_2) on inputs (x_1, x_2) and auxiliary inputs (z_1, z_2) is denoted as $M_1(x_1, z_1) \leftrightarrow M_2(x_2, z_2)$.*

Definition 48 (Protocol Output). *The output of M_i in an execution e of (M_1, M_2) is denoted as $Out_{M_i}(e)$*

Definition 49 (View of ITM). *The view of M_i in an execution e of (M_1, M_2) consists of its input, random tape, auxiliary tape and all the protocol messages it sees. It is denoted as $View_{M_i}(e)$.*

10.3 Interactive Proofs

Interactive proofs involve a pair of ITMs P and V, where P denotes the prover and V denotes the verifier.

Definition 50 (Interactive Proofs). *A pair of ITMs (P,V) is an interactive proof system for a language L if V is a PPT machine and the following properties hold:*

- *Completeness: For every x $\in L$,*

$$Pr[Out_V[P(x) \leftrightarrow V(x)] = 1] = 1$$

- *Soundness: There exists a negligible function $\nu(\cdot)$ s.t. $\forall x \notin L$ and for all adversarial provers P^*,*

$$Pr[Out_V[P^*(x) \leftrightarrow V(x)] = 1] \leq \nu(|x|)$$

Remark 11. *In this definition, the prover does not have to be efficient. The restriction of efficient provers will be visited later.*

Remark 12. *Note, however, that an* adversarial *prover can be unbounded*

What this definition is saying is that to satisfy the Completeness property, the output of V in the execution of the interactive protocol between P and V should always be accept as long as the statement x is a valid member of the language L of statements. In order to met the Soundness property, regardless of the adversarial prover P^*'s strategy, if x is not a valid statement then, the probability that the output of V in the execution of the interactive protocol between P^* and V is accept must be negligible.

10.3.1 Why Interactive Proofs?

A natural question that we should ask at this point is, why should we consider *interactive* proofs?

Indeed, for languages that are in NP, for each statement in the language there exists a polynomial sized witness. Specifically, for any NP language L with associated relation R and any statement $x \in L$, there exists a witness w s.t. checking $R(x, w) = 1$ confirms that $x \in L$. This means that w is a non-interactive proof for x.

Example. A simple example of non-interactive proofs using witnesses is Graph Isomorphism. Two Graphs are isomorphic if there is a permutation that maps one graph to the other. In this situation the permutation is the witness, as if a permutation can be shown then clearly a mapping must exist between the two graphs.

In light of the above, the question is why even bother with interactive proofs? Why not always use non-interactive proofs?
There are two main reasons for using interactive proofs:

- Proving statements for languages not known to be in NP (i.e., when a “short” witness is not available).
- Achieving a privacy guarantee for the prover

In particular, here are some known results that establish the power of interaction:

- Shamir proved that IP = PSPACE. That is the space of languages with interactive proof systems (with a single prover) is equivalent to the space of languages decidable in polynomial space.

– Babai-Fortnow-Lund established that MIP = NEXP. That is the space of languages with Multi-prover interactive proof systems is equivalent to the space of languages decidable in non-deterministic exponential time.
– Goldwasser-Micali-Rackoff presented the notion of Zero Knowledge, where verifier learns nothing from the proof beyond the validity of the statement.

In what follows, we will demonstrate the power of interaction by constructing interactive proofs for a language in co-NP, and then later, we will formalize the notion of zero knowledge.

Below, we first establish some general notation for graphs that we will later use.

10.4 Notation for Graphs

Definition 51 (Graph). *A Graph G = (V, E) where V is a set of vertices and E is a set of edges s.t.* $|V|$ *= n,* $|E|$ *= m*

Definition 52. Π_n *is the set of all permutations* π *over n vertices.*

Definition 53 (Graph Isomorphism). $G_0 = (V_0, E_0)$ *and* $G_1 = (V_1, E_1)$ *are isomorphic if there exists a permutation* π *s.t:*

– $V_1 = \{\pi(v) | v \in V_0\}$
– $E_1 = \{(\pi(v_1), \pi(v_2)) | (v_1, v_2) \in E_0\}$

Remark 13. *We will also use the notation* $G_1 = \pi(G_0)$

Graph Isomorphism is an NP problem, so even if we did not explain what the witness for this problem is, we know it must have one and thus can be proved non-interactively. However, there is a related problem that is not known to be in NP and thus cannot be efficiently proved using a witness.

Definition 54 (Graph Non-Isomorphism). G_0 *and* G_1 *are non-isomorphic if there exists no permutation* $\pi \in \Pi_n$ *s.t.* $G_1 = \pi(G_0)$

10.5 Interactive Proof for Graph Non-Isomorphism

Suppose we want to prove that two graphs G_0 and G_1 are not isomorphic. Note that graph non-isomorphism is in co-NP, and not known to be in NP.

A naive way to prove this is by enumerating all possible permutations over n vertices and showing that there is no permutation $\pi, G_1 \neq \pi(G_0)$. Note, however, that this cannot be verified efficiently.

Fortunately this is where the power of interaction comes in. We now demonstrate an interactive proof system for graph non-isomorphism.

Common Input: $x = (G_0, G_1)$

Protocol (P,V): Repeat the following procedure n times using fresh randomness

Verifier		Prover
Chooses a random $b \in \{0, 1\}$, $\pi \in \Pi_n$. Compute $H = \pi(G_b)$		
	$\xrightarrow{\text{send H}}$	
		Compute b' s.t. H and $G_{b'}$ are isomorphic
	$\xleftarrow{\text{Send } b'}$	

V(x,b,b'): V outputs 1 if b'=b and 0 otherwise.

We now argue that protocol (P, V) is an interactive proof. As per the definition, we have to establish that it satisfies the properties of Completeness and Soundness:

- Completeness: If G_0 and G_1 are not isomorphic, then an unbounded prover can always find b' s.t. b'=b. This is because H would only be isomorphic to one of the two graphs.
- Soundness: If G_0 and G_1 are isomorphic, then H is isomorphic to both G_0 and G_1. Thus in a single iteration, an unbounded prover can guess b with probability at most $1/2$. Since each iteration is independent, over n iterations, the probability of prover success is at most 2^{-n}, which is negligible.
- Additionally, the verifier is clearly efficient.

10.6 Interactive Proofs with Efficient Provers

Up until this point, the provers we were dealing with were inefficient. If there were not, then the previous protocol would have established that graph non-isomorphism is in NP.

However, what if we want Interactive proofs with efficient provers? One reason for this is because now, we can hope to implement prover strategies using standard computers or human beings (who are PPT machines). Further, we can hope to construct interactive proofs that are also zero knowledge.

In order to construct interactive proofs with efficient provers, we can only deal with languages in NP. In particular, for any statement x, we will provide a witness w for x as a private input to the prover. Then, we require that the prover strategy is be efficient when it is given a witness w for the statement x that it attempts to prove.

Definition 55. *An interactive proof system (P,V) for a language L witness relation R is said to have an efficient prover if P is a PPT and the completeness condition holds for every $w \in R(X)$*

Remark 14. *Even though honest P is efficient, we still require soundness guarantee against all possible adversarial provers.*

10.6.1 Interactive proof for Graph Isomorphism

We now construct an interactive proof for proving that two graphs G_0 and G_1 are isomorphic. At first, this may seem a little strange since as discussed earlier, there exists a simple non-interactive proof for the same: the prover simply sends the permutation that maps G_0 to G_1 to the verifier. Indeed, if the prover is provided this permutation as input, then it is already efficient.

The problem, however, with this protocol is that V learns the permutation π. Now using this permutation, it is able to repeat the proof to someone else.

This raises the natural question whether there is a way to interactively prove isomorphism without revealing the witness. Even better yet, can we construct construct a proof that reveals nothing to V beyond the validity of the statement?

Below, we construct such an interactive proof system for graph isomorphism.

Common Input: $x = (G_0, G_1)$

P's witness: $G_1 = \pi(G_0)$

Protocol (P,V): Repeat the following procedure n times using fresh randomness.

Prover

Verifier

Chooses a random $\sigma \in \Pi_n$,

Compute $H = \pi(G_0)$

$\xrightarrow{\text{send H}}$

Choose random bit $b \in \{0, 1\}$

$\xleftarrow{\text{send } b}$

$\xrightarrow{\text{if } b = 0\text{, sends } \sigma}$

$\xrightarrow{\text{else send } \phi = \sigma \cdot \pi^{-1}}$

$V(x, b, b')$: V outputs 1 iff $H = \phi(G_b)$

Proof of Completeness: If G_0 and G_1 are isomorphic, then V always accepts since $\sigma(G_0) = H$ and $\sigma(\pi^{-1}(G_1)) = \sigma(G_0) = H$.

Proof of Soundness: If G_0 and G_1 are not isomorphic, then H is isomorphic to one of the graphs but not both. Since b is chosen randomly after fixing H, H is not isomorphic to G_b with probability 1/2. Thus an unbounded adversarial prover can succeed with probability at most 1/2. Over n independent iterations, the prover can succeed with probability at most 2^{-1}.

In this protocol, one can see that intuitively, V obtained no information other than a random permutation of G_b. This is something he could have generated on its own, so intuitively, the protocol does not reveal any information.

Below, we formalize the idea of zero knowledge and then later, we will prove that the graph isomorphism protocol constructed above is in fact zero knowledge.

10.7 Zero Knowledge

Intuitively, a protocol is zero knowledge if the verifier does not “gain any knowledge” from interacting with the prover besides the validity of the statement. Towards formalizing this idea, the first natural question is how to formalize “does not gain any knowledge?”

Here are some rules to help in this direction:

- Rule 1: Randomness is for free
- Rule 2: Polynomial-time computation is for free

In other words, learning the result of a random process or a polynomial time computation gives us no knowledge.

The next question, however, is what is knowledge? To answer this question, let us understand when knowledge is conveyed.

- Scenario 1: Someone tells you he will sell you a 100-bit random string for $1000.
- Scenario 2: Someone tells you he will sell you the product of two prime numbers of your choice for $1000.
- Scenario 3: Someone tells you he will sell you the output of an exponential time computation (e.g., isomorphism between two graphs) for $1000.

Which of these offers should you accept?

Note that in the first scenario, we can generate a random string for free by flipping a coin. The second scenario can also be obtained for free since multiplying is a polynomial-time operation. However, since an exponential-time operation is hard to compute for a PPT machine, scenario 3 is the best one to consider.

The moral of the story is that we do not gain any information from an interaction if we could have performed it on our own. This leads us to the correct intuition for zero knowledge:

Intuition for Zero Knowledge: A protocol (P, V) is zero knowledge if V can generate a protocol transcript on its own, without talking to P. If this transcript is indistinguishable from a real execution, then clearly V does not learn anything by talking to P.

To formalize this intuition, we will use the idea of a Simulator as we did when defining semantic security for encryption.

Definition 56 (Honest Verifier Zero Knowledge). *An interactive proof (P,) for a language L with witness relation R is said to be honest verifier zero knowledge if there exists a PPT simulator S s.t. for ever n. u. PPT distinguisher D, there exists a negligible function $\nu(\cdot)$ s.t. for every $x \in L, w \in R(x), z \in 0, 1^*$, D distinguishes between the following distributions with probability at most $\nu(n)$:*

- $\{View_V[P(x, w) \leftrightarrow V(x, z)]\}$
- $\{S(1^n, x, z)\}$

In other words what V sees throughout the protocol is something that could have come up with on its own (by simply running the simulator with input x and z).

Remark 15. *The auxiliary input z to V captures any a priori information V may have about x. Definition promises that V does not gain any other knowledge.*

Issue. A problem with the above definition is that it promises security only of the verifier V follows the protocol. What if V is malicious and deviates from the honest strategy? In this case, we need a simulator S for every, possibly malicious (efficient) verifier strategy V^*.

We now present a definition of zero-knowledge for this case. For technical reasons, we allow the simulator to run in expected polynomial time.

Definition 57 (Zero Knowledge). *An interactive proof (P,V) for a language L with witness relation R is said to be zero knowledge if for every n.u. PPT adversary V^*, there exists an expected PPT simulator S s.t. for every n.u. PPT exists an expected PPT a negligible function $\nu(\cdot)$ s.t. for every $x \in L, w \in R(x), z \in 0, 1^*$, D distinguishes between the following distributions with probability at most $\nu(n)$:*

- $\{View_{V^*}[P(x,w) \leftrightarrow V^*(x,z)]\}$
- $\{S(1^n, x, z)\}$

Remark 16. *If the distributions are statistically close, then we call it statistical zero knowledge. If they are identical then it is know as perfect zero knowledge.*

We see that in this revised definition, no matter the verifier's strategy, the view of V^* is indistinguishable from the output of the simulator.

Theorem 23. *Sequential repetition of any ZK protocol is also ZK.*

Proof of Zero-knowledge for the interactive protocol for Graph isomorphism. To prove a single iteration of the interactive proof is perfect ZK, we need to perform the following steps:

- Construct a simulator $\mathcal{S}$ for every PPT V^*.
- Prove that the expected run time of $\mathcal{S}$ is polynomial.
- Prove that the output distribution of $\mathcal{S}$ is indistinguishable from the real execution.

The simulator is defined below,

```
$\mathcal{S}(x, z)$
$b' \leftarrow\$ \{0, 1\}, \sigma \leftarrow\$ \Pi_n$
$H = \sigma(G_{b'})$
Emulate execution of $V^*(x, z)$ by feeding it $H$. Let $b$ be its response.
If $b = b'$
    Feed $\sigma$ to $V^*$ and output its view.
Else
    Restart above procedure.
```

We briefly discuss the need to restart the procedure in the simulator. In the case that $b \neq b'$, the correct response would require the knowledge of π, which the simulator doesn't know (if it did, the adversary has all the 'knowledge' it needs). So the procedure is restarted with the hope that we will have $b = b'$ eventually.

The following lemma will aid us in performing the remaining two steps.

Lemma 24. *In the execution of $\mathcal{S}(x, z)$,*

- *H is identically distributed to $\sigma(G_0)$, and*
- $\Pr[b = b'] = \frac{1}{2}$

Proof Since G_0 is isomorphic to G_1, for a random $\sigma \leftarrow\$ \Pi_n$, the distributions $\sigma(G_0)$ and $\sigma(G_1)$ are identically distributed. Thus, H has a distribution that is independent of b'. Therefore, H has the same distribution as $\sigma(G_0)$.

The simulator chooses b' independently from x and z. When we emulate the execution of $V^*(x, z)$ on feeding H, x, z and H (as argued earlier) are independent of b'. Thus the output of $V*$ will be independent of b'. Since b' is chosen at random, $\Pr[b = b'] = \frac{1}{2}$. □

Run time: From the above lemma, we see that a single iteration of $\mathcal{S}$ has a success probability of $\frac{1}{2}$. Thus the expected number of iterations before $\mathcal{S}$ succeeds is 2. Since each iteration emulates a PPT adversary V^* in addition to some other polynomial time operations, it takes polynomial time. This in turn implies that the expected running time of $\mathcal{S}$ is polynomial.

Indistinguishability of Simulated View: The above lemma also shows that H has the same distribution as $\sigma(G_0)$. If we could always output σ, then the output distribution of $\mathcal{S}$ would match the distribution in the real execution. This is taken care of when we check if $b = b'$, and outputs H and σ only if it is true. But since H is independent of b', this does not change the output distribution. □

10.8 Reflections on Zero Knowledge

The proof of zero-knowledge property using a simulator may seem a little paradoxical for the following reasons:

- Protocol execution convinces V of the validity of x.
- But V could have generate the protocol transcript on his own.

To understand why there is no paradox, consider the following story:

> *Alice and Bob run the above protocol on input* (G_0, G_1) *where Alice acts as* P *and Bob as* V. *Now, Bob goes to Eve and tells her that* G_0 *and* G_1 *are isomorphic. Eve is skeptical about what Bob knows and asks how he knew this to be true. Bob then shows her the accepting transcript. But Eve knows all about simulators and doesn't believe Bob. She tells him that anyone could have come up with the transcript without actually knowing the isomorphism. Bob is now annoyed, and persists by telling her that he computed the transcript talking to Alice, who answered his every query correctly. But Eve remains unmoved.*

The two most important points of the above story are:

- Bob participated in a "live" conversation with Alice, and was convinced *how* the transcript was generated.
- Eve on the other hand did not see the live conversation, and has no way to tell if the transcript is from a real execution or produced by a simulator.

Thus, ZK is about transcripts while soundness is about "live" executions because of the random challenges.

10.9 Zero-knowledge Proofs for NP

We now prove a powerful theorem assuming the existence of one-way permutations. The theorem essentially states that anything that can be proved (and verified efficiently), can also be proved in ZK. The formal statement is,

Theorem 25. *If one-way permutations exist, then every language in* NP *has a ZK interactive proof.*

Remark 17. *The assumption of one-way permutations in the above theorem can be relaxed to only one-way functions.*

Now, let us consider how we could prove the above theorem. Could we achieve this by constructing a ZK proof for each language in NP? That would be ridiculously inefficient. Instead we focus on NP-complete languages, and rely on their 'completeness' property.

Proof The proof proceeds in two steps:

Step 1: Construct a ZK proof for an NP-complete language. We will consider *Graph 3-Coloring*, which is the language of all graphs whose vertices can be colored using only three colors such that no two connected vertices have the same color.

Step 2: To construct ZK proof for any NPlanguage L, do the following

- Given instance x and witness w, P and V reduce x into an instance x' of *Graph 3-Coloring* using Cook's deterministic reduction. The determinism ensures that both P and V end up with the same value x'.
- P also applies the reduction to the witness w to obtain witness w' for x'.
- Now, P and V can run the ZK proof from Step 1 on the common input x'.

We shall first show a "physical" proof. Here the colors are represented by numbers $\{1, 2, 3\}$. Let $\Pi_{\{1,2,3\}}$ denote the set of all permutations over $\{1, 2, 3\}$ and color_i refers to the 'color' of vertex $v_i \in V$ where $|V| = n$. We also define $\boxed{\text{color}_i}$ to be a locked box containing color_i. As with any locked box, it has a key key_i which locks and unlocks it. Obviously, it should be hard, if not impossible, to open or view the contents of this locked box without the key (we assume the box is opaque). The physical interactive proof follows below, Interactive proof for Graph three coloring

Repeat the following procedure $n|E|$ times using *fresh randomness*

$\underline{\mathsf{P}(G, (\mathsf{color}_1, \cdots, \mathsf{color}_n))}$ $\qquad\qquad$ $\underline{\mathsf{V}(G, z)}$

$\pi \leftarrow\$\, \Pi_{\{1,2,3\}}$

$\forall i \in [n], \widetilde{\mathsf{color}_i} = \pi(\mathsf{color}_i)$

$\forall i \in [n],\ \widetilde{\mathsf{color}_i} \xrightarrow[\mathsf{key}_i]{\mathsf{lock}} \boxed{\widetilde{\mathsf{color}_i}}$

$$\xrightarrow{\left(\boxed{\widetilde{\mathsf{color}_i}}, \cdots, \boxed{\widetilde{\mathsf{color}_i}}\right)}$$

$(u, v) \leftarrow\$\, E$

$$\xleftarrow{(u, v)}$$

$$\xrightarrow{\mathsf{key}_u, \mathsf{key}_v}$$

$\boxed{\widetilde{\mathsf{color}_u}} \xrightarrow[\mathsf{key}_u]{\mathsf{unlock}} \widetilde{\mathsf{color}_u}$

$\boxed{\widetilde{\mathsf{color}_v}} \xrightarrow[\mathsf{key}_v]{\mathsf{unlock}} \widetilde{\mathsf{color}_v}$

If $\widetilde{\mathsf{color}_u} \neq \widetilde{\mathsf{color}_v}$ Accept; else Reject

The completeness is trivial, and the intuitions for soundness follows from the fact that in each iteration, a cheating prover is caught with probability $\frac{1}{|E|}$ (we shall explain this later). For ZK, in each iteration, V only sees something it knew before - two random but different colors.

To “digitize" the above proof, we need some way to implement these locked boxes. Specifically, we need the two following properties about locked boxes:

- **Hiding:** V should not be able to see the contents inside a locked box.
- **Binding:** P should not be able to modify the content inside a box once it is locked.

Why do we even care about the second property? It’s something that’s so obvious about “physical” locked boxes that we often forget it exists. But this is what stops P from cheating when it doesn’t know a correct coloring. If this property wasn’t present, P on receiving (u, v) could modify the contents in the locked boxes containing the colorings of u and v to always unlock to different values. This would let P always convince V even without knowing the solution (coloring), thus violating the soundness requirement. $\square$

10.10 Commitment Schemes

The digital analogue of 'locked' boxes contains of two phases: A **Commit phase:** where the sender locks a value v inside a box. And a **reveal phase:** where the sender unlocks the box and reveals v. This can be implemented using interactive protocols, but we will consider the non-interactive case where both commit and reveal phases will consist of a single message. We call the digital analogue "commitment schemes" and formally define them,

Definition 58 (Commitment). *A randomized polynomial-time algorithm* Com *is called a* **commitment scheme** *for n-bit strings if it satisfies the following properties:*

- **Binding:** *For all $v_0, v_1 \in \{0,1\}^n$ and $r_0, r_1 \in \{0,1\}^n$ it holds that*

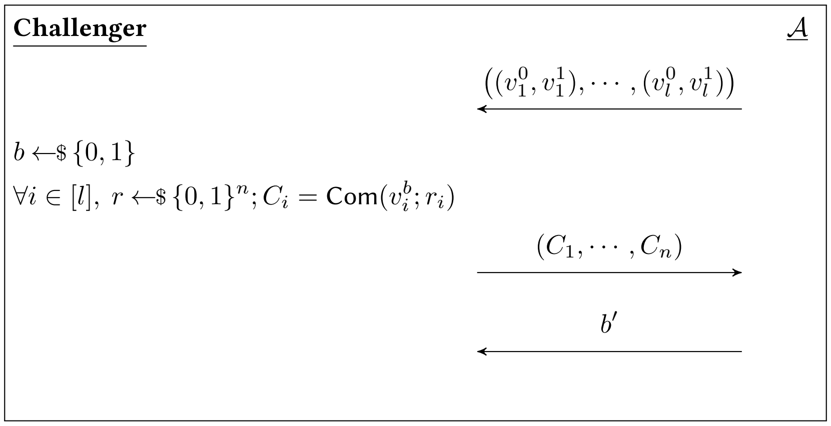
$$\mathsf{Com}(v_0; r_0) \neq \mathsf{Com}(v_1; r_1).$$

- **Hiding:** *For every non-uniform PPT distinguisher D, there exists a negligible function $\nu(\cdot)$ such that for every $v_0, v_1 \in \{0,1\}^n$, D distinguishes between the following distribution with probability at most $\nu(n)$*

$$\left\{r \leftarrow\$ \{0,1\}^n : \mathsf{Com}(v_0; r)\right\} \textit{ and } \left\{r \leftarrow\$ \{0,1\}^n : \mathsf{Com}(v_1; r)\right\}.$$

The above definition talks about *perfect binding* and *computational hiding*. Why don't we have *perfect binding* and *perfect hiding*? This is unfortunately impossible. It's left as an exercise to the reader to see why this is the case.

This definition only guarantees hiding for a single commitment. What about *multi-value hiding*? We sketch its definition here, similar to *multi-message secure* encryption schemes. Specifically, the adversary $\mathcal{A}$ needs to guess the bit b chosen by the challenger below.


Challenger
$\mathcal{A}$
$\big((v_1^0, v_1^1), \cdots, (v_l^0, v_l^1)\big)$
$b \leftarrow\$ \{0,1\}$
$\forall i \in [l],\ r \leftarrow\$ \{0,1\}^n; C_i = \mathsf{Com}(v_i^b; r_i)$
$(C_1, \cdots, C_n)$
b'

For security we require that the adversary $\mathcal{A}$ has only a negligible advantage in guessing b. We now claim that any commitment scheme satisfies *multi-value hiding*. Like public-key encryption, commitment schemes do not have any 'key', and we follow the same technique. The formal proof is left as an exercise. The corollary to this claim is that one-bit commitment implies string commitment.

Construction. The following theorem shows us how to construct a bit commitment scheme based on one-way permutations. Other such constructions based on pseudorandom generators, among others, exist too[**?**].

Theorem 26. *If one-way permutations exist, then commitment schemes exist.*

Proof Let f be a one-way permutation and h be the hard core predicate for f. We shall use these primitives to construct a bit-commitment scheme.

Commit phase: For this phase the sender computes $C = \mathsf{Com}(b; r) = (f(r), h(r) \oplus b)$. The bit b is being masked by $h(r)$.

Open phase: Sender reveals (b, r). Receiver accepts if $C = (f(r), h(r) \oplus b)$, and reject otherwise.

Binding follows from the fact that f is a permutation. Hence $f(r_1) = f(r_2) \iff r_1 = r_2$. The hard core bit is deterministic once r is fixed, thus ensuring binding.

For *hiding* we follow the proof for proving a secure single bit encryption scheme based on trapdoor permutation. For us, the trapdoor makes no difference to the binding or hiding properties and thus follows immediately. □

Now that we have our digital "locked boxes", we can proceed to our ZK proof for *Graph 3-coloring*.

10.11 Zero-knowledge Proof for Graph 3-coloring

The protocol for the interactive proof is presented below,

Interactive proof for Graph Isomorphism

Repeat the following procedure $n|E|$ times using *fresh randomness*

$\mathsf{P}(G, (\mathsf{color}_1, \cdots, \mathsf{color}_n))$ $\qquad\qquad$ $\mathsf{V}(G, z)$

$\pi \leftarrow\$\, \Pi_{\{1,2,3\}}$

$\forall i \in [n], \widetilde{\mathsf{color}}_i = \pi(\mathsf{color}_i)$

$\forall i \in [n],\ C_i = \mathsf{Com}(\widetilde{\mathsf{color}}_i;\ r_i)$

$$\xrightarrow{(C_1, \cdots, C_n)}$$

$(u, v) \leftarrow\$\, E$

$$\xleftarrow{(u, v)}$$

$$\xrightarrow{(\widetilde{\mathsf{color}}_u;\ r_u), (\widetilde{\mathsf{color}}_v;\ r_v)}$$

If C_u, C_v are valid
 and $\widetilde{\mathsf{color}}_u \neq \widetilde{\mathsf{color}}_v$ Accept
else Reject

The completeness trivially follows from the fact that knowledge of the right coloring ensures that the prover never fails. We need to prove the soundness and ZK for for the interactive proof.

Proof of Soundness:

Let G be the graph that is not 3-colorable. Then any coloring $\mathsf{color}_1, \cdots, \mathsf{color}_n$ will have at least one edge which have the same colors on both endpoints. Let one such edge be (i^*, j^*). From the binding property of Com, we know that $C_1, \cdots, C_n$ have unique openings $\widetilde{\mathsf{color}}_1 \cdots, \widetilde{\mathsf{color}}_n$. In each iteration P chooses $(u, v) = (i^*, j^*)$ with probability $\frac{1}{|E|}$. There might be other such edges too, but we take the pessimistic approach of assuming only one such edge. The cheating

P only succeeds if it is able to cheat in every iteration. Therefore, the probability P successfully cheats in every one of the $n|E|$ iterations is at most:

$$\left(1 - \frac{1}{|E|}\right)^{n|E|} \approx e^{-n}.$$

Proof of Zero Knowledge:

Intuition. In each iteration of the protocol, V only sees two random colors. The hiding property of Com guarantees that everything else remains hidden from V.

Simulator S(x=G,z):

1. Choose a random edge $(i', j') \xleftarrow{\$} E$ and pick random colors $color'_{i'}, color'_{j'} \xleftarrow{\$} \{1, 2, 3\}$ such that $color'_{i'} \neq color'_{j'}$. For every other $k \in [n] \backslash \{i', j'\}$, set $color'_k = 1$.
2. For every $\ell \in [n]$, computer $C_\ell = Com(color'_\ell)$
3. Emulate execution of $V^*(x, z)$ by feeding it $(C_1, ..., C_n)$. Let (i, j) denote its response.
4. If $(i, j) = (i', j')$, then feed the openings of C_i, C_j to V^* and output its view. Otherwise, restart the above procedure at most $n|E|$ times.
5. If simulation has not succeeded after $n|E|$ attempts, then output fail.

Correctness of Simulation. We will prove this using hybrid experiments.

H_0: Real execution

H_1 : Hybrid simulator S' that acts like the real prover using witness $(color_1, ..., color_n)$, except that it also chooses $(i', j') \xleftarrow{\$} E$ at random and if $(i', j') \neq (i, j)$ in all $n|E|$ trials , then it outputs fail.

H_2 : Simulator S

Now, we want to show that H_0 is indistinguishable from H_2. First, we will show that H_0 and H_1 are indistinguishable. If S' does not output fail, then H_0 and H_1 are identical. Since (i, j) and (i', j') are independently chosen, S' fails with probability at most $(1 - \frac{1}{|E|})^{n|E|} \approx e^{-n}$.

Now, we will show that H_1 and H_2 are indistinguishable. The only difference between H_1 and H_2 is that for all $k \in [n]\ \{i', j'\}, C_k$ is a commitment to $\pi(color_k)$ in H_1 and a commitment to 1 in H_2. Then, from the multi-value hiding property of Com, it follows that $H_1 \approx H_2$. □

Chapter 11

Secure Computation

11.1 Introduction

Consider two billionaires Alice and Bob with net worths x and y, respectively. They want to find out who is richer by computing the following function :

$$f(x,y) = \begin{cases} 1 & \text{if } x > y \\ 0 & \text{otherwise} Otherwise \end{cases}$$

One potential solution is that Alice sends x to Bob, who sends y to Alice. They each compute f on their own. However, this means that Alice learns Bob's net worth (and vice-versa). There is no privacy here. This presents the question: Can Alice and Bob compute f in a "secure manner" such that they only learn the output of f and nothing more?

To generalize this, consider two parties A and B, with private inputs x and y, respectively. They want to securely compute a function f over their inputs. If both A and B are honest, then they should learn the output $f(x,y)$. Further, even if one party is adversarial, it should not learn anything beyond the output (and its own input). How can we formalize this security requirement? Can we promise that the only thing that the parties learn is the output of the function (and their own respective inputs)?

11.2 Adversary Models

There are two types of adversaries:

Definition 59 (Honest but curious (aka semi-honest)). *Such an adversary follows the instructions of the protocol, but will later analyze the protocol transcript to* *learn any "extra information" about the input of the other party*

Definition 60 (Malicious). *Such an adversary can deviate from the protocol instructions and follow an arbitrary strategy*

In the context of secure computation, we will only consider semi-honest adversaries. There are generic transformations to amplify security against semi-honest adversaries to security against malicious adversaries.

11.3 Definition

Intuition. We want to formalize the concept that no semi-honest adversary learns anything from the protocol execution beyond its input and the (correct) output. The idea is to use simulation paradigm, as in zero-knowledge proofs. The view of adversary in the protocol execution can be efficiently simulated given only its input and output, and without the input of the honest party.

Definition 61 (Semi-Honest Secure Computation). *A protocol $\pi = (A, B)$ securely computes a function f in the semi-honest model if there exists a pair of non-uniform PPT simulator algorithms $\mathbb{S}_A, \mathbb{S}_B$ such that for every security parameter n, and all inputs $x, y \in \{0,1\}^n$, it holds that:*

$$\{\mathbb{S}_A(x, f(x,y)), f(x,y)\} \approx \{e \leftarrow [A(x) \leftrightarrow B(y)] : View_A(e), OUT_B(e)\}$$

$$\{\mathbb{S}_B(y, f(x,y)), f(x,y)\} \approx \{e \leftarrow [A(x) \leftrightarrow B(y)] : View_B(e), OUT_{A^\bullet}(e)\}$$

We talk about two simulators because we want privacy for both sides. In the case of zero knowledge, we didn't need any privacy against a cheating prover since the verifier has no private input.

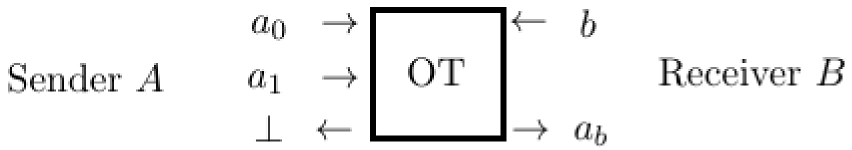
Remarks on Definition. Recall that in zero-knowledge, we only require indistinguishability of simulated view and adversary's view in the real execution. Here, indistinguishability is with respect to the joint distribution over the adversary's view and the honest party's output.

This is necessary for correctness. It implies that the output of the honest party in the protocol execution must be indistinguishable from the correct output $f(x, y)$. This guarantees that when the honest party talks to a corrupted party, the output is still always correct. If we remove this requirement, then a clearly wrong protocol where parties are instructed to output y would be trivially secure! We not only want security, we also want correctness of the output of the honest party.

11.4 Oblivious Transfer

Consider the following functionality, called, 1-out-of-two oblivious transfer (OT). There are two parties, a sender A and receiver B. A's input is a pair of bits (a_0, a_1), and B's input is a bit b. B's output is a_b and A receives no output.

This is bit OT because the inputs of sender are bits, but it can be easily generalized to string OT.



We define security for OT using Definition 61. Note that the definition promises that in a secure OT protocol, A does not learn b and B does not learn a_{1-b}.

11.5 Importance of Oblivious Transfer

OT can be realized from physical channels, as shown by Wiener and Rabin. In particular, a noisy channel can be used to implement OT. Furthermore, OT is *complete* for secure computation. This means that given a secure protocol for OT, any function can be securely computed. Finally, OT is also *necessary* for secure computation.

11.5.1 Construction

Let $\{f_i\}_{i \in I}$ be a family of trapdoor permutations with sampling algorithm Gen. Let h be a hardcore predicate for f_i.

Alice(a_0, a_1)		**Bob**(b)
$(f_i, f_i^{-1}) \leftarrow \mathsf{Gen}(1^n)$		
	$\xrightarrow{f_i}$	
		$x \leftarrow\$ \{0,1\}^n$
		$y_b \coloneqq f_i(x)$
		$y_{1-b} \leftarrow\$ \{0,1\}^n$
	$\xleftarrow{y_0, y_1}$	
$z_0 \coloneqq h_i(f_i^{-1}(y_0)) \oplus a_0$		
$z_1 \coloneqq h_i(f_i^{-1}(y_1)) \oplus a_1$		
	$\xrightarrow{z_0, z_1}$	
		Output $a_b = z_b \oplus h(x)$

11.6 Proof of Security

Intuition. Let's first consider security against adversarial A. Note that both y_0 and y_1 are uniformly distributed and therefore independent of b. Thus, b is hidden from A.

Next, let's consider security against adversarial B. We want to argue that a_{1-b} is hidden. We will use the fact that B does not know the pre-image of y_{1-b}. By the security of hardcore predicates, we know that $h_i(f_i^{-1}(y_{1-b})$ will look random to B. If B could learn a_{1-b}, then it would be able to predict the hardcore predicate. In particular, $h_i(f_i^{-1}(y_{1-b}))$ appears to be random for B.

Remark 18 (Malicious Receiver). *This protocol is not secure against a malicious receiver. Indeed, a malicious B can easily learn a_{1-b} by deviating from the protocol strategy. B can choose y_{1-b} by first choosing a pre-image x' and then computing $h(x')$.*

To formally prove semi-honest security, we will construct simulators for both cases.

Simulator $\mathbb{S}_A((a_0, a_1), \perp)$**:**

1. Fix a random tape r_A for A. Run honest emulation of A using (a_0, a_1) and r_A to obtain the first message f_i.

2. Choose two random strings $y_0, y_1 \in \{0,1\}^n$ as B's message
3. Run honest emulation of A using (y_0, y_1) to obtain the third message (z_0, z_1).
4. Stop and output $\perp$

Lemma 27. *The following two distributions are identical:*

$$\{\mathbb{S}_A((a_0, a_1), \perp), a_b\}, \text{ and}$$

$$\{e \leftarrow [A(a_0, a_1) \leftrightarrow B(b)] : View_A(e), Out_B(e)\}.$$

Proof The only difference between $\mathbb{S}_A$ and real execution is in step 2. However, since f is a permutation, y_0, y_1 are identically distributed in both cases. In the real case, y_b is computed by first sampling a random preimage and then computing f_i over it whereas in the simulation, it is sampled at random. Both of these have the same distribution. □

Simulator $\mathbb{S}_B(b, a_b)$**:**

1. Sample f_i, f_i^{-1}.
2. Choose random tape r_B for B. Run honest emulation of B using $b, r_B, f_i)$ to produce (x, y_0, y_1) s.t. $y_b = f_i(x)$ and $y_{1-b} \leftarrow \{0,1\}^n$.
3. Compute $z_b = h(x) \oplus a_b$ and $z_{1-b} \leftarrow \{0,1\}$
4. Output (z_0, z_1) as third message and stop

Lemma 28. *The following two distributions are indistinguishable:*

$$\{\mathbb{S}_B(b, a_b), \perp\}, \text{ and}$$

$$\{e \leftarrow [A(a_0, a_1) \longleftrightarrow B(b)] : View_B(e), OUT_A(e)\}.$$

Proof The only difference is in step 3, where $\mathbb{S}_B$ computes z_{1-b} as a random bit. However, since $h(f_i^{-1}(y_{1-b}))$ is indistinguishable from random (even given y_{1-b}), this change is indistinguishable □

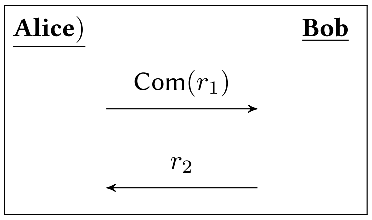
11.7 Remarks

Extension to 1-out-of-k OT. The previous protocol can be easily generalized to construct 1-out-of-k OT for $k > 2$.

Security against Malicious Adversaries. In reality, adversary may be malicious and not semi-honest. Goldreich-Micali-Widgerson [GMW] gave a compiler to transform any protocol that is secure against semi-honest adversaries into one secure against malicious adversaries. This is a general compiler and can be applied to any protocol. Below, we briefly discuss the main ideas in their compiler.

Once we have security against semi-honest adversaries, there are two main additional things that we have to worry about against a malicious adversary.

- First, a malicious adversary may choose a random tape so that the tape is biased. In this case, the protocol may not remain secure. The transformation of GMW uses coin-flipping to make sure that adversary's random tape is truly random. A simple coin-flipping protocol is described below:



  The random tape of P_2 is $r_1 \oplus r_2$, which is indistinguishable from random, and also hidden from P_1. Similarly, we can do coin-tossing to fix the random tape of P_1.
- A malicious adversary may also deviate from the protocol. In this case, all bets are off. For example, in the OT protocol, if B computed y_{1-b} from a pre-image, then it could learn a_{1-b}.
  One way to ensure that the parties do not deviate from the protocol instructions is to require them to attach a proof with every message to establish that it is following the protocol instructions. This way, each party can verify that the other party was acting semi-honestly. Furthermore, this proof must be zero knowledge or else it might reveal the inputs.

Securely Computing any Function Extending this to any functionality. The main question is how to enable the parties to compute any arbitrary function on their private inputs. 2 possible solutions for this are:

- Goldreich- Micali-Wigderson (GMW): This is a highly interactive solution. It extends naturally to multiparty case.
- Yao's Garbled Circuits: This requires relatively less interaction, but is tailored to only 2-party case.

11.8 Goldreich- Micali-Wigderson (GMW) Protocol

11.8.1 Circuit Representation

The Function $f(x, y)$ can be written as a boolean circuit C:

- *Input:* Input wires of C correspond to inputs x and y to f
- *Gates:* C contains AND and NOT gates, where each gate has fan in at most 2 and arbitrary fan out.
- *Output:* Output wires of C correspond to output of $f(x, y)$.

11.8.2 Secret Sharing

A k-out-of-n secret sharing scheme allows for "dividing“ a secret value s into n parts $s_1, ..., s_n$ s.t.

- **Correctness:** Any subset of k shares can be "combined“ to reconstruct the secret s.
- **Privacy:** The value s is completely hidden from anyone who only has at most $k - 1$ shares of s

Definition 62 (Secret Sharing). *A (k, n) secret-sharing consists of a pair of PPT algorithms (*$\mathsf{Share}, \mathsf{Reconstruction}$*) s.t.:*

- $\mathsf{Share}(s)$ *produces an n tuple $(s_1, ..., s_n)$*
- $\mathsf{Reconstruct}(s'_{i_1}, ..., s'_{i_k})$ *is s.t. if $\{s'_{i_1}, ..., s'_{i_k}\} \subseteq \{s_1, ..., s_n\}$, then it outputs s*
- *For any two s and $\tilde{s}$, and for any subset of at most $k - 1$ indices $X \subset [1, n]$, $|X| < k$, the following two distributions are statistically close:*

$$\{(s_1, ..., s_n) \leftarrow \mathsf{Share}(s) \ : \ (s_i | i \in X)\},$$

$$\{(\tilde{s}_1, ..., \tilde{s}_n) \leftarrow \mathsf{Share}(\tilde{s}) \ : \ (\tilde{s}_i | i \in X)\}.$$

Construction

1. An (n, n) secret-sharing scheme for $s \in \{0, 1\}$ based on XOR:
   - $\mathsf{Share}(s)$: Sample random bits $(s_1, ..., s_n)$ s.t. $s_1 \oplus ... \oplus s_n = s$
   - $\mathsf{Reconstruct}(s'_1, ..., s'_n)$: Output $s'_1 \oplus ... \oplus s'_n$
2. Shamir’s secret-sharing scheme: A (k, n) secret-sharing using polynomials

11.8.3 Protocol Notations

- **Protocol Ingredients:** A (2,2) secret-sharing scheme (Share, $\mathsf{Reconstruct}$), and a 1-out-of-4 OT scheme (OT=(S, R))
- **Common Input:** Circuit C for function $f(.,.)$ with two n-bit inputs and an n-bit output.
- **A's input:** $x = x_1, ..., x_n$ where $x_i \in \{0, 1\}$
- **B's input:** $y = y_1, ..., y_n$ where $y_i \in \{0, 1\}$

Protocol Invariant: For every wire in $C(x, y)$ with value $w \in \{0, 1\}$, A and B have shares w^A and w^B, respectively, s.t. $\mathsf{Reconstruct}(w^A, w^B) = w$

11.8.4 Protocol Details

Protocol $\Pi = (A, B)$**:** The GMW protocol consists of three phases:

1. ***Input Sharing:*** Each party secret-shares its input into two parts and sends one part to the other party.
   - A computes $(x_i^A, x_i^B) \leftarrow \mathsf{Share}(x_i)$ for every $i \in [n]$ and sends $(x_1^B, ...x_n^B)$ to B. B acts analogously.
2. ***Circuit Evaluation:*** The parties evaluate the circuit in a gate-by-gate fashion in such a manner that for every internal wire w in the circuit, each party holds a secret share of the value of wire w.
   - Run the CircuitEval sub-protocol. A obtains out_i^A and B obtains out_i^B for every output wire i.
3. ***Output Phase:*** Finally, the parties exchange the secret shares of the output wires. Each party then, on its own, combines the secret shares to compute the output of the circuit.
   - For every output wire i, A sends out_i^A to B, and B sends out out_i^B to A. Each party computes
   $$\mathsf{out}_i = \mathsf{Reconstruct}(\mathsf{out}_i^A, \mathsf{out}_i^B) \tag{11.1}$$

   The output is $\mathsf{out} = \mathsf{out}_1, ..., \mathsf{out}_n$

11.8.5 CircuitEval

NOT Gate: Input u, output w

- A holds u^A, B holds u^B

- A computes $w^A = u^A \oplus 1$
- B computes $w^B = u^B$

The shares that the two parties A and B now have, are shares of $\bar{u}$ ($w^A \oplus w^B = u^A \oplus 1 \oplus u^B = \bar{u}$)

AND Gate: Inputs u, v, output w

- A holds u^A, v^A, B holds u^B
- A samples $w^A \xleftarrow{\$} \{0,1\}$ and computes $w_1^B, ..., w_4^B$ as follows:

u^B	v^B	w^B
0	0	$w_1^B = w^A \oplus ((u^A \oplus 0).(v^A \oplus 0))$
0	1	$w_1^B = w^A \oplus ((u^A \oplus 0).(v^A \oplus 1))$
1	0	$w_1^B = w^A \oplus ((u^A \oplus 1).(v^A \oplus 0))$
1	1	$w_1^B = w^A \oplus ((u^A \oplus 1).(v^A \oplus 1))$

- A and B run $\mathsf{OT} = (S, R)$ where A acts as sender S with inputs $(w_1^B, ..., w_4^B)$ and B acts as receiver R with input $b = 1 + 2u^B + v^B$

11.8.6 Security

For every wire in C (except the input and output wires), each party only holds a secret share of the wire value. Security of NOT gate follows from construction. The security of AND gate follows from the security of OT. Simulator for Π can be constructed using the Simulator for OT to prove indistinguishability.

11.9 Yao's Garbled Circuits

Definition 63. *A Garbling Scheme consists of two procedures, $Garble$ and $Eval$:*

- *Garble(C): Takes a circuit C as input and will output a collection of garbled gates $\widehat{G}$ and garbled input wires $\widehat{In}$ where*

$$\widehat{G} = \{\widehat{g}_1, \dots, \widehat{g}_{|c|}\}$$

$$\widehat{In} = \{\widehat{in}_1, \dots, \widehat{in}_n\}$$

- *Eval($\widehat{G}$, $\widehat{In}_x$): Takes as input a garbled circuit $\widehat{G}$ and garbled input wires $\widehat{In}$ corresponding to an input x and outputs $z = C(x)$*

Now we will outline how Garbling Schemes work.

- Each wire i in the circuit C is associated with two keys (k_0^i, k_1^i) of a secret-key encryption scheme, one corresponding to the wire value being 0 and other for wire value being 1
- For an input x, the evaluator is given the input wire keys $(k_{x_1}^1, \dots, k_{x_n}^n)$ corresponding to x. Also for every gate $g \in C$, it is also given an encrypted truth table of g, which is something we will show later.
- We want the evaluator to use the input wire keys and the encrypted truth tables to uncover a single key k_v^i for every internal wire i corresponding to the value v of that wire. However, k_{1-v}^i should remain hidden from the evaluator.

In order to implement this we will have to define a special encryption scheme.

Definition 64. Special Encryption Scheme : *We need a secret-key encryption scheme* (Gen, Enc, Dec) *with an extra property: there exists a negligible function* $\nu(\cdot)$ *s.t. for every* n *and every message* $m \in \{0,1\}^n$*,*

$$Pr[k \leftarrow Gen(1^n), k' \leftarrow Gen(1^n), Dec_k(Enc_k(m)) = \bot] < 1 - \nu(n)$$

Essentially this is saying if a ciphertext is decrypted using a different or "wrong" key, then answer is always $\bot$

Construction : In order to create this special secret encryption simply modify the secret-key encryption scheme discussion in the secret lecture, except instead of encryption m, we encrypt $0^n||m$. Upon decrypting we check if the first n bits of the message are all 0's; if they aren't we output $\bot$

11.9.1 Garbled Circuits Construction

We are now going to define Garble and Eval for our Garbled Circuit. Let (Gen, Enc, Dec) be a special encryption scheme (as defined above). Assign an index to each wire in C s.t. the input wires have indices $1, \dots, n$.

Garble(C):

- For every non-output wire i in C, sample $k_0^i \leftarrow Gen(1^n)$, $k_1^i \leftarrow Gen(1^n)$. For every output wire i in C, set $k_0^i = 0$, $k_1^i = 1$.
- For every $i \in [n]$, set $\widehat{in}_i = (k_0^i, k_1^i)$. Set $\widehat{In} = (\widehat{in}_1, \dots, \widehat{in}_n)$
- For every gate g in C with input wire (i

First Input	Second Input	Output
k_0^i	k_0^j	$z_1 = Enc_{k_0^i}(Enc_{k_0^j}(k^l_{g(0,0)}))$
k_0^i	k_1^j	$z_2 = Enc_{k_0^i}(Enc_{k_1^j}(k^l_{g(0,1)}))$
k_1^i	k_0^j	$z_3 = Enc_{k_1^i}(Enc_{k_0^j}(k^l_{g(1,0)}))$
k_1^i	k_1^j	$z_4 = Enc_{k_1^i}(Enc_{k_1^j}(k^l_{g(1,1)}))$

Set $\widehat{g}$ = RandomShuffle(z_1, z_2, z_3, z_4). Output ($\widehat{G}$ = $(\widehat{g}_1, \dots, \widehat{g}_{|C|}), \widehat{In})$

Why is the random shuffle necessary? If we do not randomly shuffle the outputs an Adversary would know information about the combination of k_i, k_j used to achieve the output just based on the index of the returned value.

Eval($\widehat{G}, \widehat{In}_x$):

- Parse $\widehat{G} = (\widehat{g}_1, \dots, \widehat{g}_{|C|})$. $\widehat{In}_x = (k^1, \dots, k^n)$
- Parse $\widehat{g}_i = (\widehat{g}_1, \dots, \widehat{g}_4)$
- Decrypt each garbled gate $\widehat{g}_i$ one-by-one in canonical order:
  - Let k^i and k^j be the input wire keys for gate g.
  - Repeat the following for every $p \in [4]$:

$$\alpha_p = Dec_{k^i}(Dec_{k^j}(\widehat{g}_i^p))$$

  if $\exists \alpha_p \neq \bot$, set k^l = α_p
- Let out_i be the value obtained for each output wire i. Output $out = (out_1, \dots, out_n)$

11.9.2 Secure Computation from Garbled Circuits

Let us discuss a plausible approach for securely computing $C(x, y)$ using Garbled Circuits.

A generates a garbled circuit for $C(\cdot, \cdot)$ along with garbled wire keys for first and second input to C. It then sends the garbled wire keys corresponding to its input x along with the garbled circuit to B. Note, however, that in order to evaluate the garbled circuit on (x, y),B also needs the garbled wire keys corresponding to its input y.

A possible solution is for A to send all the wire keys corresponding to the second input of C to B. At first, this may seem to be a good idea. However this would mean B can not only compute $C(x, y)$ but also $C(x, y')$ for any y' of its choice. This is clearly an insecure solution!

To solve this problem A will transmit the garbled wire keys corresponding to B's input by using oblivious transfer. Below, we describe the solution in detail.

Ingredients: Garbling Scheme (Garble, Eval), 1-out-of-2 OT scheme OT = (S, R) as defined in previous lecture on secure computation.

Common Input: Circuit C for $f(\cdot,\cdot)$

A**'s input:** $x = x_1, \dots, x_n$

B**'s input:** $y = y_1, \dots, y_n$

Protocol $Pi = (A, B)$**:**

$A \to B$	A computes $(\widehat{G}, \widehat{In})$ Parse $\widehat{In} = (\widehat{in}_1, \dots, \widehat{in}_{2n})$ where $\widehat{in}_i = (k_0^i, k_1^i)$. Set $\widehat{In}_x = (k_{x_1}, \dots, k_{x_n}^n)$. Send $(\widehat{G}, \widehat{In}_x)$ to B
$A \leftrightarrow B$	For every $i \in [n]$, A and B run OT = (S, R) where A plays sender S with input (k_0^{n+i}, k_1^{n+i}) and B plays reciever R with in put y_i. Let $\widehat{In}_y = (k_{y_1}^{n+1}, \dots, k_{y_n}^{2n})$ be the outputs of the n OT executions received by B.
B	B outputs Eval$(\widehat{G}, \widehat{In}_x, \widehat{In}_y)$

In order to argue the security of the construction, we use two properties.

Property 1: For every wire i, B only learns one of the two wire keys:

- Input Wires: For input wires corresponding to $A's$ input, it follows from protocol description. For input wires corresponding to B's input it follows from security of OT
- Internal Wires: Follows from the security of the encryption scheme

Property 2: B does not know whether the key corresponds to wire value being 0 or 1 (except the keys corresponding to its own input wires).

From this we can notice that B only learns the output and nothing else. A does not learn anything (in particular, B's input remains hidden from A due to the security of OT). The full proof of security can be found in [**?**].

Chapter 12

Non-Interactive Zero Knowledge

12.1 Introduction

So far we have discussed the case of interactive Zero-Knowledge proofs. But what if Alice has the resource to send only a single message to Bob? This proof will now become "non-interactive". But 1-message Zero-Knowledge is only possible for languages in **BPP**. This is because any simulator that can simulate the "single" message can use this as a witness for x. But this is pretty useless, at the very least we want to be able to prove statements for languages in **NP**.

Fortunately, our savior is a "random string in the sky". This means that both Alice and Bob have access to a *common random string* that was honestly generated by someone they both trust. This string is something beyond the influence of either participant. While this is a departure from the model we have been considering, how can we hope to prove statements non-interactively using the common random string?

Let us start by formally defining non-interactive proofs,

12.2 Non-Interactive Proofs

Definition 65. *A non-interactive proof system for a language L with witness relation R is a tuple of algorithms $(\mathsf{K}, \mathsf{P}, \mathsf{V})$ such that:*

1. **Setup:** *$\sigma \leftarrow \mathsf{K}(1^n)$ outputs a common random string.*
2. **Prove:** *$\pi \leftarrow \mathsf{P}(\sigma, x, w)$ takes as input a common random string σ and a statement $x \in L$ and a witness w and outputs a proof π.*

3. **Verify:** $\mathsf{V}(\sigma, x, \pi)$ *outputs 1 if it accepts the proof and 0 otherwise.*

A non-interactive proof system must satisfy completeness and soundness properties given below.

Completeness: $\forall x \in L, \forall w \in R(x)$:

$$\mathsf{Pr}[\sigma \leftarrow \mathsf{K}(1^n);\ \pi \leftarrow \mathsf{P}(\sigma, x, w) : \mathsf{V}(\sigma, x, \pi) = 1] = 1$$

Non-Adaptive Soundness: There exists a negligible function $\nu(\cdot)$ such that $\forall x \notin L$

$$\mathsf{Pr}[\sigma \leftarrow \mathsf{K}(1^n);\ \exists\, \pi \text{ s.t. } \mathsf{V}(\sigma, x, \pi) = 1] \leq \nu(n)$$

Adaptive Soundness: There exists a negligible function $\nu(\cdot)$ such that

$$\mathsf{Pr}[\sigma \leftarrow \mathsf{K}(1^n);\ \exists\, (x, \pi) \text{ s.t. } x \notin L \wedge \mathsf{V}(\sigma, x, \pi) = 1] \leq \nu(n)$$

The reader should note, in non-adaptive soundness, the adversary chooses x before seeing the common random string whereas in adaptive soundness, it can choose x depending upon the common random string. Adaptive soundness is a stronger notion of soundness.

Similar to soundness, we will define two variants of Non-interactive Zero Knowledge (NIZK). Before we proceed, we note that we can transform any NIZK proof system with non-adaptive soundness into one that achieves adaptive soundness in a manner similar to the hardness amplification done earlier in the course. The details can be found in [**?**].

Since each statement can have multiple witnesses, we define the following set for each $x \in L$:

$$R(x) = \{w \mid R(x, w) = 1\}$$

where R is the relation for the language L.

Definition 66 (Non-Adaptive NIZK). *A non-interactive proof system* $(\mathsf{K}, \mathsf{P}, \mathsf{V})$ *for a language L with witness relation R is non-adaptive Zero-Knowledge if there exists a PPT simulator $\mathcal{S}$ s.t. for every $x \in L, w \in R(x)$, the output distribution of the following two experiments are computationally indistinguishable:*

$\mathsf{REAL}(1^n, x, w)$	$\mathsf{IDEAL}(1^n, x)$
$\sigma \leftarrow \mathsf{K}(1^n)$	$(\sigma, \pi) \leftarrow \mathcal{S}(1^n, x)$
$\pi \leftarrow \mathsf{P}(\sigma, x, w)$	
Output (σ, π)	*Output* (σ, π)

Here the simulator generates both the common random string and the simulated proof given the statement x as input. The simulated common random string can depend on x and thus can be used only for a single proof.

We proceed to define the adaptive variant below.

Definition 67 (Adaptive NIZK). *A non-interactive proof system* $(\mathsf{K}, \mathsf{P}, \mathsf{V})$ *for a language* L *with witness relation* R *is adaptive Zero-Knowledge if there exists a PPT simulator* $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ *s.t. for every* $x \in L, w \in R(x)$*, the output distribution of the following two experiments are computationally indistinguishable:*

$\mathsf{REAL}(1^n, x, w)$	$\mathsf{IDEAL}(1^n, x)$
$\sigma \leftarrow \mathsf{K}(1^n)$	$(\sigma, \tau) \leftarrow \mathcal{S}_0(1^n)$
$\pi \leftarrow \mathsf{P}(\sigma, x, w)$	$\pi \leftarrow \mathcal{S}_1(\sigma, \tau, x)$
Output (σ, π)	*Output* (σ, π)

Here τ is the "trapdoor" for the simulated common random string σ that is used by $\mathcal{S}_1$ to generate an accepting proof for x without knowing the witness. This definition also captures the definition of *reusable* common random strings.

Recall, unlike the definition of (interactive) Zero-Knowledge, we don't define the Zero-Knowledge property over "all non-uniform PPT adversary V^*". We leave it to the reader to see why it is the case.

We make a few remarks about the NIZK definition before we proceed further,

- In NIZK, the simulator is given the seemingly extra power to choose the common random string along with a possible trapdoor that allows for simulation without a witness.
- In the interactive case, we gave the simulator the extra power to "reset" the verifier. Is this extra power inherent?
- It turns out that a simulator must always have some extra power over the normal prover, otherwise, the definition would be impossible to realize for languages outside **BPP**.
- We justify the extra power since we require indistinguishability of the joint-distribution over the common random string and the proof.

Lemma 29. *There exists an efficient transformation from any non-interactive proof system* $\mathsf{K}, \mathsf{P}, \mathsf{V}$ *with non-adaptive soundness into a non-interactive proof system* $\mathsf{K}', \mathsf{P}', \mathsf{V}'$ *with adaptive soundness.*

12.3 NIZKs for NP

I. Non-adaptive Zero Knowledge: We first construct NIZKs for **NP** with non-adaptive Zero-Knowledge property using the following two steps:

1. Construct a NIZK proof system for **NP** in the **hidden-bit model**. This step is unconditional.
2. Using trapdoor permutation, transform any NIZK proof system for language in the hidden-bit model to a non-adaptive NIZK proof system in the common random string model.

In today's class we shall define NIZKs in the hidden-bit model, and show a transformation from NIZKs in hidden-bit model to NIZKs in the common-random string model. In the next class we shall build NIZKs for **NP** in the hidden-bit model.

II. Adaptive Zero Knowledge: Next, we transform non-adaptive NIZKs for **NP** into adaptive NIZKs for **NP**. This step only requires one-way functions, which are implied by trapdoor permutations. This will be a part of the homework.

Putting all the steps together, we obtain adaptive NIZKs for **NP** based on trapdoor permutations.

12.4 The Hidden-Bit Model

In this section we shall describe the hidden-bit model and define NIZK in the hidden-bit model. It is important to note that this model provides a step towards our ultimate goal of building NIZKs for **NP**, and is not meant to be realistic.

In this model, the prover is given some sequence of bits that are hidden from the verifier. To prove some statement $x \in L$, the prover may choose to reveal some of some of these bits to the verifier. The remaining bits remain hidden from the verifier. Also, the prover cannot tamper these bits before revealing them to the verifier.

Definition 68. *A non-interactive proof system for a language L with witness relation R in the hidden-bit model is a tuple of algorithms $(\mathsf{K_{HB}}, \mathsf{P_{HB}}, \mathsf{V_{HB}})$ such that:*

1. **Setup:** $r \leftarrow \mathsf{K}(1^n)$ *outputs a common random string.*
2. **Prove:** $\pi \leftarrow \mathsf{P_{HB}}(r, x, w)$ *generates the indices $I \subseteq [|r|]$ of r to reveal along with a proof π.*
3. **Verify:** $\mathsf{V_{HB}}(I, \{r_i\}_{i \in I}, x, \pi)$ *outputs 1 if it accepts the proof and 0 otherwise.*

We define Zero-Knowledge below. A non-interactive proof system must satisfy completeness and soundness properties defined earlier.

Definition 69 (NIZK in Hidden-Bit Model). *A non-interactive proof system $(\mathsf{K_{HB}}, \mathsf{P_{HB}}, \mathsf{V_{HB}})$ for a language L with witness relation R in the hidden-bit model* *is (non-adaptive)Zero-Knowledge if there exists a PPT simulator $\mathcal{S}_{\mathsf{HB}}$ s.t. for every $x \in L, w \in R(x)$, the output distribution of the following two experiments are computationally indistinguishable:*

$\mathsf{REAL}(1^n, x, w)$	$\mathsf{IDEAL}(1^n, x)$
$r \leftarrow \mathsf{K}_{\mathsf{HB}}(1^n)$	$(I, \{r_i\}_{i\in I}, \pi) \leftarrow \mathcal{S}_{\mathsf{HB}}(1^n, x)$
$(I, \pi) \leftarrow \mathsf{P}_{\mathsf{HB}}(r, x, w)$	
Output $(I, \{r_i\}_{i\in I}, \pi)$	*Output* $(I, \{r_i\}_{i\in I}, \pi)$

12.5 From NIZK in HB model to NIZK in CRS model

We sketch our intuition for the construction. We need to transform a "public" random string into a "hidden" random string. If the prover samples a trapdoor permutation (f, f^{-1}) with hardcore predicate h. Given a common random string $\sigma = \sigma_1, \cdots, \sigma_n$, the prover can compute $r = r_1 \cdots, r_n$ where:

$$r_i = h(f^{-1}(\sigma_i)).$$

Since f is a permutation and h is a hard-core predicate, r is guaranteed to be random. Then we can treat r as the hidden random string, revealing only parts of it to V. We now proceed to the construction.

Construction. Let $\mathcal{F} = \{f, f^{-1}\}$ be a family of 2^n trapdoor permutations with hardcore predicate h. We assume that it is easy to test membership of $\mathcal{F}$. Let $(\mathsf{K}_{\mathsf{HB}}, \mathsf{P}_{\mathsf{HB}}, \mathsf{V}_{\mathsf{HB}})$ be a NIZK proof system for L in the hidden-bit model with soundness error $\frac{1}{2^{2n}}$. The procedures for K, P and V are given below.

```
K(1^n)
1 :  for every $i \in [n]$
2 :      $\sigma_i \leftarrow\$ \{0,1\}^n$
3 :  Output $\sigma = \sigma_1\sigma_2\cdots\sigma_n$
```

```
P(σ, x, w)
1 : $(f, f^{-1}) \leftarrow\$ \mathcal{F}(1^n)$
2 : for every $i \in [n]$
3 :     $\alpha_i = f^{-1}(\sigma_i)$
4 : for every $i \in [n]$
5 :     $r_i = h(\alpha_i)$
6 : $(I, \Phi) \leftarrow \mathsf{P}_{\mathsf{HB}}(r, x, w)$
7 : Output $\pi = (\sigma, f, I, \{\alpha_i\}_{i \in I}, \Phi)$
```

```
V(σ, x, π)
1 : $(\sigma, f, I, \{\alpha_i\}_{i \in I}, \Phi) \leftarrow \pi$
2 : Check $f \in \mathcal{F}$
3 : Check for every $i \in I$
4 :     $f(\alpha_i) = \sigma_i$
5 : for every $i \in I$
6 :     $r_i = h(\alpha_i)$
7 : Output $\mathsf{V}_{\mathsf{HB}}(I, \{\alpha_i\}_{i \in I}, x, \Phi)$
```

Theorem 30. *Given that* $(\mathsf{K}_{\mathsf{HB}}, \mathsf{P}_{\mathsf{HB}}, \mathsf{V}_{\mathsf{HB}})$ *is a NIZK proof system for* L *in the hidden-bit model with soundness error* $\frac{1}{2^{2n}}$*, then our construction* $(\mathsf{K}, \mathsf{P}, \mathsf{V})$ *above is a NIZK proof system for* L *in the CRS model.*

Proof We need to argue that each property of the NIZK proof system in the CRS model is satisfied.

Completeness

We sample each σ_i uniformly at random, and since f^{-1} is a permutation, α_i will be uniform random. For completeness, by definition, we assume that the prover is honest and thus picks f and f^{-1} correctly. Since h is a hardcore predicate, each r_i is also random. Since the sampling and computations are done independently, we get r to be uniformly distributed. Now this reduces to the *hidden-bit model.* Completeness follows from the completeness of (K_{HB}, P_{HB}, V_{HB})

Soundness

When we fix f to be f_0, r is uniformly distributed. Thus, from the (non-adaptive) soundness of $(\mathsf{K}_{\mathsf{HB}}, \mathsf{P}_{\mathsf{HB}}, \mathsf{V}_{\mathsf{HB}})$, we have

$$\Pr[\sigma \leftarrow K(1^n) : \mathsf{P}^* \text{ can cheat using } f_0] \leq \frac{1}{2^{2n}}$$

There are only 2^n possible choices for f, and the verifier checks if f is indeed from $\mathcal{F}$. By the union bound, we have

$$\mathsf{Pr}[\sigma \leftarrow K(1^n) : \mathsf{P}^* \text{ can cheat}] = \mathsf{Pr}[\sigma \leftarrow K(1^n) : \mathsf{P}^* \text{ can cheat for some } f] \leq \frac{1}{2^n}$$

Zero-Knowledge

Let $\mathcal{S}_{\mathsf{HB}}$ be the simulator for $(\mathsf{K}_{\mathsf{HB}}, \mathsf{P}_{\mathsf{HB}}, \mathsf{V}_{\mathsf{HB}})$. The simulator is,

```
$\mathcal{S}(1^n, x)$
1 : $(I, \{r_i\}_{i \in I}, \Phi) \leftarrow\$ \mathcal{S}_{\mathsf{HB}}(1^n, x)$
2 : $(f, f^{-1}) \leftarrow\$ \mathcal{F}$
3 : for every $i \in I$
4 :     $\alpha_i = h^{-1}(r_i)$
5 : for every $i \in I$
6 :     $\sigma_i = f(\alpha_i)$
7 : for every $i \notin I$
8 :     $\sigma_i \leftarrow\$ \{0,1\}^n$
9 : Output $(\sigma, f, I, \{\alpha_i\}_{i \in I}, \Phi)$
```

Here $h^{-1}(r_i)$ denotes sampling from the pre-image of r_i, which can be done efficiently by simply trying random α_i's until $h(\alpha_i) = r_i$. This method has low expected number of attempts as we are trying to match just one bit r_i. To prove Zero-Knowledge, we build a sequence of hybrids below. We argue the computational indistinguishability of the hybrids at the end. Changes from the previous hybrid are marked with a box.

```
$H_0(1^n, x, w) = \mathsf{REAL}(1^n, x, w)$
1 : $\sigma \leftarrow\$ K(1^n)$ where $\sigma = \sigma_1, \cdots, \sigma_n$
2 : $(f, f^{-1}) \leftarrow\$ \mathcal{F}$
3 : for every $i \in [n]$
4 :     $\alpha_i = f^{-1}(\sigma_i)$
5 : for every $i \in [n]$
6 :     $r_i = h(\alpha_i)$
7 : $(I, \Phi) \leftarrow \mathsf{P}_{\mathsf{HB}}(r, x, w)$
8 : Output $(\sigma, f, I, \{\alpha_i\}_{i \in I}, \Phi)$
```

```
$H_1(1^n, x, w)$
1 : $\boxed{\alpha_i \leftarrow\$ \{0,1\}^n \text{ for every } i \in [n]}$
2 : $(f, f^{-1}) \leftarrow\$ \mathcal{F}$
3 : for every $i \in [n]$
4 :     $\boxed{\sigma_i = f(\alpha_i)}$
5 : for every $i \in [n]$
6 :     $r_i = h(\alpha_i)$
7 : $(I, \Phi) \leftarrow \mathsf{P}_{\mathsf{HB}}(r, x, w)$
8 : Output $(\sigma, f, I, \{\alpha_i\}_{i \in I}, \Phi)$
```

```
$H_2(1^n, x, w)$
1 : $\boxed{r_i \leftarrow\$ \{0,1\}^n \text{ for every } i \in [n]}$
2 : $(f, f^{-1}) \leftarrow\$ \mathcal{F}$
3 : for every $i \in [n]$
4 :     $\boxed{\alpha_i = h^{-1}(r_i)}$
5 : for every $i \in [n]$
6 :     $\sigma_i = f(\alpha_i)$
7 : $(I, \Phi) \leftarrow \mathsf{P}_{\mathsf{HB}}(r, x, w)$
8 : Output $(\sigma, f, I, \{\alpha_i\}_{i \in I}, \Phi)$
```

```
$H_3(1^n, x, w)$
1 : $r_i \leftarrow\$ \{0,1\}^n$ for every $i \in [n]$
2 : $(f, f^{-1}) \leftarrow\$ \mathcal{F}$
3 : for every $i \in [n]$
4 :     $\alpha_i = h^{-1}(r_i)$
5 : $(I, \Phi) \leftarrow \mathsf{P}_{\mathsf{HB}}(r, x, w)$
6 : $\boxed{\text{for every } i \in I}$
7 :     $\boxed{\sigma_i = f(\alpha_i)}$
8 : $\boxed{\text{for every } i \notin I}$
9 :     $\boxed{\sigma_i \leftarrow\$ \{0,1\}^n}$
10 : Output $(\sigma, f, I, \{\alpha_i\}_{i \in I}, \Phi)$
```

```
$H_4(1^n, x) = \mathsf{IDEAL}(1^n, x)$
1 : $\boxed{(I, \{r_i\}_{i \in I}, \Phi) \leftarrow\$ \mathcal{S}_{\mathsf{HB}}(1^n, x)}$
2 : $(f, f^{-1}) \leftarrow\$ \mathcal{F}$
3 : $\boxed{\text{for every } i \in I}$
4 :     $\alpha_i = h^{-1}(r_i)$
5 : for every $i \in I$
6 :     $\sigma_i = f(\alpha_i)$
7 : for every $i \notin I$
8 :     $\sigma_i \leftarrow\$ \{0,1\}^n$
9 : Output $(\sigma, f, I, \{\alpha_i\}_{i \in I}, \Phi)$
```

$\mathbf{H_0} \approx \mathbf{H_1}$: In H_1, we sample α_i at random and then compute σ_i. This is in contrast to H_0 where we sample σ_i before computing α_i. Since f is a permutation, H_1 induces the same distribution as H_0.

$\mathbf{H_1} \approx \mathbf{H_2}$: In H_2, we first sample r_i before sampling α_i from the pre-image of r_i. This distribution is identical to H_1.

$\mathbf{H_2} \approx \mathbf{H_3}$: In H_3, we output a random σ_i for $i \notin I$. From the security of the hard-core predicate h, it follows that

$$\{f(h^{-1}(r_i))\} \approx_c U_n$$

Indistinguishability of H_2 and H_3 follows using the above equation.

$\mathbf{H_3} \approx \mathbf{H_4}$: In H_4, we swap $\mathbf{P_{HB}}$ with $\mathcal{S}_{\mathbf{HB}}$. Indistinguishability follows from the Zero-Knowledge property of $(\mathsf{K}_{\mathsf{HB}}, \mathsf{P}_{\mathsf{HB}}, \mathsf{V}_{\mathsf{HB}})$.

Thus $H_0 \approx H_4$. This gives us the Zero-Knowledge proof. □

Next we shall construct NIZKs for all languages in **NP** in the hidden-bit model.

12.6 Hamiltonian Graphs

A Hamiltonian graph is a graph that consists a Hamiltonian cycle. In other words, there exists a cycle formed by edges in the graph that visits each vertex exactly once. More formally:

Definition 70 (Hamiltonian Graph). *Let $G = (V, E)$ be a graph with $|V| = n$. We say that G is a Hamiltonian graph if it has a Hamiltonian cycle, i.e. there are $v_1, ..., v_n \in V$ such that for all $i \in [n]$:*

$$(v_i, v_{(i+1) \mod n}) \in E$$

Fact: Deciding whether a graph is Hamiltonian in **NP**-Complete. Let L_H be the language of Hamiltonian graphs $G = (V, E)$ s.t. $|V| = n$

Any graph can be represented as an adjacency matrix. The number of rows and columns of this matrix is the same as the number of vertices in the graph. A value of 1 at a given position represents the presence of an edge between the vertices corresponding to the row and column. More specifically:

Definition 71 (Adjacency Matrix). *A graph $G = (V, E)$ with $|V| = n$, can be represented as an $n \times n$ adjacency matrix M_G of boolean values such that:*

$$M[i, j] = \begin{cases} 1 & \textit{if } (i, j) \in E \\ 0 & \textit{otherwise} \end{cases}$$

Definition 72 (Cycle Matrix). *A cycle matrix is a boolean matrix that corresponds to a graph that contains a Hamiltonian cycle and no other edges.*

Definition 73 (Permutation Matrix). *A permutation matrix is a boolean matrix such that each row and each column has exactly one entry equal to 1.*

Note: Every cycle matrix is a permutation matrix, but the converse is not true. For every n, there are $n!$ permutation matrices, but only $(n - 1)!$ cycle matrices.

Note that a consequence of the above is that if we pick a permutation matrix at random, it is also a cycle matrix with probability $\frac{1}{n}$.

12.7 NIZKs for L_H in Hidden-Bit Model

We want to come up with NIZKs for the Hamiltonian graph problem in the HB model. We are going to do this in two steps:

Step I: NIZK (K_1, P_1, V_1) for L_H in hidden-bit model where K produces (hidden) strings r with a specific distribution: each r represents an $n \times n$ cycle matrix.

This first step is a simplified case. In the HB model we had a truly random string. The prover gets to see this random string, but the verifier only gets to see some part of this random string that is decided by the prover. What we are doing in the first step is considering a simplified model where we will allow the string to have a very particular distribution. It will not be truly random, but biased. In particular, we will consider NIZKs for the Hamiltonian graph problem where the algorithm will produce strings that will represent a $n \times n$ matrix.

Step II: Modify the above construction to obtain $K_2, P_2, V_2)$ where the (hidden) string r is uniformly random

Once we have the construction from the previous step, we will show how to extend this construction so that we can move to the real world where the string is supposed to be uniform. At high level, we will use a very large random string and the come up with some deterministic algorithm which will give us a way to convert such a long random string into a small random string which will have the desired distribution with very high probability.

12.7.1 Step I

Construction of $K_1, P_1, V_1)$ for L_H: We describe the algorithms below.

$K_1(1^n)$: Output $r \leftarrow \{0,1\}^{n^2}$ s.t. it represents an $n \times n$ cycle matrix M_C
In other words, K takes the security parameter and outputs a string r with a very particular distribution such that r represents a cycle matrix. This represents the input given to the prover.

$P_1(r, x, w)$: Execute the following steps:

- Parse $x = G = (V, E)$ s.t. $|V| = n$, and $w = H$ where $H = (v_1, ..., v_n)$ is a Hamiltonian cycle in G. To be more specific, instance X corresponds to a graph supposedly Hamiltonian, and w is some witness represented by a Hamiltonian cycle in the graph.

- Choose a permutation $\varphi : V \to \{1, ..., n\}$ that maps H to the cycle in M_C, i.e., for every $i \in [n]$:

$$M_C[\varphi(v_i), \varphi(v_{(i+1) \mod n})] = 1$$

- Define $I = \{\varphi(u), \varphi(v) | M_G[u, v] = 0\}$ to be the set of non-edges in G
- Output (I, φ)

To summarize, the prover first picked a permutation that maps the witness to the cycle in the random string r. It wants to be given some evidence that it actually has a cycle. It is going to show that all non-edges in $\varphi(G)$ are mapped to a 0 value. The verifier can accomplish this task. The idea is that if G does not actually have a cycle, then no matter what mapping we come up with, at least one non-edge in G will get mapped to an edge in the cycle graph and then the verifier will catch it. (M_G is the adjacency matrix of G).

$V_1(I, r_I, \varphi)$: Execute the following steps:

- Parse $r_I = \{M_C[u, v]\}_{(u,v) \in I}$
- Check that for every $(u, v) \in I$, $M_C[u, v] = 0$
- Check that for every $(u, v) \in I$, $M_G(\varphi^{-1}(u), \varphi^{-1}(v)) = 0$
- If both the checks succeed, then output 1 and 0 otherwise

Completeness: An honest prover P can always find a correct mapping φ that maps H to the cycle in M_C.

Soundness: If $G = (V, E)$ is not a Hamiltonian graph, then for any mapping $\varphi \to \{1, ..., n\}$, $\varphi(G)$ will not cover all the edges in M_C. There must exist at least one non-zero entry in M_C that is revealed as a non-edge of G.

Zero Knowledge: Simulator S performs the following steps:

- Sample a random permutation $\varphi : V \to \{1, ..., n\}$
- Compute $I = \{\varphi(u), \varphi(v) | M_G[u, v] = 0\}$
- For every $(a, b) \in I$, set $M_C[a, b] = 0$
- Output $(I, \{M_C[a, b]\}_{(a,b) \in I}, \varphi)$

It is easy to verify that the above output distribution is identical to the real experiment.

Note that here, the simulator can choose the set r_I, so it controls the string r seen by the verifier. Therefore, this simulator is non-adaptive. When we transform such a NIZK in HB model to a NIZK in the CRS model, this non-adaptivity property carries over which makes the CRS non-reusable.

12.7.2 Step II

We start by describing the strategy in this step:

- Define a deterministic procedure Q that takes as input a (polynomially long) random string r and outputs a biased string s that corresponds to a cycle matrix with inverse polynomial probability $\frac{1}{l(n)}$. We want to come up with a way to amplify this probability and make it closer to 1.
- If we feed $Qn \cdot l(n)$ random inputs, then with high probability, at least one of the outputs will correspond to a cycle matrix
- In the NIZK construction, the (hidden) random string will be $r = r_1, \ldots, r_{n \cdot l(n)}$
- For every i, the prover will try to compute a proof using $s_i = Q(r_i)$. In other words, it will take a chunk from the random string r_I, apply the deterministic procedure Q on it to obtain some string s_i, and now with this much probability s_i will be a cycle matrix
- At least one s_i will contain a cycle matrix, so we can use the NIZK proof system from Step 1

We now explain the deterministic procedure Q.

Procedure $Q(r)$:

- Parse $r = r_1, ..., r_{n^4}$ s.t. $\forall i, |r_i| = \lceil 3 \log n \rceil$
- Compute $s = s_1, ..., s_{n^4}$, where:

$$s_i = \begin{cases} 1 & \text{if } r_i = 111 \cdots 1 \\ 0 & \text{otherwise} \end{cases}$$

- Define an $n^2 \times n^2$ boolean matrix M consisting of entries from s
- If M contains an $n \times n$ sub-matrix M_C s.t. M_C is a cycle matrix, then output (M, M_C), else output $(M, \perp)$.

Analysis of Q: Let GOOD be the set of outputs of $Q(\cdot)$ that contain a cycle matrix and BAD be the complementary set.

Lemma 31. *For a random input r, $PR[Q(r) \in GOOD] \geq \frac{1}{3n^3}$*

Let M be an $n^2 \times n^2$ matrix computed by Q on a random input r. We will prove the above lemma via a sequence of claims:

Claim 1: M contains exactly n 1's with probability at least $\frac{1}{3n}$
Claim 2: M contains a permutation sub-matrix with probability at least $\frac{1}{3n^2}$
Claim 3: M contains a cycle sub-matrix with probability at least $\frac{1}{3n^3}$

Proof of Claim 1: Let X be the random variable denoting the number of 1's in M.

- X follows the binomial distribution with $N = n^4, p = \frac{1}{n^3}$
- $E(X) = N \cdot p = n$
- $Var(X) = Np(1-p) < n$
- Recall Chebyshev's Inequality: $Pr[|X - E(X)| > k] \leq \frac{Var(X)}{k^2}$
  Setting $k = n$, we have:

$$Pr[|X - n| > n] \leq \tfrac{1}{n}$$

- Observe:

$$\sum_{i=1}^{2n} Pr[X = i] = 1 - Pr[|X - n| > n] > 1 - \frac{1}{n}$$

- $Pr[X = i]$ is maximum at $i = n$
- Observe:

$$Pr[X = n] \geq \frac{\sum_{i=1}^{2n} Pr[X = i]}{2n}$$
$$\geq \tfrac{1}{3n}$$

Proof of Claim 2: We want to bound the probability that each of the n '1' entries in M is in a different row and column.

- After k '1' entries have been added to M,

$$Pr[\text{new '1' entry is in different row and column}] = \left(1 - \tfrac{k}{n^2}\right)^2$$

- Multiplying all:

$$Pr[\text{no collision}] \geq \left(1 - \tfrac{1}{n^2}\right)^2 \cdots \left(1 - \tfrac{n-1}{n^2}\right)^2$$

$$\geq \tfrac{1}{n}$$

– Combining the above with Claim 1:

$$Pr[\text{M contains a permutation } n \times n \text{ submatrix}] \geq \tfrac{1}{3n^2}$$

Proof of Claim 3 We want to bound the probability that M contains an $n \times n$ cycle sub-matrix

– Observe:

$$Pr[n \times n \text{ permutation matrix is a cycle matrix }] = \tfrac{1}{n}$$

– Combining the above with Claim 2,

$$Pr[M \text{ contains a cycle } n \times n \text{ submatrix } \geq \tfrac{1}{3n^3}$$

Construction of (K_2, P_2, V_2) for L_H: We now describe the algorithms

$K_2(1^n)$: Output $r \leftarrow \{0,1\}^L$ where $L = \lceil 3 \log n \rceil \cdot n^8$

$P_2(r, x, w)$: Parse $r = r_1, ..., r_{n^4}$ s.t. for every $i \in [n^4]$, $|r_i| = \lceil 3 \log n \rceil \cdot n^4$.
For every $i \in [n^4]$:

- If $Q(r_i) = (M^i, \perp)$,set $I_i = [|r_i|]$(i.e., reveal the entire r_i), and $\pi_i = \emptyset$
- Else, let $(M^i, M_C^i) \leftarrow Q(r_i)$. Compute$(I_i', \varphi_i) \leftarrow P_1(M_C^i, x, w)$. Set $I_i = I_i' \cup J_i$ where J_i is the set of indices s.t. r_i restricted to J_i yields the residual M^i after removing M_C^i, and $\pi_i = \varphi_i$

Output $(I = \{I_i\}, \pi = \{\pi_i\})$

$V_2(I, r_I, \pi)$: Parse $I = I_1, ..., I_{n^4}$, and $\pi = \pi_1, ..., \pi_{n^4}$. For every $i \in [n^4]$:

- If I_i is the complete set, then check that $Q(s_i) = (\cdot, \perp)$
- Otherwise, parse $I_i = I_i' \cup J_i$. Parse $s_i = s_i^1, s_i^2$ and check that s_i^2 is the all 0 string. Also, check that $V_1(I_i', s_i^1, \pi_i) = 1$.

If all the checks succeed, then output 1 and 0 otherwise.

Completeness: It follows from completeness of the construction in Step I.

Soundness: For random $r = r_1, ..., r_{n^4}, Q(r_i) \in$ GOOD for at least one r_i with high probability. Soundness then follows from the soundness of the construction in Step I.

Zero-Knowledge: For i s.t. $Q(R_i) \in$ GOOD, V does not learn any information from the zero-knowledge property of the construction in Step 1. For i s.t. $Q(r_i) \in$ BAD, V does not see anything besides r_i.

Chapter 13

CCA Security

13.1 Definition

Motivation: IND-CPA is not secure enough if an adversary is able to find an oracle that decrypts ciphertexts, which could be real-world possible attack. Hence we need to augment IND-CPA security to allow the adversary to make decryption queries of its choices. We then get two kinds of CCA security definitions.

Definition 74. *(IND-CCA-1 Security) A public key encryption scheme* $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *is IND-CCA-1 secure if for all n.u. PPT adversaries $\mathcal{A}$, there exists a negligible function $\mu(n)$ s.t. for all auxiliary inputs $z \in \{0,1\}^*$:*

$$|Pr[\textbf{\textit{Expt}}_{\mathcal{A}}^{CCA1}(1,z) = 1] - Pr[\textbf{\textit{Expt}}_{\mathcal{A}}^{CCA1}(0,z) = 1]| \leq \mu(n)$$

where $\textbf{\textit{Expt}}_{\mathcal{A}}^{CCA1}(b,z)$ is defined as:

$\textbf{\textit{Expt}}_{\mathcal{A}}^{CCA1}(0,z)$

- $st = z$
- $(pk, sk) \leftarrow \mathsf{KGen}(1^n)$
- *Decryption query phase (repeated polynomial times)*
  - $c \leftarrow \mathcal{A}(pk, st)$
  - $m \leftarrow \mathsf{Dec}(sk, c)$
  - $st = (st, m)$
- $(m_0, m_1) \leftarrow \mathcal{A}(pk, st)$
- $c^* \leftarrow \mathsf{Enc}(pk, m_b)$
- *Output* $b' \leftarrow \mathcal{A}(pk, c^*, st)$

Definition 75. *(IND-CCA-2 Security) A public key encryption scheme* $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *is IND-CCA-2 secure if for all n.u. PPT adversaries $\mathcal{A}$, there exists a negligible function $\nu(n)$ s.t. for all auxiliary inputs $z \in \{0,1\}^*$:*

$$|Pr[\textbf{\textit{Expt}}_{\mathcal{A}}^{CCA2}(1,z)=1] - Pr[\textbf{\textit{Expt}}_{\mathcal{A}}^{CCA2}(0,z)=1]| \le \nu(n)$$

where $\textbf{\textit{Expt}}_{\mathcal{A}}^{CCA2}(b,z)$ is defined as:
$\textbf{\textit{Expt}}_{\mathcal{A}}^{CCA2}(0,z)$

- $st = z$
- $(pk, sk) \leftarrow \mathsf{KGen}(1^n)$
- *Decryption query phase 1 (repeated polynomial times)*
  - $c \leftarrow \mathcal{A}(pk, st)$
  - $m \leftarrow \mathsf{Dec}(sk, c)$
  - $st = (st, m)$
- $(m_0, m_1) \leftarrow \mathcal{A}(pk, st)$
- $c^* \leftarrow \mathsf{Enc}(pk, m_b)$
- *Decryption query phase 2 (repeated polynomial times)*
  - $c \leftarrow \mathcal{A}(pk, c^*, st)$
  - *If $c = c^*$, output reject*
  - $m \leftarrow \mathsf{Dec}(sk, c)$
  - $st = (st, m)$
- *Output $b' \leftarrow \mathcal{A}(pk, c^*, st)$*

Note: CCA-2 is stronger than CCA-1 as it can make queries not only before challenge (as CCA-1) and also after challenge. And to prevent trivial attacks, decryption queries c should be different from the challenge ciphertext c^*.

13.2 IND-CCA-1 Construction

Main Challenge When building IND-CCA-1 secure PKE starting from IND-CPA secure PKE, we should not use the secret key in the secure experiment. However, we need the secret key to answer the decryption queries of the adversary. Thus the main idea is to use *two copies of the encryption scheme.*

Main Idea We could encrypt a message twice, using each of the two copies of the encryption scheme. To answer a decryption query (c_1, c_2), we only need to decrypt one of the two ciphertext. That means, we only need to know one of the secret key to answer the decryption queries. We can then use the IND-CPA security of another encryption scheme whose secret key is not used to answer decryption queries. Then switch the secret key and use IND-CPA security of the other one.

But there's a problem. What if the adversary sends (c_1, c_2) such that c_1 and c_2 are ciphertext of different messages? To solve this, we modify the scheme such that the encryption of messages m contains a NIZK proof that proves that c_1 and c_2 encrypts same message m.

Theorem 32. *(Naor-Yung) Assuming that NIZKs in the CRS model and IND-CPA secure public-key encryption, the encryption scheme* $(\mathsf{KGen}', \mathsf{Enc}', \mathsf{Dec}')$ *below is IND-CCA-1 secure public-key encryption.*

Let $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be an IND-CPA encryption scheme.

Let $(\mathsf{K}, \mathsf{P}, \mathsf{V})$ be an adaptive NIZK with Simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$.

$\mathsf{KGen}'(1^n)$:

- Compute (pk_1, sk_1) and (pk_2, sk_2) using $\mathsf{KGen}(1^n)$
- Compute $\sigma \leftarrow \mathsf{K}(1^n)$
- Output $pk' = (pk_1, pk_2, \sigma), sk' = sk_1$

$\mathsf{Enc}'(pk', m)$:

- Compute $c_i \leftarrow \mathsf{Enc}(pk_i, m; r_i)$ for $i \in [2]$
- Compute $\pi \leftarrow \mathsf{P}(\sigma, x, w)$ where $x = (pk_1, pk_2, c_1, c_2)$, $w = (m, r_1, r_2)$ and $R(x, w) = 1$ iff c_1 and c_2 encrypts the same message m.
- Output $C =)c_1, c_2, \pi$

$\mathsf{Dec}'(sk', c')$: If $\mathsf{V}(\sigma, \pi) = 0$, output $\perp$. Else, output $\mathsf{Dec}(sk_1, c_1)$.

Proof We use Hybrid Lemma to prove the theorem. We construct hybrids as follows:

Hybrids H_0: $= \mathbf{Expt}_{\mathcal{A}}^{CCA1}(0, z)$

- $(pk_i, sk_i) \leftarrow \mathsf{KGen}(1^n)$ for $i \in [2]$
- $\sigma \leftarrow \mathsf{K}(1^n)$
- $pk' = (pk_1, pk_2, \sigma), sk' = sk_1$
- On receiving a decryption query $c = (c_1, c_2, \pi)$ from $\mathcal{A}(z, pk')$, if $\mathsf{V}(\sigma, x = (c_1, c_2), \pi) = 1$, return $\mathsf{Dec}(sk' = sk_1, c_1)$
- $(m_0, m_1) \leftarrow \mathcal{A}(z, pk')$
- $c_1^* \leftarrow \mathsf{Enc}(pk_1, m_0; r_1^*)$
- $c_2^* \leftarrow \mathsf{Enc}(pk_2, m_0; r_2^*)$
- $\pi^* \leftarrow \mathsf{P}(\sigma, x^* = (c_1^*, c_2^*), w^* = (m_0, r_1, r_2))$
- Output $\mathcal{A}(z, pk', C = (c_1^*, c_2^*, \pi^*))$

Hybrids H_1:

- $(pk_i, sk_i) \leftarrow \mathsf{KGen}(1^n)$ for $i \in [2]$
- $(\sigma, \tau) \leftarrow \mathcal{S}_0(1^n)$
- $pk' = (pk_1, pk_2, \sigma), sk' = sk_1$
- On receiving a decryption query $c = (c_1, c_2, \pi)$ from $\mathcal{A}(z, pk')$, if $\mathsf{V}(\sigma, x = (c_1, c_2), \pi) = 1$, return $\mathsf{Dec}(sk' = sk_1, c_1)$
- $(m_0, m_1) \leftarrow \mathcal{A}(z, pk')$
- $c_1^* \leftarrow \mathsf{Enc}(pk_1, m_0; r_1^*)$
- $c_2^* \leftarrow \mathsf{Enc}(pk_2, m_0; r_2^*)$
- $\pi^* \leftarrow \mathcal{S}_1(\sigma, \tau, x^* = (c_1^*, c_2^*))$
- Output $\mathcal{A}(z, pk', C = (c_1^*, c_2^*, \pi^*))$

Hybrids H_2:

- $(pk_i, sk_i) \leftarrow \mathsf{KGen}(1^n)$ for $i \in [2]$
- $(\sigma, \tau) \leftarrow \mathcal{S}_0(1^n)$
- $pk' = (pk_1, pk_2, \sigma), sk' = sk_1$
- On receiving a decryption query $c = (c_1, c_2, \pi)$ from $\mathcal{A}(z, pk')$, if $\mathsf{V}(\sigma, x = (c_1, c_2), \pi) = 1$, return $\mathsf{Dec}(sk' = sk_1, c_1)$
- $(m_0, m_1) \leftarrow \mathcal{A}(z, pk')$
- $c_1^* \leftarrow \mathsf{Enc}(pk_1, m_0; r_1^*)$
- $c_2^* \leftarrow \mathsf{Enc}(pk_2, m_1; r_2^*)$
- $\pi^* \leftarrow \mathcal{S}_1(\sigma, \tau, x^* = (c_1^*, c_2^*))$
- Output $\mathcal{A}(z, pk', C = (c_1^*, c_2^*, \pi^*))$

Hybrids H_3:

- $(pk_i, sk_i) \leftarrow \mathsf{KGen}(1^n)$ for $i \in [2]$
- $(\sigma, \tau) \leftarrow \mathcal{S}_0(1^n)$
- $pk' = (pk_1, pk_2, \sigma)$, $sk' = sk_2$
- On receiving a decryption query $c = (c_1, c_2, \pi)$ from $\mathcal{A}(z, pk')$, if $\mathsf{V}(\sigma, x = (c_1, c_2), \pi) = 1$, return $\mathsf{Dec}(sk' = sk_2, c_2)$
- $(m_0, m_1) \leftarrow \mathcal{A}(z, pk')$
- $c_1^* \leftarrow \mathsf{Enc}(pk_1, m_0; r_1^*)$
- $c_2^* \leftarrow \mathsf{Enc}(pk_2, m_1; r_2^*)$
- $\pi^* \leftarrow \mathcal{S}_1(\sigma, \tau, x^* = (c_1^*, c_2^*))$
- Output $\mathcal{A}(z, pk', C = (c_1^*, c_2^*, \pi^*))$

Hybrids H_4:

- $(pk_i, sk_i) \leftarrow \mathsf{KGen}(1^n)$ for $i \in [2]$
- $(\sigma, \tau) \leftarrow \mathcal{S}_0(1^n)$
- $pk' = (pk_1, pk_2, \sigma)$, $sk' = sk_2$
- On receiving a decryption query $c = (c_1, c_2, \pi)$ from $\mathcal{A}(z, pk')$, if $\mathsf{V}(\sigma, x = (c_1, c_2), \pi) = 1$, return $\mathsf{Dec}(sk' = sk_2, c_2)$
- $(m_0, m_1) \leftarrow \mathcal{A}(z, pk')$
- $c_1^* \leftarrow \mathsf{Enc}(pk_1, m_1; r_1^*)$
- $c_2^* \leftarrow \mathsf{Enc}(pk_2, m_1; r_2^*)$
- $\pi^* \leftarrow \mathcal{S}_1(\sigma, \tau, x^* = (c_1^*, c_2^*))$
- Output $\mathcal{A}(z, pk', C = (c_1^*, c_2^*, \pi^*))$

Hybrids H_5:

- $(pk_i, sk_i) \leftarrow \mathsf{KGen}(1^n)$ for $i \in [2]$
- $(\sigma, \tau) \leftarrow \mathcal{S}_0(1^n)$
- $pk' = (pk_1, pk_2, \sigma)$, $sk' = sk_1$
- On receiving a decryption query $c = (c_1, c_2, \pi)$ from $\mathcal{A}(z, pk')$, if $\mathsf{V}(\sigma, x = (c_1, c_2), \pi) = 1$, return $\mathsf{Dec}(sk' = sk_1, c_1)$
- $(m_0, m_1) \leftarrow \mathcal{A}(z, pk')$
- $c_1^* \leftarrow \mathsf{Enc}(pk_1, m_1; r_1^*)$
- $c_2^* \leftarrow \mathsf{Enc}(pk_2, m_1; r_2^*)$
- $\pi^* \leftarrow \mathcal{S}_1(\sigma, \tau, x^* = (c_1^*, c_2^*))$
- Output $\mathcal{A}(z, pk', C = (c_1^*, c_2^*, \pi^*))$

Hybrids H_6: $=$ **Expt**$_{\mathcal{A}}^{CCA1}(0, z)$

- $(pk_i, sk_i) \leftarrow \mathsf{KGen}(1^n)$ for $i \in [2]$
- $\sigma \leftarrow \mathsf{K}(1^n)$
- $pk' = (pk_1, pk_2, \sigma)$, $sk' = sk_1$
- On receiving a decryption query $c = (c_1, c_2, \pi)$ from $\mathcal{A}(z, pk')$, if $\mathsf{V}(\sigma, x = (c_1, c_2), \pi) = 1$, return $\mathsf{Dec}(sk' = sk_1, c_1)$
- $(m_0, m_1) \leftarrow \mathcal{A}(z, pk')$
- $c_1^* \leftarrow \mathsf{Enc}(pk_1, m_1; r_1^*)$
- $c_2^* \leftarrow \mathsf{Enc}(pk_2, m_1; r_2^*)$
- $\pi^* \leftarrow \mathsf{P}(\sigma, x^* = (c_1^*, c_2^*), w^* = (m_1, r_1, r_2))$
- Output $\mathcal{A}(z, pk', C = (c_1^*, c_2^*, \pi^*))$

In short, the changes in hybrids are:

- H_0 : Expt$_{\mathcal{A}}^{CCA1}(1, z)$.
- H_1: Simulate the CRS in public-key and simulate the proof in challenge ciphertext.
- H_2: Modify c_2^* in challenge ciphertext to be an encryption of m_1.
- H_3: Change the decryption key to sk_2.
- H_4: Modify c_2^* in challenge ciphertext to be an encryption of m_1.
- H_5: Change the decryption key back to sk_1.
- H_6 : Expt$_{\mathcal{A}}^{CCA1}(0, z)$:

Now we argue the indistinguishability of these hybrids.

$H_0 \approx H_1$: This follows from the zero knowledge property of NIZK. Suppose that $\mathcal{A}'$ can distinguish H_0 and H_1 with at least a noticeable probability $\frac{1}{p(n)}$ where $p(n)$ is a polynomial function. Then we can build a distinguisher $\mathcal{D}$ against the zero-knowledge property of NIZK: on input (σ, π), $\mathcal{D}$ runs the experiment with (σ) and π^* replaced by input. $\mathcal{D}$ runs as follows:
$\mathcal{D}(\sigma, \pi)$

- $(pk_i, sk_i) \leftarrow \mathsf{KGen}(1^n)$ for $i \in [2]$
- $pk' = (pk_1, pk_2, \sigma), sk' = sk_1$
- Receive decryption queries from $\mathcal{A}$: $c = (c_1, c_2, \pi)$ from $\mathcal{A}(z, pk')$, if $\mathsf{V}(\sigma, x = (c_1, c_2), \pi) = 1$, return $\mathsf{Dec}(sk' = sk_1, c_1)$
- $(m_0, m_1) \leftarrow \mathcal{A}(z, pk')$
- $c_1^* \leftarrow \mathsf{Enc}(pk_1, m_0; r_1^*)$
- $c_2^* \leftarrow \mathsf{Enc}(pk_2, m_0; r_2^*)$
- Pass the output $\mathcal{A}(z, pk', C = (c_1^*, c_2^*, \pi))$ to $\mathcal{A}'$. If $\mathcal{A}'$ says the output is sampled from H_0, output "real proof". Else if $\mathcal{A}'$ says the output is from H_1, output "simulated proof".

Notice that $Pr[\mathcal{D}$ outputs real proof$] = Pr[\mathcal{A}'$ output $H_0]$ and that $Pr[\mathcal{D}$ outputs simulated proof$] = Pr[\mathcal{A}'$ output $H_1]$. It follows that $\mathcal{D}$ can distinguishes the real and simulated proof with noticeable probability $\frac{1}{p(n)}$. This contradicts the zero-knowledge property of NIZK.

Actually, notice that even though $x \notin L$, simulator $(\mathcal{S}_0, \mathcal{S}_1)$ can still come up with a simulated proof. Otherwise, simulator can actually decide L in polynomial time!

$H_1 \approx H_2$: This follows from the IND-CPA security of $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ with sk_2. Suppose that $\mathcal{A}'$ can distinguish H_1 and H_2 with at least a noticeable probability $\frac{1}{p(n)}$ where $p(n)$ is a polynomial function. Then we can build an adversary $\mathcal{B}$ against the IND-CPA security. $\mathcal{B}$ runs as follows:

- $(pk_1, sk_1) \leftarrow \mathsf{KGen}(1^n)$.
- Let pk_2 be the public key $\mathcal{B}$ got from challenger.
- $(\sigma, \tau) \leftarrow \mathcal{S}_0(1^n)$
- $pk' = (pk_1, pk_2, \sigma), sk' = sk_1$
- Receive decryption queries from $\mathcal{A}$: $c = (c_1, c_2, \pi)$ from $\mathcal{A}(z, pk')$, if $\mathsf{V}(\sigma, x = (c_1, c_2), \pi) = 1$, return $\mathsf{Dec}(sk' = sk_1, c_1)$
- Run $\mathcal{A}$ to get message query (m_0, m_1) and pass (m_0, m_1) to challenger.
- $c_1^* \leftarrow \mathsf{Enc}(pk_1, m_0; r_1^*)$

- Let c_2^* be the cipher text $\mathcal{B}$ got from challenger.
- Pass the output $\mathcal{A}(z, pk', C = (c_1^*, c_2^*, \pi))$ to $\mathcal{A}'$. If $\mathcal{A}'$ says the output is sampled from H_1, output $b = 0$. Else if $\mathcal{A}'$ says the output is from H_2, output $b = 1$.

When challenger choose to encrypt m_0, the output passed to $\mathcal{A}'$ is identical to that in H_1; if it is m_1 that is encrypted, the output is identical to H_2. Note that $\mathcal{B}$ can handle the decryption queries from $\mathcal{A}$ because $\mathcal{B}$ generates (pk_1, sk_1) itself. Also, it doesn't matter that $\mathcal{B}$ has no access to the randomness used to encrypt m_0, as the simulator $\mathcal{S}_1$ doesn't need r_2 to simulate the proof (unlike the real prover). Thus

$$Pr[\mathcal{B} \text{ distinguishes encryption of } m_0 \text{ and } m_1] = Pr[\mathcal{A} \text{ distinguishes } H_1 \text{ and } H_2] \geq \frac{1}{p(n)}$$

This contradicts the IND-CPA security of the PKE.

$H_2 \approx H_3$: This follows from the soundness of NIZK. Notice that the adversary can only makes successful decryption queries (c_1, c_2) if c_1 and c_2 encrypts the same message. Suppose $\mathcal{A}'$ can distinguishes H_2 and H_3 with noticeable probability. Then $Pr[\mathcal{A} \text{ distinguishes } H_2 \text{ and } H_3] = Pr[E]$ where E denotes the event that c_1 and c_2 encrypts different messages but $V(\sigma, (c_1, c_2), \pi) = 1$. Let $L = \{(c_1, c_2) | c_1 \text{ and } c_2 \text{ encrypts same message}\}$. According to the soundness property of NIZK, there exists $\nu(n)$ such that

$$Pr[\sigma \leftarrow \mathsf{K}(1^n), \exists(x, \pi) s.t. x \notin L \wedge V(\sigma, x, \pi) = 1] \leq \nu(n)$$

Now we argue that

$$Pr[(\sigma, \tau) \leftarrow \mathcal{S}_0(1^n), \exists(x, \pi) s.t. x \notin L \wedge V(\sigma, x, \pi) = 1] \leq \nu(n)$$

If not, suppose the probability above is at least $\frac{1}{p(n)}$ where $p(\cdot)$ is a polynomial function. We can then build distinguisher $\mathcal{B}$ that can tell the random string and the simulated string apart. On input σ, $\mathcal{B}$ runs as follows:

- $(pk_i, sk_i) \leftarrow \mathsf{KGen}(1^n)$ for $i \in [2]$.
- $pk' = (pk_1, pk_2, \sigma), sk' = sk_1$
- On each decryption query $c = (c_1, c_2, \pi)$ from $\mathcal{A}(z, pk')$, if $\mathsf{Dec}(sk' = sk_1, c_1) \neq \mathsf{Dec}(sk' = sk_2, c_2)$ but $\mathsf{V}(\sigma, x = (c_1, c_2), \pi) = 1$, return 1. Otherwise, repeat dealing with next query.

It's obvious that if σ is real random string, then the probability $\mathcal{B}$ outputs 1 is negligible. If σ is generated by simulator, then the probability $\mathcal{B}$ outputs 1 is at least $1 - (1 - \frac{1}{p(n)})^N$ where N is the number of queries made by $\mathcal{A}$. Hence $\mathcal{B}$ could distinguish the real random string with the one simulated by the simulator, which is a contradiction that NIZK is zero-knowledge. Hence $Pr[E]$ is negligible, which implies that $H_2 \approx H_3$.

$H_3 \approx H_4$: follows in the same manner as $H_1 \approx H_2$.

$H_4 \approx H_5$: follows in the same manner as $H_2 \approx H_3$.

$H_3 \approx H_4$: follows in the same manner as $H_0 \approx H_1$. Notice that now c_1^* and c_2^* are encrypting same message, hence P can come up with a valid proof.

Above all, $H_0 \approx H_6$, which implies the IND-CCA-1 security of the scheme $\mathsf{KGen}', \mathsf{Enc}', \mathsf{Dec}'$. □

13.3 IND-CCA-2 Security

We begin by defining the challenge experiment $\textbf{Expt}_{\mathcal{A}}^{\mathsf{CCA2}}(b, z)$ for an adversary in the CCA-2 Security model.

$\boxed{\textbf{Expt}_{\mathcal{A}}^{\mathsf{CCA2}}(b, z)}$:

```
– st = $z$
– $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$
– Decryption query phase 1 (repeated poly times):
    · $c \leftarrow \mathcal{A}(pk, \mathsf{st})$
    · $m \leftarrow \mathsf{Dec}(sk, c)$
    · st = (st, $m$)
– $(m_0, m_1) \leftarrow \mathcal{A}(pk, \mathsf{st})$
– $c^* \leftarrow \mathsf{Enc}(pk, m_b)$
– Decryption query phase 2 (repeated poly times):
    · $c \leftarrow \mathcal{A}(pk, c^*\ \mathsf{st})$
    · If $c = c^*$, output reject.
    · $m \leftarrow \mathsf{Dec}(sk, c)$
    · st = (st, $m$)
– Output $b' \leftarrow \mathcal{A}(pk, c^*, \mathsf{st})$
```

Definition 76. ***IND-CCA-2 Security**:*
A public-key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is IND-CCA-1 secure if for all n.u.*

PPT adversaries $\mathcal{A}$, there exists a negligible function $\mu(.)$, s.t. for all auxiliary inputs $z \in \{0,1\}^*$:

$$|\Pr[\mathbf{Expt}_{\mathcal{A}}^{\mathsf{CCA2}}(1,z)=1] - \Pr[\mathbf{Expt}_{\mathcal{A}}^{\mathsf{CCA2}}(0,z)=1]| \leq \mu(n)$$

A CCA-1 secure encryption scheme does not necessarily guarantee security in the CCA-2 model. This is mainly because in CCA-2, the challenge ciphertext is known to the adversary before the second decryption query phase. Thus, the adversary may be able to “maul" the challenge ciphertext into another ciphertext and then request decryption in the second phase. This is called *malleability attack*.
Such attacks can be prevented, if we make the encryption *non-malleable*, i.e., ensure that the adversary's decryption query is “independent" of (instead of just being different from) the challenge ciphertext.

13.4 CCA-2 Secure Public-Key Encryption

The first construction of CCA-2 secure encryption scheme was given by Dolev-Dwork-Naor. The following cryptographic primitives are required for this construction:

– An IND-CPA secure encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$
– An adaptive NIZK proof $(\mathsf{K}, \mathsf{P}, \mathsf{V})$
– A strongly unforgeable one-time signature (OTS) scheme $(\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$. Recall, that for a strongly unforgeable signature scheme we require, that it should be computationally hard for an adversary to come up with a new signature on a message, even if a signature corresponding to that message is already known to him. We assume, without loss of generality that, verification keys in OTS scheme are of length n.

Remark. Note that apart from the primitives used in the construction a CCA-1 secure encryption scheme, we also require a signature scheme. This is mainly required to cater to the additional requirement of non-malleability of the encryption scheme in the CCA-2 model.

13.4.1 Construction

Assuming we have an IND-CPA secure encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, an adaptive NIZK proof $(\mathsf{K}, \mathsf{P}, \mathsf{V})$ and a strongly unforgeable OTS scheme $(\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$, we construct an encryption scheme $(\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ as follows:

$\boxed{\mathsf{Gen}'(1^n)}$ **: Execute the following steps:**

- Compute CRS for NIZK:
$$\sigma \leftarrow \mathsf{K}(1^n)$$
- Compute $2n$ key pairs of IND-CPA encryption scheme:
$$(pk_i^j, sk_i^j) \leftarrow \mathsf{Gen}(1^n)$$
where $j \in \{0,1\}$, $i \in [n]$.
- Output $pk' = (\begin{bmatrix} pk_1^0 & pk_2^0 & \dots & pk_n^0 \\ pk_1^1 & pk_2^1 & \dots & pk_n^1 \end{bmatrix}, \sigma)$, $sk' = \begin{bmatrix} sk_1^0 \\ sk_1^1 \end{bmatrix}$

$\boxed{\mathsf{Enc}'(pk', m)}$ **: Execute the following steps:**

- Compute key pair for OTS scheme:
$$(SK, VK) \leftarrow \mathsf{Setup}(1^n)$$
- Let $VK = VK_1, ...VK_n$. For every $i \in [n]$, encrypt m using $pk_i^{VK_i}$ and randomness r_i:
$$c_i \leftarrow \mathsf{Enc}(pk_i^{VK_i}, m; r_i)$$
- Compute proof that each c_i encrypts the same message:
$$\pi \leftarrow \mathsf{P}(\sigma, x, w)$$
where $x = (\{pk_i^{VK_i}\}, \{c_i\})$, $w = (m, \{r_i\})$ and $R(x, w) = 1$ iff every c_i encrypts the same message m.
- Sign everything:
$$\Phi \leftarrow \mathsf{Sign}(SK, M)$$
where $M = (\{c_i\}, \pi)$
- Output $c' = (VK, \{c_i\}, \pi, \Phi)$

$\boxed{\mathsf{Dec}'(sk', c')}$ **: Execute the following steps:**

- Parse $c' = (VK, \{c_i\}, \pi, \Phi)$
- Let $M = (\{c_i\}, \pi)$
- Verify the signature: Output $\perp$ if
$$\mathsf{Verify}(VK, M, \Phi) = 0$$
- Verify the NIZK proof: Output $\perp$ if
$$\mathsf{V}(\sigma, x, \pi) = 0$$
where $x = (\{pk_i^{VK_i}\}, \{c_i\})$

– Else, decrypt the first ciphertext component:

$$m' \leftarrow \mathsf{Dec}(sk_1^{VK_1}, c_1)$$

– Output m'

Remark. Note that key pair for the signature scheme is not generated in $\mathsf{Gen}'(.)$, because we want to construct a public key encryption scheme. If the key pair for signature scheme, were to be generated in $\mathsf{Gen}'(.)$, the signing key SK, would have to be kept hidden. As a result (because of the structure of ciphertext in this construction), given only the public key, not everybody would be able to encrypt messages, which would make it a secret key encryption scheme.

13.4.2 Security

Theorem 33. *The encryption scheme presented above, is CCA-2 secure if* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is an IND-CPA secure encryption scheme,* $(\mathsf{K}, \mathsf{P}, \mathsf{V})$ *is an adaptively-secure NIZK proof system, and* $(\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ *is a strongly- unforgeable OTS scheme.*

Proof We begin by outlining the intuition to argue security of the above construction. Consider the decryption queries in the second phase, i.e., after the adversary receives the challenge ciphertext C^*. Let $C \neq C^*$ be a decryption query. Then the following two cases are possible:

- **Case 1:** $VK = VK^*$
  The verification key VK in C and the verification key VK^* in C^* are same.
  $\Rightarrow (\{c_i^*\}, \pi^*, \Phi^*) \neq (\{c_i\}, \pi, \Phi)$
  If this is the case, then we have been able to generate different signatures corresponding to the same verification key and thus, can break the strong unforgeability of the OTS scheme.
- **Case 2:** $VK \neq VK^*$
  In this case, VK and VK^* must differ in atleast one position $\ell \in [n]$:
  - Answer decryption query using the secret key $sk_\ell^{VK_i}$
  - Knowledge of secret keys $sk_i^{VK_i^*}$, for $i \in [n]$ is not required.
  - Reduce to IND-CPA security of underlying encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

We now construct the following hybrids, to prove security of the above construction in CCA-2 attack model.

$\mathbf{H_0} := \mathbf{Expt}_{\mathcal{A}}^{\mathsf{CCA2}}(0, z)$ (Honest Encryption of m_0)

– $\sigma \leftarrow \mathsf{K}(1^n)$

- $(pk_i^j, sk_i^j) \leftarrow \mathsf{Gen}(1^n)$ for $j \in \{0,1\}, i \in [n]$.
- $pk' = (\{pk_i^0, pk_i^1\}, \sigma)$, $sk' = (sk_1^0, sk_1^1)$.
- On receiving a decryption query $c = (VK, \{c_i\}, \pi, \Phi)$ from $\mathcal{A}(pk', z)$,
  if $\mathsf{Verify}(VK, M = (\{c_i\}, \pi), \Phi) = 1$ and $\mathsf{V}(\sigma, x = (\{pk_i^{VK_i}\}, \{c_i\}), \pi) = 1$,
  return $\mathsf{Dec}(sk_1^{VK_1}, c_1)$
- $(m_0, m_1) \leftarrow \mathcal{A}(pk, z)$
- $(SK^*, VK^*) \leftarrow \mathsf{Setup}(1^n)$, $VK^* = VK_1^*, ..., VK_n^*$
- $c_i^* \leftarrow \mathsf{Enc}(pk_i^{VK_i^*}, m_0; r_i^*)$
- $\pi^* \leftarrow \mathsf{P}(\sigma, x^* = (\{pk_i^{VK_i^*}\}, \{c_i^*\}), w^* = (m_0, \{r_i^*\}))$
- $\Phi^* \leftarrow \mathsf{Sign}(SK^*, M* = (\{c_i\}, \pi))$
- $c^* = (VK^*, \{c_i^*\}, \pi^*, \Phi^*)$
- On receiving a decryption query $c = (VK, \{c_i\}, \pi, \Phi)$ from $\mathcal{A}(pk', c^*, z)$,
  if $c \neq c^*$ and $\mathsf{Verify}(VK, M = (\{c_i\}, \pi), \Phi) = 1$ and $\mathsf{V}(\sigma, x = (\{pk_i^{VK_i}\}, \{c_i\}), \pi) = 1$,
  return $\mathsf{Dec}(sk_1^{VK_1}, c_1)$
- Output $\mathcal{A}(pk', c^*, z)$

$\mathbf{H_1}$: Compute CRS σ in public key and proof π in challenge ciphertext using NIZK simulator

- $\boxed{(\sigma, \tau) \leftarrow \mathcal{S}_0(1^n)}$
- $(pk_i^j, sk_i^j) \leftarrow \mathsf{Gen}(1^n)$ for $j \in \{0,1\}, i \in [n]$.
- $pk' = (\{pk_i^0, pk_i^1\}, \sigma)$, $sk' = (sk_1^0, sk_1^1)$.
- On receiving a decryption query $c = (VK, \{c_i\}, \pi, \Phi)$ from $\mathcal{A}(pk', z)$,
  if $\mathsf{Verify}(VK, M = (\{c_i\}, \pi), \Phi) = 1$ and $\mathsf{V}(\sigma, x = (\{pk_i^{VK_i}\}, \{c_i\}), \pi) = 1$,
  return $\mathsf{Dec}(sk_1^{VK_1}, c_1)$
- $(m_0, m_1) \leftarrow \mathcal{A}(pk, z)$
- $(SK^*, VK^*) \leftarrow \mathsf{Setup}(1^n)$, $VK^* = VK_1^*, ..., VK_n^*$
- $c_i^* \leftarrow \mathsf{Enc}(pk_i^{VK_i^*}, m_0; r_i^*)$
- $\boxed{\pi^* \leftarrow \mathcal{S}_1(\sigma, \tau, x^* = (\{pk_i^{VK_i^*}\}, \{c_i^*\}), w^* = (m_0, \{r_i^*\}))}$
- $\Phi^* \leftarrow \mathsf{Sign}(SK^*, M* = (\{c_i\}, \pi))$
- $c^* = (VK^*, \{c_i^*\}, \pi^*, \Phi^*)$
- On receiving a decryption query $c = (VK, \{c_i\}, \pi, \Phi)$ from $\mathcal{A}(pk', c^*, z)$,
  if $c \neq c^*$ and $\mathsf{Verify}(VK, M = (\{c_i\}, \pi), \Phi) = 1$ and $\mathsf{V}(\sigma, x = (\{pk_i^{VK_i}\}, \{c_i\}), \pi) = 1$,
  return $\mathsf{Dec}(sk_1^{VK_1}, c_1)$
- Output $\mathcal{A}(pk', c^*, z)$

$\mathbf{H_2}$: Choose VK^* in the beginning during Gen'

- $(\sigma, \tau) \leftarrow \mathcal{S}_0(1^n)$
- $\boxed{(SK^*, VK^*) \leftarrow \mathsf{Setup}(1^n), VK^* = VK_1^*, ..., VK_n^*}$
- $(pk_i^j, sk_i^j) \leftarrow \mathsf{Gen}(1^n)$ for $j \in \{0,1\}, i \in [n]$.
- $pk' = (\{pk_i^0, pk_i^1\}, \sigma)$, $sk' = (sk_1^0, sk_1^1)$.
- On receiving a decryption query $c = (VK, \{c_i\}, \pi, \Phi)$ from $\mathcal{A}(pk', z)$, if $\mathsf{Verify}(VK, M = (\{c_i\}, \pi), \Phi) = 1$ and $\mathsf{V}(\sigma, x = (\{pk_i^{VK_i}\}, \{c_i\}), \pi) = 1$,
  return $\mathsf{Dec}(sk_1^{VK_1}, c_1)$
- $(m_0, m_1) \leftarrow \mathcal{A}(pk, z)$
- $c_i^* \leftarrow \mathsf{Enc}(pk_i^{VK_i^*}, m_0; r_i^*)$
- $\pi^* \leftarrow \mathcal{S}_1(\sigma, \tau, x^* = (\{pk_i^{VK_i^*}\}, \{c_i^*\}), w^* = (m_0, \{r_i^*\}))$
- $\Phi^* \leftarrow \mathsf{Sign}(SK^*, M* = (\{c_i\}, \pi))$
- $c^* = (VK^*, \{c_i^*\}, \pi^*, \Phi^*)$
- On receiving a decryption query $c = (VK, \{c_i\}, \pi, \Phi)$ from $\mathcal{A}(pk', c^*, z)$, if $c \neq c^*$ and $\mathsf{Verify}(VK, M = (\{c_i\}, \pi), \Phi) = 1$ and $\mathsf{V}(\sigma, x = (\{pk_i^{VK_i}\}, \{c_i\}), \pi) = 1$,
  return $\mathsf{Dec}(sk_1^{VK_1}, c_1)$
- Output $\mathcal{A}(pk', c^*, z)$

$\underline{\mathbf{H_3}:}$

- $(\sigma, \tau) \leftarrow \mathcal{S}_0(1^n)$
- $(SK^*, VK^*) \leftarrow \mathsf{Setup}(1^n), VK^* = VK_1^*, ..., VK_n^*$
- $(pk_i^j, sk_i^j) \leftarrow \mathsf{Gen}(1^n)$ for $j \in \{0,1\}, i \in [n]$.
- $pk' = (\{pk_i^0, pk_i^1\}, \sigma)$
- On receiving a decryption query $c = (VK, \{c_i\}, \pi, \Phi)$ from $\mathcal{A}(pk', z)$, if $\mathsf{Verify}(VK, M = (\{c_i\}, \pi), \Phi) = 1$:
  - If $VK = VK^*$, then abort.
  - Else, let $\ell \in [n]$ be such that VK^* and VK in c differ at position ℓ. Set $sk' = sk_i^{\overline{VK_i^*}}$, $i \in [n]$, where $\overline{VK_i^*} = 1 - VK_i^*$.
    If $\mathsf{V}(\sigma, x = (\{pk_i^{VK_i}\}, \{c_i\}), \pi) = 1$, return $\mathsf{Dec}(sk_\ell^{\overline{VK_\ell^*}}, c_\ell)$
- $(m_0, m_1) \leftarrow \mathcal{A}(pk, z)$
- $c_i^* \leftarrow \mathsf{Enc}(pk_i^{VK_i^*}, m_0; r_i^*)$
- $\pi^* \leftarrow \mathcal{S}_1(\sigma, \tau, x^* = (\{pk_i^{VK_i^*}\}, \{c_i^*\}), w^* = (m_0, \{r_i^*\}))$
- $\Phi^* \leftarrow \mathsf{Sign}(SK^*, M* = (\{c_i\}, \pi))$
- $c^* = (VK^*, \{c_i^*\}, \pi^*, \Phi^*)$

- $\boxed{\begin{array}{l}\text{On receiving a decryption query } c = (VK, \{c_i\}, \pi, \Phi) \text{ from } \mathcal{A}(pk', c^* z), \text{ if} \\ \mathsf{Verify}(VK, M = (\{c_i\}, \pi), \Phi) = 1 \text{ and } c \neq c^*: \\ \quad - \text{ If } VK = VK^*, \text{ then abort.} \\ \quad - \text{ Else, let } \ell \in [n] \text{ be such that } VK^* \text{ and } VK \text{ in } c \text{ differ at position } \ell. \text{ Set } sk' = \\ \quad\;\; sk_i^{\overline{VK_i^*}}, i \in [n], \text{ where } \overline{VK_i^*} = 1 - VK_i^*. \\ \quad\;\; \text{If } \mathsf{V}(\sigma, x = (\{pk_i^{VK_i}\}, \{c_i\}), \pi) = 1, \text{ return } \mathsf{Dec}(sk_\ell^{\overline{VK_\ell^*}}, c_\ell)\end{array}}$
- Output $\mathcal{A}(pk', c^*, z)$

$\mathbf{H_4}$: Change every c_i^* in C^* to be encryption of m_1

- $(\sigma, \tau) \leftarrow \mathcal{S}_0(1^n)$
- $(SK^*, VK^*) \leftarrow \mathsf{Setup}(1^n)$, $VK^* = VK_1^*, ..., VK_n^*$
- $(pk_i^j, sk_i^j) \leftarrow \mathsf{Gen}(1^n)$ for $j \in \{0, 1\}$, $i \in [n]$.
- $pk' = (\{pk_i^0, pk_i^1\}, \sigma)$
- On receiving a decryption query $c = (VK, \{c_i\}, \pi, \Phi)$ from $\mathcal{A}(pk', z)$, if $\mathsf{Verify}(VK, M = (\{c_i\}, \pi), \Phi) = 1$:
  - If $VK = VK^*$, then abort.
  - Else, let $\ell \in [n]$ be such that VK^* and VK in c differ at position ℓ. Set $sk' = sk_i^{\overline{VK_i^*}}$, $i \in [n]$, where $\overline{VK_i^*} = 1 - VK_i^*$. If $\mathsf{V}(\sigma, x = (\{pk_i^{VK_i}\}, \{c_i\}), \pi) = 1$, return $\mathsf{Dec}(sk_\ell^{\overline{VK_\ell^*}}, c_\ell)$
- $(m_0, m_1) \leftarrow \mathcal{A}(pk, z)$
- $\boxed{c_i^* \leftarrow \mathsf{Enc}(pk_i^{VK_i^*}, m_1; r_i^*)}$
- $\pi^* \leftarrow \mathcal{S}_1(\sigma, \tau, x^* = (\{pk_i^{VK_i^*}\}, \{c_i^*\}), \boxed{w^* = (m_1, \{r_i^*\})})$
- $\Phi^* \leftarrow \mathsf{Sign}(SK^*, M^* = (\{c_i\}, \pi))$
- $c^* = (VK^*, \{c_i^*\}, \pi^*, \Phi^*)$
- On receiving a decryption query $c = (VK, \{c_i\}, \pi, \Phi)$ from $\mathcal{A}(pk', c^* z)$, if $\mathsf{Verify}(VK, M = (\{c_i\}, \pi), \Phi) = 1$ and $c \neq c^*$:
  - If $VK = VK^*$, then abort.
  - Else, let $\ell \in [n]$ be such that VK^* and VK in c differ at position ℓ. Set $sk' = sk_i^{\overline{VK_i^*}}$, $i \in [n]$, where $\overline{VK_i^*} = 1 - VK_i^*$. If $\mathsf{V}(\sigma, x = (\{pk_i^{VK_i}\}, \{c_i\}), \pi) = 1$, return $\mathsf{Dec}(sk_\ell^{\overline{VK_\ell^*}}, c_\ell)$
- Output $\mathcal{A}(pk', c^*, z)$

$\mathbf{H_5} := \mathbf{Expt}_{\mathcal{A}}^{\mathsf{CCA2}}(1, z)$ (Honest Encryption of m_1)

- $\boxed{\sigma \leftarrow \mathsf{K}(1^n)}$

- $(pk_i^j, sk_i^j) \leftarrow \mathsf{Gen}(1^n)$ for $j \in \{0,1\}$, $i \in [n]$.
- $pk' = (\{pk_i^0, pk_i^1\}, \sigma)$, $sk' = (sk_1^0, sk_1^1)$.
- $\boxed{\begin{array}{l}\text{On receiving a decryption query } c = (VK, \{c_i\}, \pi, \Phi) \text{ from } \mathcal{A}(pk', z),\\ \text{if } \mathsf{Verify}(VK, M = (\{c_i\}, \pi), \Phi) = 1 \text{ and } \mathsf{V}(\sigma, x = (\{pk_i^{VK_i}\}, \{c_i\}), \pi) = 1,\\ \text{return } \mathsf{Dec}(sk_1^{VK_1}, c_1)\end{array}}$
- $(m_0, m_1) \leftarrow \mathcal{A}(pk, z)$
- $\boxed{(SK^*, VK^*) \leftarrow \mathsf{Setup}(1^n), VK^* = VK_1^*, ..., VK_n^*}$
- $c_i^* \leftarrow \mathsf{Enc}(pk_i^{VK_i^*}, m_1; r_i^*)$
- $\boxed{\pi^* \leftarrow \mathsf{P}(\sigma, x^* = (\{pk_i^{VK_i^*}\}, \{c_i^*\}), w^* = (m_1, \{r_i^*\}))}$
- $\Phi^* \leftarrow \mathsf{Sign}(SK^*, M* = (\{c_i\}, \pi))$
- $c^* = (VK^*, \{c_i^*\}, \pi^*, \Phi^*)$
- $\boxed{\begin{array}{l}\text{On receiving a decryption query } c = (VK, \{c_i\}, \pi, \Phi) \text{ from } \mathcal{A}(pk', c^*, z),\\ \text{if } c \neq c^* \text{ and } \mathsf{Verify}(VK, M = (\{c_i\}, \pi), \Phi) = 1 \text{ and } \mathsf{V}(\sigma, x = (\{pk_i^{VK_i}\}, \{c_i\}), \pi) = 1,\\ \text{return } \mathsf{Dec}(sk_1^{VK_1}, c_1)\end{array}}$
- Output $\mathcal{A}(pk', c^*, z)$

We now argue indistinguishability of the above hybrids:

- $\mathbf{H_0} \approx \mathbf{H_1}$: Since the only difference between the two hybrids is that in H_1, CRS σ and proof π are computed using NIZK simulator. The indistinguishability of these hybrids follows from the Zero Knowledge property of NIZK.
- $\mathbf{H_1} \approx \mathbf{H_2}$: From an adversary's point of view, generating VK^* early or later does not change the distribution.
- $\mathbf{H_2} \approx \mathbf{H_3}$: We argue indistinguishability of these hybrids as follows:
  - Case 1: The protocol is aborted.
    We claim that the probability of aborting is negligible. By the definition of CCA-2, $c \neq c^*$. So if $VK = VK^*$, then it must be that $(\{c_i\}, \pi, \Phi) \neq (\{c_i^*\}, \pi^*, \Phi^*)$. Now, if $\mathsf{Verify}(VK, (\{c_i\}, \pi), \Phi) = 1$, then we can break strong unforgeability of the OTS scheme.
  - Case 2: The protocol is not aborted.
    Let ℓ be the position s.t. $VK_\ell \neq VK_\ell^*$. Note that the only difference in H_2 and H_3 in this case might be the answers to the decryption queries of adversary. In particular, in H_2, we decrypt c_1 in c using $sk_1^{VK_1}$. In contrast, in H_3, we decrypt c_ℓ in c using $sk_\ell^{\overline{VK_\ell^*}}$. Now, from soundness of NIZK, it follows that except with negligible probability, all the $c_i's$ in c encrypt the same message. Therefore decrypting c_ℓ instead of c_1 does not change the answer.

- $\mathbf{H_3} \approx \mathbf{H_4}$: Indistinguishability of these hybrids follows from the IND-CPA security if underlying PKE (Gen, Enc, Dec)
- $\mathbf{H_4} \approx \mathbf{H_5}$: Combining the above steps, we get $H_0 \approx H_3$. Indistinguishability of these hybrids (H_4 and H_5) can be argued in a similar manner (in the reverse order).

Combining the above we get $H_0 \approx H_5$.
Hence, Encryption of m_0 is computationally indistinguishable from the encryption of m_1 in the CCA-2 model. □

Bibliography

[Bar] Boaz Barak. http://www.boazbarak.org/cs127/chap001-mathematical-background.pdf.

[BC10] Mihir Bellare and David Cash. Pseudorandom functions and permutations provably secure against related-key attacks. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 666–684, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 501–519, Buenos Aires, Argentina, March 26–28, 2014. Springer, Heidelberg, Germany.

[BLMR13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 410–428, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 719–737, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part II*, volume 8270 of *Lecture*

Notes in Computer Science, pages 280–300, Bengalore, India, December 1–5, 2013. Springer, Heidelberg, Germany.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition.* MIT Press, 2009.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th Annual Symposium on Foundations of Computer Science*, pages 464–479, Singer Island, Florida, October 24–26, 1984. IEEE Computer Society Press.

[Kat] Jonathan Katz. http://www.cs.umd.edu/~jkatz/complexity/f11/all.pdf.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013: 20th Conference on Computer and Communications Security*, pages 669–684, Berlin, Germany, November 4–8, 2013. ACM Press.

[Lev03] Leonid A. Levin. The tale of one-way functions. *Probl. Inf. Transm.*, 39(1):92–103, 2003.

[Lov] Andrew D Loveless. https://sites.math.washington.edu/~aloveles/Math300Summer2011/m300Quantifiers.pdf.

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th Annual Symposium on Foundations of Computer Science*, pages 458–467, Miami Beach, Florida, October 19–22, 1997. IEEE Computer Society Press.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing*, pages 475–484, New York, NY, USA, May 31 – June 3, 2014. ACM Press.