CS 601.642/442
Modern Cryptography
Lecture Notes

Abhishek Jain

September 16, 2017

THE DOCUMENT IS UNDER CONTINUAL UPDATE

Contents

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

In this chapter we will establish some basic definitions that will be used throughout the course.

1.1 Adversary Model

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.
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. Our adversary will also be efficient (captured via its running time)
Computational Security: Security against efficient adversaries. We will mostly focus on computational security throughout the course.
Information-theoretic Security: Security against inefficient adversaries.

1.2 Algorithms and Running Times

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

Note, that the two terms "Turing machine" and "algorithm" will be used interchangeably from now on.

Definition 2 (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.*

Definition 3 **(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. If an algorithm runs in super-polynomial time $T(n) = 2^n$ or $T(n) = n^{\log n}$, then we will say it is inefficient

Note that here, c could be an arbitrary constant. In particular, it may not be small. Then, does this definition of an efficient algorithm reflect what we commonly think of as being efficient? 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. In particular, for us, inefficient

algorithms correspond to those that have super-polynomial running times such as $T(n) = 2^n$ or $T(n) = n^{logn}$.

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 4 (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.*

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.

Definition 5 (Non-uniform PPT) *A non-uniform probabilistic polynomial time Turing machine is a Turing machine A made up of a sequence of probabilistic machines $A = \{A_1, A_2, ...\}$ for which there exists a polinomial $p(\cdot)$ such that for every $A_i \in A$, the description size $|A_i|$ and the running time of A_i are at most $p(i)$. We write A(x) to denote the distribution obtained by running $A_{|x|}(x)$.*

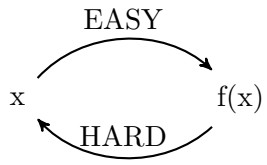
Our adversary will usually be a non-uniform probabilistic polynomial running time algorithm (n.u. PPT).

Chapter 2

One-Way Functions

2.1 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 $f(x)$.



Definition 6 (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".*

What is this probability even over? Taking probability over adversary's random tape is not a good idea. What the adversary can do is find the best available random tape and just hardwire that inside the description. Instead, we want to take the probability over the choice of x. Take any non-uniform PPT adversary, with the best strategy, and then for any input length for x, the probability that 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 definition, we will make an attempt to describe the 2 conditions in a more formal way.

2.2 Formal Definition of One-way Functions

Definition 7 (One-way Function (1^{st} Attempt)) *A function $f : \{0,1\}^* \to \{0,1\}^*$ 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. $\forall x \in \{0,1\}^*$,*

$$Pr[C(x) = f(x)] = 1.$$

What do we mean when we say the "probability is 1" - What set is this probability over? - It's the probability over the random tape input to the randomized algorithm C.

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

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

We need to specify what exactly "small" means. 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 willl call this function *negligible* Then, our security definition becomes:

$$Pr[x \leftarrow^{\$} \{0,1\}^n; A \text{ inverts } f(x)] \leq \nu(n).$$

Definition 8 (Negligible function) *A function* $\nu(n)$ *is negligible if for every c, there exists some* n_0 *such that for all* $n > n_0, \nu(n) \leq \frac{1}{n^c}$

$$\text{i.e } \forall c \in \mathbb{N}, \exists n_0 \in \mathbb{N} \text{ such that } \forall n > n_0, \nu(n) \leq \frac{1}{n^c}$$

In other words, a negligible function decays faster than all "inverse-polynomial" functions ($n^{-\omega(1)}$). An example of an obviously negligible function is an exponentially decaying function 2^{-n} or $n^{-\log(n)}$

Updating our previous definition:

Definition 9 (One-way Function (2^{nd} Attempt)) *A function* $f : \{0,1\}^* \to \{0,1\}^*$ *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.* $\forall x \in \{0,1\}^*$,

$$Pr[C(x) = f(x)] = 1.$$

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

$$Pr[x \leftarrow^{\$} \{0,1\}^n; A \text{ inverts } f(x)] \leq \nu(|x|).$$

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

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

However, if the size of y is much smaller than the size of the domain, A cannot write the inverse even if it can find it. For example, if we consider the function $f(x)$ =first $\log|x|$ Although it is trivial to invert this function ($f^{-1}(y) = y||\underbrace{0000...0}_{\text{n - lg n}}$ where$n = 2^{|y|}$), it still satisfies the definition we wrote above. Although f is easy to compute, and A cannot invert f in time $poly(|y|)$ as it needs $2^{|y|}$ to write the answer, it is not a one-way function. In order to fix this issue, we adopt the convention to always pad y and write $A(1^n, y)$, so that A has sufficient time to write the answer.

Definition 10 (Strong One-way Function) *A function $f : \{0,1\}^* \to \{0,1\}^*$ 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. $\forall x \in \{0,1\}^*$,*

$$Pr[C(x) = f(x)] = 1.$$

2. **Hard to invert:** *There exists a negligible function $\mu : \mathbb{N} \to \mathbb{R}$ s.t. for every non-uniform PPT adversary A and $\forall n$ in$\mathbb{N}$:*

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

Definition 11 **(Injective or 1-1 OWFs)** *A function f is injective if each image has a unique pre-image:*

$$f(x_1) = f(x_2) \Rightarrow x_1 = x_2$$

Definition 12 **(One Way Permutation)** *An injective one way function with the additional condition that "each image has a pre-image" or in other words t hat the domain and range of the function have the same size.*

$$f(x_1) = f(x_2) \Rightarrow x_1 = x_2.$$

2.3 Factoring Problem

Since proving that f is a one way function requires proving (at least) $P \neq NP$, we cannot tell for sure if OWFs exist unconditionally. However, by making certain assumptions about the hardness of some problems, we can construct conditional one-way functions also called "candidates". One such problem is the Factoring Problem:

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

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

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. Thus:

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

$$Pr[p \xleftarrow{\$} \prod_n; q \xleftarrow{\$} \prod_n; N = pq : A(N) \in \{p,q\}] \leq \nu(n)$$

Untill now, there have been no "good attacks" on this assumption. The best known algorithms for breaking the Factoring Assumption are:

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

Looking back at our multiplication function, it is clear that if a random x and y happen to be prime, no A could invert it, which is a GOOD case. If such a case happens with probability greater than ϵ, then every A must fail to invert the function with probability at least ϵ. If ϵ is a noticeable function, then A fails to invert the function with noticeable probability. Lastly, if ϵ is a noticeable function, then A fails with a noticeable probability. This is what we call a WEAK one-way function.

Definition 13 (Noticeable Function) *Function $\epsilon : \mathbb{N} \to \mathbb{R}$ is noticeable means that $\exists c$ and integer N_c such that $\forall n > N_c : \epsilon(n) \geq \frac{1}{n^c}$*

Definition 14 **(Weak One-way Function)** *A function $f : \{0,1\}^* \to \{0,1\}^*$ is a weak one way function (OWF) if it satisfies the following two conditions:*

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

$$Pr[C(x) = f(x)] = 1.$$

2. **Somewhat hard to invert:** *There is a noticeable function $\epsilon : \mathbb{N} \to \mathbb{R}$ s.t. for every non-uniform PPT A and $\forall n \in \mathbb{N}$:*

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

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

Theorem 1 *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 we will use Chebyshev's theorem to show that the fraction of prime numbers between 1 and 2^n is noticeable.

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

Proof.

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

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

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

$$Pr[A] \leq \nu(n)\frac{1}{4n^2} + 1(1 - \frac{1}{4n^2}) = 1 - \frac{1}{4n^2}(1 - \nu(n))$$

Now we only need to prove that $\frac{1}{4n^2}(1 - \nu(n))$ is a noticeable function. Considering that $\forall c > 0, \nu(n) \leq \frac{1}{n^c}$, hence, 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.

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 leats $1 - \frac{1}{8n^2}$. That is

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

Now we construct a non-uniform adversary algorithm $\mathcal{B}$ on input z (which is a product of two random n-bit prime numbers) to break the factoring assumption. $\mathcal{B}$ runs as follows:

1. Pick (x, y) randomly from $\{0,1\}^n \times \{0,1\}^n$;
2. if x, y are both prime, let $z' = z$;
3. else, let $z' = xy$;
4. run $\omega = A(1^{2n}, z')$;
5. if x, y are both prime, return ω.

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:

$$\begin{aligned}
&Pr[\mathcal{B} \text{ fails to break factoring assumption}] \\
=&Pr[\mathcal{B} \text{ pass input to } \mathcal{A}]Pr[\mathcal{A} \text{ fails to invert } f_\times] + Pr[\mathcal{B} \text{ fails to pass input to } \mathcal{A}] \\
\leq&Pr[\mathcal{A} \text{ fails to invert } f_\times] + Pr[\mathcal{B} \text{ fails to pass input to } \mathcal{A}] \\
\leq&\frac{1}{8n^2} + (1 - \frac{1}{4n^2}) + \leq 1 - \frac{1}{8n^2}
\end{aligned}$$

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

2.4 Weak to strong OWF

Now, once we have a weak OWF, how can we get strong OWF? Can we transform the multiply function into a strong OWF? Can we do this generically. The answer is yes, proven by Yao.

Theorem 3 (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 4 *For any weak OWF $f : \{0,1\}^n \to \{0,1\}^n$, $\exists$ polynomial $N(\cdot)$ s.t. $F : \{0,1\}^{nN(n)} \to \{0,1\}^{nN(n)} : F(x_1, ..., x_N(n)) = (f(x_1), ..., f(x_N))$ is a strong OWF.*

Proof.

Since f is weak OWF, then let $q : \mathbb{N} \to \mathbb{N}$ be a polynomial function, and for every non-uniform $\mathcal{A}$

$$Pr[x \xleftarrow{\$} \{0,1\}^n, y = f(x), \mathcal{A}(1^n, y) \in f^{-1}(y)] \leq 1 - \frac{1}{q(n)}$$

We want to find a N s.t. $(1 - \frac{1}{q(n)})^N$ tends to be very small. Thus we pick $N = 2nq(n)$, and $(1 - \frac{1}{q(n)})^N \sim e^{-2n}$.

Suppose that F is not a strong OWF. Then $\exists$ polynomial function $p'(\cdot)$ and a non-uniform $\mathcal{A}'$ s.t.

$$Pr[(x_1, ..., x_N) \xleftarrow{\$} \{0,1\}^{nN}, (y_1, ..., y_N) = F(x_1, ..., x_N), \mathcal{A}'(1^{nN}, (y_1, ..., y_N)) \in F^{-1}(y_1, ..., y_N)] \geq \frac{1}{p'(nN)}$$

Since N is a polynomial in n, the above polynomial can be re-written as:

$$Pr[(x_1, ..., x_N) \xleftarrow{\$} \{0,1\}^{nN}, (y_1, ..., y_N) = F(x_1, ..., x_N), \mathcal{A}'(1^{nN}, (y_1, ..., y_N)) \in F^{-1}(y_1, ..., y_N)] \geq \frac{1}{p(n)}$$

Now we construct a non-uniform PPT $\mathcal{B}$ to break f with probability more than $1 - \frac{1}{q(n)}$.

First we construct $\mathcal{B}_0$ on input $y = f(x)$ for random $x \in \{0,1\}^n$ as follows:

1. Randomly pick $i \in [1, N]$
2. For $j \neq i$, randomly pick $x_j \xleftarrow{\$} \{0,1\}^n$, let $y_j = f(x_j)$. Let $y_i = y$.
3. Let $(z_1, ..., z_N) = \mathcal{A}'(1^{nN}, (y_1, ..., y_N))$.
4. If $f(z_i) = y$, output z_i; otherwise, output $\perp$

To improve the chance of inverting f, we will run B_0 several times using independently chosen random coins. We define $\mathcal{B} : \{0,1\}^n \to \{0,1\}^n \cup \perp$ on input y to run $B_0(y)$ for $2nN^2p(n)$ times independently (to choose x_j independently and randomly each time). $\mathcal{B}$ outputs the first non-$\perp$ it receives. If all runs of $\mathcal{B}_0$ results in $\perp$, then $\mathcal{B}$ also outputs $\perp$.

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

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

. Otherwise, call x "bad". Then the probability that $\mathcal{B}$ fails to invert f on GOOD set is $(1 - \frac{1}{2N^2p(N)})^{2nN^2p(n)} \sim e^{-n}$, which is extremely small.

Now we prove that the fraction of GOOD set is noticeable.

Lemma 5 *There are at least* $2^n(1 - \frac{1}{2q(n)})$ *good elements in* $\{0,1\}^n$.

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

This is a contradiction to our assumption that f is $q(n)$- weak. The only thing that remains to be proven is Lemma 3.

Proof. [lemma 5] For the sake of contradiction, assume there are $> 2^n(\frac{1}{2q(n)})$ bad elements.

$$\begin{aligned}
&\Pr[\forall i \in [N],\ x_i \leftarrow \{0,1\}^n; y_i \leftarrow f(x_i) : \mathcal{A}'(y_1, ...y_N)\ succeeds] \\
&\quad = \Pr[\forall i \in [N],\ x_i \leftarrow \{0,1\}^n; y_i \leftarrow f(x_i) : \mathcal{A}'(y_1, ...y_N) \neq \perp \wedge\ some\ x_i \notin \mathsf{GOOD}] \\
&\quad + \Pr[\forall i \in [N],\ x_i \leftarrow \{0,1\}^n; y_i \leftarrow f(x_i) : \mathcal{A}'(y_1, ...y_N) \neq \perp \wedge\ all\ x_i \in \mathsf{GOOD}]
\end{aligned}$$

For each $j \in [1, N]$, we have

$$\begin{aligned}
&\Pr[\forall i \in [N],\ x_i \leftarrow \{0,1\}^n; y_i \leftarrow f(x_i) : \mathcal{A}'(y_1, ...y_N) \neq \perp \wedge x_j \notin \mathsf{GOOD}] \\
&\quad \leq \Pr[\forall i \in [N],\ x_i \leftarrow \{0,1\}^n; y_i \leftarrow f(x_i) : \mathcal{A}'(y_1, ...y_N)\ succeeds | x_j \notin \mathsf{GOOD}] \\
&\quad \leq N \times \Pr[\mathcal{B}_0(f(x_j))\ succeeds |\ x_j\ is\ bad] \\
&\quad \leq \frac{N}{2N^2 p(n)} = \frac{1}{2Np(n)}
\end{aligned}$$

So by taking union bound, we have

$$\begin{aligned}
&\Pr[\forall i \in [N],\ x_i \leftarrow \{0,1\}^n; y_i \leftarrow f(x_i) : \mathcal{A}'(y_1, ...y_N) \neq \perp \wedge\ some\ x_i \notin \mathsf{GOOD}] \\
&\quad \leq \sum_j \Pr[\forall i \in [N],\ x_i \leftarrow \{0,1\}^n; y_i \leftarrow f(x_i) : \mathcal{A}'(y_1, ...y_N) \neq \perp \wedge x_j \notin \mathsf{GOOD}] \\
&\quad \leq \frac{N}{2Np(n)} = \frac{1}{2p(n)}
\end{aligned}$$

Also,

$$\begin{aligned}
&\Pr[\forall i \in [N],\ x_i \leftarrow \{0,1\}^n; y_i \leftarrow f(x_i) : \mathcal{A}'(y_1, ...y_N) \neq \perp \wedge\ all\ x_i \in \mathsf{GOOD}] \\
&\leq \Pr[\forall i \in [N],\ x_i \leftarrow \{0,1\}^n :\ all\ x_i \in \mathsf{GOOD}] \\
&\leq (1 - \frac{1}{2q(n)})^N = (1 - \frac{1}{2q(n)})^{2nq(n)} \approx e^{-n}
\end{aligned}$$

Hence,

$$\begin{aligned}
&\Pr[\forall i \in [N],\ x_i \leftarrow \{0,1\}^n; y_i \leftarrow f(x_i) : \mathcal{A}'(y_1, ...y_N)\ succeeds] \\
&\quad < \frac{1}{2p(n)} + e^{-n} \\
&\quad < \frac{1}{p(n)}
\end{aligned}$$

This contradicts with the assumption that F is not strongly one-way. Thus F is a strong OWF.

Chapter 3

Hard Core Predicate

3.1 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, then

$$f'(x) = f(x)||x_{[1,\cdots,n/2]},$$

where $x_{[1,\cdots,n/2]}$ are the first half bits of x, is also a one-way function!

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 15 (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* negl *such that for every non-uniform PPT adversary $\mathcal{A}$, and $\forall n \in \mathbb{N}$,*

$$\Pr\left[x \xleftarrow{\$} \{0,1\}^n : \mathcal{A}(1^n, f(x)) = h(x)\right] \leq \frac{1}{2} + \mathsf{negl}(n)\,.$$

Here the randomness is over both the choice of x and the random coins used by $\mathcal{A}$.

3.2 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 6 (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 $\langle x, r \rangle = (\sum_i x_i r_i) \mod 2$.

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 $x^* = x_1^* \cdots x_n^*$.

$\mathcal{B}(1^n, f(x))$

for $i = 1$ **to** n do
  $x_i^* \leftarrow \mathcal{A}(f(x), e_i)$
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 $r \xleftarrow{\$} \{0,1\}^n$. 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 6 will be on the first homework assignment. Note that this theorem is actually very important, even outside cryptography! It has applications to learning, list-decoding codes, etc.

3.3 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: https://arxiv.org/pdf/cs/0012023.pdf.

Chapter 4

Pseudorandomness

4.1 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 a lot of randomness is required (such as for encryption).

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.

4.2 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 $G : \{0,1\}^n \to \{0,1\}^{n+1}$ 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 $\{r \xleftarrow{\$} \{0,1\}^{n+1} : r\}$. 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 16 (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 $\sum_s p_s = 1$.*

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

Generally, we will take X_n to be a distribution over $\{0,1\}^{\ell(n)}$, where $\ell(\cdot)$ is a polynomial.

Now, we will 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 D can tell X and Y apart, or the behavior of 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 D a sample of X. Then, 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 D's output with one bit. In order for X and Y to be indistinguishable, D's average score on a sample of X should be basically the same as its average score on a sample of Y.

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

or

$$\Pr[x \leftarrow X; D(1^n, x) = 1] - \Pr[y \leftarrow Y; D(1^n, y) = 1] \leq \mu(n)$$

for negligible $\mu(\cdot)$.

This brings us to the formal definition of **computational indistinguishability**.

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

$$|\Pr[x \leftarrow X; D(1^n, x) = 1] - \Pr[y \leftarrow Y; D(1^n, y) = 1]| \leq \nu(n).$$

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 D a sample from either X or Y, and ask it to identify which distribution it is from, if D is not right with probability better than $\frac{1}{2}$, then X and Y look the same to it! We will change notation a bit and set $X^{(1)} = X, X^{(0)} = Y$.

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

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

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

Proof.

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

So they are equivalent within a factor of 2.

Lemma 7 (Prediction Lemma) *Let $\{X_n^{(0)}\}, \{X_n^{(1)}\}$ be ensembles of probability distributions. Let D be a non-uniform PPT adversary that $\varepsilon(\cdot)$-distinguishes $\{X_n^{(0)}\}, \{X_n^{(1)}\}$ for infinitely many $n \in$. 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 $n \in$.

Properties of Computational Indistinguishability.

1. First, we define the notation $\{X_n\} \approx_C \{Y_n\}$ to mean computational indistinguishability.
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.

This last property is actually quite nice, and we would like to generalize it a bit.

Lemma 8 (Hybrid Lemma) *Let $X^1, \ldots, X^m$ be distribution ensembles for $m = \text{poly}(n)$. Suppose D distinguishes X^1 and X^m with advantage ε. Then $\exists i \in \{1, \cdots, m-1\}$ such that D distinguishes X^i, X^{i+1} with advantage $\geq \frac{\varepsilon}{m}$.*

This follows from the pigeonhole principle.

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.

4.3 Next-Bit Test

Our last topic for this lecture is about an interesting way to characterize pseudorandomness. We know a pseudorandom string should pass all efficient tests a true random string would pass. For a truly random string, given a subsequence, it is not possible to predict the "next bit" with probability better than $\frac{1}{2}$ if you're given the previous bits in the subsequence. So this gives us the notion of a test using the "next bit" idea. So, 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.

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

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

Theorem 9 (Completeness of the Next-bit Test) *If $\{X_n\}$ is next-bit unpredictable then $\{X_n\}$ is pseudorandom.*

Proof.
FSOC, suppose $\exists$ a n.u. PPT distinguisher D, and a polynomial $p(\cdot)$ s.t. for infinitely many $n \in \mathbb{N}$, D distinguishes X_n and $U_{l(n)}$ with probability $\frac{1}{p(n)}$. Let $\mathcal{A}$ be a machine that predicts the next bit of X_n for every n. A sequence of *hybrid distributions* is defined as:

$$H_n^i = \{x \leftarrow X_n : u \leftarrow U_{\ell(n)} : x_0 x_1 ... x_i u_{i+1} u_{i+2} ... u_{\ell(n)}\}$$

Note that the first hybrid H_n^0 is the uniform distribution $U_{\ell(n)}$, and the last hybrid $H_n^{\ell(n)}$ is the distribution X_n. Thus, D distinguishes between H_n^0 and $H_n^{\ell(n)}$ with probability $\frac{1}{p(n)}$. By hybrid lemma, $\exists$ some $i \in [0, \ell(n)]$ s.t. D distinguishes between H_n^i and H_n^{i+1} with probability $\frac{1}{p(n)\ell(n)}$. 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}. Thus, given only $x_1 ... x_i$, D can distinguish x_{i+1} from the random set. The distribution $\tilde{H}_n^i$ is defined as:

$$\tilde{H}_n^i = \{x \leftarrow X_n : u \leftarrow U_{\ell(n)} : x_0 x_1 ... x_{i-1} \bar{x_i} u_{i+1} u_{i+2} ... u_{\ell(n)}\} \text{ where } \bar{x_i} = 1 - x_i$$

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

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

The observation that D distinguishes H_n^i and H_n^{i+1} with probability $\frac{1}{p(n)\ell(n)}$ implies that D distinguishes H_n^{i+1} and $\tilde{H}_n^{i+1}$ with probability $\frac{2}{p(n)\ell(n)}$. Therefore, by the prediction lemma, $\exists$ a machine $\mathcal{A}$ s.t.

$$\Pr\left[b \leftarrow \{0,1\}; t \leftarrow H_n^{i+1,b} : D(t) = b\right] > \frac{1}{2} + \frac{1}{p(n)\ell(n)}$$

where $b = 1$ if $\mathcal{A}$ predicts a sample came from H_n^{i+1} and $b = 0$ if it is from $\tilde{H}_n^{i+1}$. Then, we construct a machine $\mathcal{A}'$ that predicts the $(i+1)^{th}$ bit of the pseudorandom sequence, x_{i+1}. $\mathcal{A}'$ picks $\ell(n) - i$ random bits from $u_{i+1} ... u_{\ell(n)} \leftarrow U^{\ell(n)-1}$ and run $g \leftarrow \mathcal{A}(t_1 ... t_i u_{i+1} ... u_{\ell(n)})$. If $g = 1$ then $\mathcal{A}'$ outputs u_{i+1}, else $\mathcal{A}'$ outputs $\bar{u}_{i+1} = 1 - u_{i+1}$. Then,

$$\begin{aligned}
&\Pr\left[t \leftarrow X_n : \mathcal{A}'(1^n, t_1 ... t_i) = t_{i+1}\right] \\
&= \Pr\left[b \leftarrow \{0,1\}; t \leftarrow H_n^{i+1,b} : \mathcal{A}(t) = 1\right] > \frac{1}{2} + \frac{1}{p(n)\ell(n)}
\end{aligned}$$

which shows that D correctly predicts the next bit with some noticeable probability. This is a contradiction. ■

4.4 Pseudorandom Generators (PRG)

We have defined the notion of pseudorandomness and next-bit unpredictability. Now, we turn to construct the definition of pseudorandom generators using the above theorem.

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

1. *(efficiency): G can be computed in polynomial time*

2. *(expansion): $|G(x)| > |x|$*

3. *$\{x \leftarrow \{0,1\}^n : G(x)\} \approx_c \{U_{\ell(n)}\}$ where $\ell(n) = |G(0^n)|$*

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

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

4.5 PRG with 1-bit Stretch

We can think of an initial construction of PRG as $G(s) = f(s)\|h(s)$ where f is a one-way function and h is the hardcore predicate associated with f. However, 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.

To solve this problem, we will set f to be a one-way *permutation* (OWP) over $\{0,1\}^n$. Now, we can 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 over $\{0,1\}^n$ if s is randomly chosen. In particular:

$$\forall y : \Pr[f(s) = y] = \Pr[s = f^{-1}(y)] = 2^{-n}.$$

  Thus, $f(s)$ is uniform and cannot start with a fix value.

Theorem 10 (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.
FSOC, suppose $\exists$ a n.u. PPT adversary $\mathcal{A}$ and a polynomial $p(n)$ s.t. $\forall n$, $\exists$ i s.t. $\mathcal{A}$ predicts the i^{th} bit with non-negligible probability $\frac{1}{p(n)}$. Since f is a permutation, the first n bits of $G(s)$ are uniformly distributed. $\mathcal{A}$ must predict $(n+1)^{th}$ bit with advantage $\frac{1}{p(n)}$, i.e.

$$\Pr[\mathcal{A}(f(s)) = h(s)] > \frac{1}{2} + \frac{1}{p(n)}$$

which contradicts the assumption that $h(s)$ is hard-core for f. Thus, G is a PRG. ■

4.6 PRG with Poly-Stretch

Lemma 11 *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:*

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

Then, G' is a PRG.

Proof. We first establish some notation. Let $G'(s) = G^m(s)$, where

$$G^0(x) = \epsilon$$
$$G^i(x) = b \| G^{i-1}(x') \text{ where } x' \| b \leftarrow G(x)$$

and ϵ denotes the empty string. FSOC, suppose $\exists$ a distinguisher D and a polynomial $p(\cdot)$ s.t. $\forall n$, D distinguishes $\{U_{m(n)}\}_n$ and $\{G'(U_n)\}_n$ with non-negligible probability $\frac{1}{p(n)}$.

Let $H_n^i = U_{m(n)-i} \| G^i(U_n)$ be the hybrid distributions for $i = 1, ..., m(n)$. Then, $H_n^0 = U_{m(n)}$ and $H_n^{m(n)} = G^{m(n)}(U_n)$ and D distinguishes H_n^0 and $H_n^{m(n)}$ with probability $\frac{1}{p(n)}$. By the hybrid lemma, $\forall n$, $\exists i$ s.t. D distinguishes H_n^i and H_n^{i+1} with probability $\frac{1}{m(n)p(n)}$. Then,

$$\begin{aligned} H_n^i &= U_{m-i} \| G^i(U_n) \\ &= U_{m-i-1} \| U_1 \| G^i(U_n) \\ H_n^{i+1} &= U_{m-i-1} \| G^{i+1}(U_n) \\ &= U_{m-i-1} \| b \| G^i(x) \text{ where } x \| b \leftarrow G(U_n) \end{aligned}$$

Suppose $\exists$ a n.u. PPT $M(y)$ that outputs from the following experiment:

$$b_{prev} \leftarrow U_{m-i-1}$$
$$b \leftarrow y_1$$
$$b_{next} \leftarrow G^i(y_2 ... y_{n+1})$$
$$\text{Output } b_{prev} \| b \| b_{next}$$

The output of D is distributed identically to the output of H_n^i if the input was sampled from U_{n+1} and H_n^{i+1} if it was from $G(U_n)$. Since $\{U_{n+1}\}_n \approx \{G(U_n)\}_n$ with advantage $\frac{1}{p(n)\ell(n)}$ and G runs in polynomial time, $\{H_n^i\}_n \approx \{H_n^{i+1}\}_n$, which is a contradiction. ■

4.7 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 defining a random function? Consider a function $f : \{0,1\}^n \to \{0,1\}^n$. If we write f as a table, where first column has input strings from 0^n to 1^n and the second column has the function value against each input, each row of the table is of the form $(x, f(x))$. Then, the size of the table is $2^n \times n = n2^n$. Thus, 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

   - A randomized program M keeps a table T (initially empty) to implement a random function F
   - On input x,
     - if x is not in the table, choose a random string y and add the entry (x, y) to T
     - otherwise, M picks (x, y) corresponding to x from T, and outputs the entry
   - The distribution of M's output is identical to that of F.

However, 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 T, if one makes **polynomial** calls to the random function.

Pseudorandom Functions (PRF): Intuition. PRF looks like a random function and is described in polynomial bits. At first, it seems like a good idea to use computational indistinguishability to make PRF "look like" a Random Function. However, the main issue with this idea is that a random function is of an exponential size. If D can't even read the input efficiently, then it can distinguish between PRFs and RFs by looking at its input size, and computational indistinguishability is violated. One way to solve this issue is to allow D to only *query* the function on inputs of its choice, and let it see the output. We'll formalize this idea in the next lecture.

4.8 Pseudorandom Functions (PRF)

PRFs look like random functions, but need only polynomial number of bits to be described and emulated. What does "look like" really mean. Let us use computational indistinguishability to define it. But the question that arises here is that, can the distinguisher, D be given the entire description of a random function or a PRF.
Issue: Since the description of a random function is of exponential size, 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 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 D, is similar to providing security by obscurity which in general is not a good idea according to Kerchoff's principle. Therefore 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. This is in accordance with Kerchoff's principle. The security of PRF is defined via game base definition.

4.8.1 Security of PRF via Game Based Definition

There are 2 players a Challenger Ch and an Adversary/Distinguisher D

Ch	D
$b \xleftarrow{\$} \{0,1\}$	
if $b == 0$: Ch implements F as a PRF else: Ch implements F as a random function	
	send queries $x_1, x_2, ...$ to Ch one-by-one.
correctly replies $F(x_1), F(x_2), ...$	
	outputs his guess b' i.e, whether F is random or a PRF.

If $b == b'$, D wins the game. Intuitively, we want that the adversary wins with only ($\frac{1}{2}$ + negligible) probability

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 D, there exists a negligible function ν such that $\forall n \in N$:*

$$|Pr[D \text{ wins } GuessGame] - 1/2| \leq \nu(n)$$

$GuessGame(1^n)$ that incorporates D, can be formally defined as follows:

- The game chooses a key for PRF and a random bit b.
- It runs D, answering every query x as follows:
  1. If $b == 0$: answer using PRF. Output $F_k(x)$.
  2. If $b == 1$: answer using random F. Keep a table T for all previous queries, if $T[x]$ exists, output $T[x]$. Else, choose a random n-bit string y, $T[x] = y$, output y.
- Game stops, when D halts and outputs a bit b'.

All queries in the game are answered using the same function (either PRF or random F). If $b' == b$, D wins the *GuessGame*.
Remark. We are ensuring that the challenger is efficient by implementing the random function using the second method. It is important to construct challengers that are efficient, because while building reductions, the reduction acts as the challenger. If the challenger is not efficient, it would lead to the construction of an inefficient reduction. Since we only consider n.u. PPT adversaries, an inefficient reduction would not be able to give us a contradiction.

4.8.2 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 $F_k : \{0,1\} \to \{0,1\}^n$ using a length doubling PRG, G as follows:

- Let G be a length doubling PRG such that, $G(s) = y_0||y_1$, where $|y_0| = |y_1| = n$.
- For PRF, set $s = k$, and $F_k(0) = y_0$, $F_k(1) = y_1$

The security of PRFs constructed like this, can be argued by the security of PRGs.
Proof. Since this PRF is directly using the output of PRG, we can say that if there exists a distinguisher A that can efficiently distinguish between a random F and this PRF, then another distinguisher B can be constructed using A, that will be able to distinguish between PRG and a random number generator in the following way:

1. B gets a 2n- bit input y which is either a random string or the output of G for some s.

2. The adversary A, which is a distinguisher for the PRF, is allowed to query on inputs of its choice. So if A sends a query for input "0", B sends the first n-bits of y to A. On input query "1", B sends the last n-bits of y to A.

3. Based on the queries and their responses, A either outputs "1" (if the function is a PRF) or "0" (if the function is random).

4. B outputs "0" if the output of A was "0", and it outputs "1", if the output of B was "1".

Since the distinguisher A can distinguish between a random F and PRF with noticeable probability. From the above construction, it follows that B can also distinguish between a random string and the output of a PRG with noticeable probability. But, we know that no such distinguisher for PRGs exists, therefore, no such distinguisher for this PRF can exist. ■

4.8.3 PRF with n-bit input

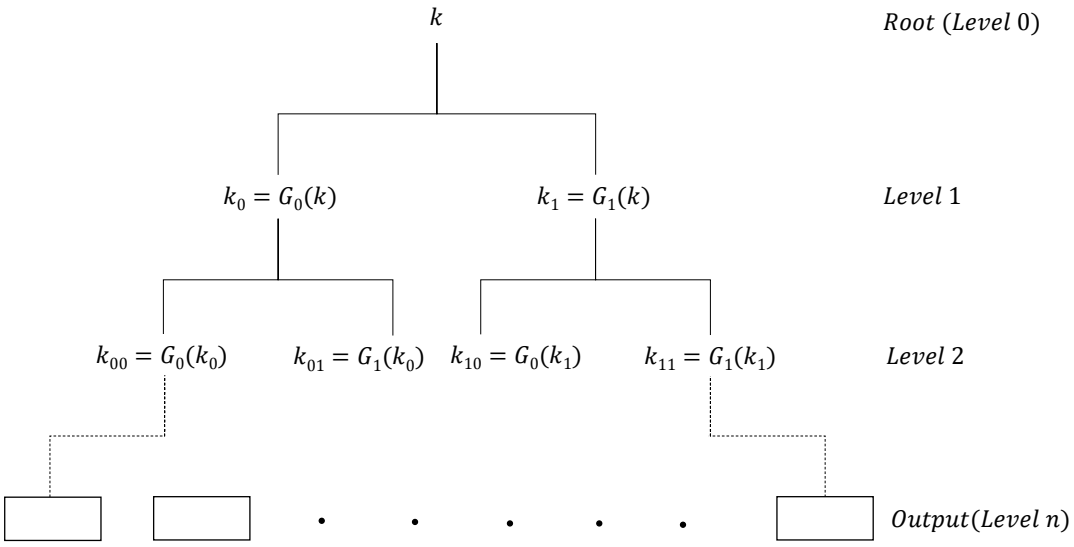
Total number of possible n-bit inputs are exponential. Since PRGs only stretch to poly(n) bits, a single PRG with poly stretch can only be used for constructing PRFs with $log(n)$ bit input. F_k: $\{0,1\}^{log(n)} \rightarrow \{0,1\}^n$ can be constructed using $G(s) = y_1||y_2||...||y_{L=poly(n)}$, where $|y_i| =$ n-bits.
However, For n-bit input, the "double and choose" policy used for constructing PRFs with 1-bit input can be used multiple times, repeatedly.

Theorem 12 (Goldreich-Goldwasser-Micali (GGM)) *If pseudorandom generators exist, then pseudorandom functions exist.*

Construction:
For a length doubling PRG G, we define G_0 and G_1 as: $G(s) = G_0(s)||G_1(s)$.
For n-bit input $x = x_1x_2...x_n$,

$$F_k(x) = G_{x_n}(G_{x_{n-1}}(.....(G_{x_1}(k))..)$$

It is convenient to think of the construction of F_k as a binary tree of size 2^n.
k denotes the root (chosen randomly). While $k_{x_1....x_l}$, denotes the the leaves of the tree at level l.
At level l, there are 2^l nodes, one for each path, denoted by $k_{x_1....x_l}$. 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 x_0, either the path traverses through the left branch or the right branch. Subsequently, the choice of branches is based on individual bits of the input string. Every input would correspond to a unique path and a unique leaf node, since atleast one bit (and correspondingly, one edge in the tree) would be different.



Efficiency: Considering the efficiency of this construction, there is no need to store this tree, the output values can be computed on the fly. To compute an output value, RunningTime$(F_k(x)) = n\times$ RunningTime$(G(.)) = n \times poly(n) = poly'(n)$.

Proof Strategy:
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. But since there are exponential number of nodes in the tree, this would result in exponential number of hybrids. While we know that the Hybrid Lemma, only works for polynomial number of hybrids. To overcome this, we use an interesting observation, which is that any PPT adversary is only allowed to make polynomial queries. Since each query corresponds to a unique path (of n nodes), total number of nodes affected by all queries $= n \times poly(n) = poly(n)$. Therefore, we only need to change polynomial number of nodes in the tree. We can achieve this by using 2 layers of nested hybrids.

Proof. Level 1 Hybrids:
H_0 : Level 0 is random, level $i > 0$ is pseudorandom. (actual PRF)
H_1 : Level 0,1 are random, level $i > 1$ is pseudorandom.
..
..
H_i : Levels $\leq i$ are random, levels $> i$ are pseudorandom.
..
..
H_n : all levels are random. (random function)

Remark. By saying that level i is random or pseudorandom, we mean that all the nodes at level i, that are affected by the adversary's queries are random or pseudorandom respectively. H_0 corresponds to a PRF, while H_n corresponds to a random function. if we assume that PRF and random function are distinguishable, then $\exists D$, such that it can differentiate between H_0 and H_n with noticable advantage $\epsilon(n)$, then by Hybrid Lemma, $\exists i \in [n]$, such

that $\exists D'_{H_i,H_{i+1}}$ that distinguishes between H_i and H_{i+1} with advantage atleast $\frac{\epsilon(n)}{n}$. The only difference between H_i and H_{i+1} is that:
- in H_i, level $i+1$ is pseudorandom.
- in H_{i+1}, level $i+1$ is random.
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, i.e., $|S| = poly(n)$.

Level 2 Hybrids (assuming all nodes in S are in lexicographic order):
$H_{i,j\in|S|}$: Same as H_i, except that all nodes at level $i+1$, that are children of nodes $\leq j$, are random.
$H_{i,0} \equiv H_i$ (a dummy hybrid)
$H_{i,|S|} \equiv H_{i+1}$

Given $D'_{H_i,H_{i+1}}$, by Hybrid Lemma, $\exists j \in |S|$ and $\exists D''_{H_{i,j},H_{i,j+1}}$ such that, it distinguishes between $H_{i,j}$ and $H_{i,j+1}$ with advantage atleast $\frac{\epsilon(n)}{n\times|S|} \geq \frac{1}{poly(n)}$ (recall that $\epsilon(n)$ is noticeable (i.e., $\geq \frac{1}{poly(n)}$)). The only difference between $H_{i,j}$ and $H_{i,j+1}$ is that:
- in $H_{i,j}$, node j in level $i+1$ is pseudorandom.
- in $H_{i,j+1}$, node j in level $i+1$ is random.
If $D''_{H_{i,j},H_{i,j+1}}$ can distinguish between $H_{i,j}$ and $H_{i,j+1}$ with a noticeable advantage, then it is possible to construct another adversary D''' that can efficiently distinguish between PRG and a random number generator as follows:

1. D''' gets a 2n-bit input $y_0||y1$ ($|y_0| = |y_1|$), that is sampled either as a random string or as $G(s)$ for some random string s.

2. In his mind, D''' 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. D''' replies to the queries of $D''_{H_{i,j},H_{i,j+1}}$ using this tree.

3. If y_0 and y_1 were pseudorandom the distribution of D''''s outputs would be similar to $H_{i,j}$ and they were random then, the output distribution would be similar to $H_{i,j+1}$.

4. If $D''_{H_{i,j},H_{i,j+1}}$ decides that the distribution of these nodes matches the distribution of $H_{i,j}$, then D''' decides that the input is pseudorandom, else if it matches the distribution of $H_{i,j+1}$, then D''' decides that the input is random.

According to the above construction, B can efficiently distinguish between a random string and the output of a PRG. Since we know that no such distinguisher for PRGs exists, therefore no $D''_{H_{i,j},H_{i,j+1}}$ can exist. From earlier definitions and assumptions, it follows that no distinguisher $D'_{H_i,H_{i+1}}$ for $H_i(H_{i,0})$ and $H_{i+1}(H_{i,|S|})$ can exist. And subsequently no D that distinguishes between a random function and a PRF can exist. Which implies that this construction of PRF is indistinguishable. ■

Chapter 5

Secret-Key Encryption

5.1 Setting

We assume that Alice and Bob share a secret $s \in \{0,1\}^n$. We do not discuss here how they were able to share that secret (this will be discussed in a later lecture). 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.

5.2 Secret-key Encryption

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

- $\mathsf{KGen}(1^n) \to s$
- $\mathsf{Enc}(s, m) \to c$
- $\mathsf{Dec}(s, c) \to m'$

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 $\mathsf{Enc}(s, m) \to c$ and sends c on the public channel to Bob. Bob then uses the decryption algorithm $\mathsf{Dec}(s, c) \to m$ to recover the message.

A secret-key encryption scheme must satisfy the two properties described below:

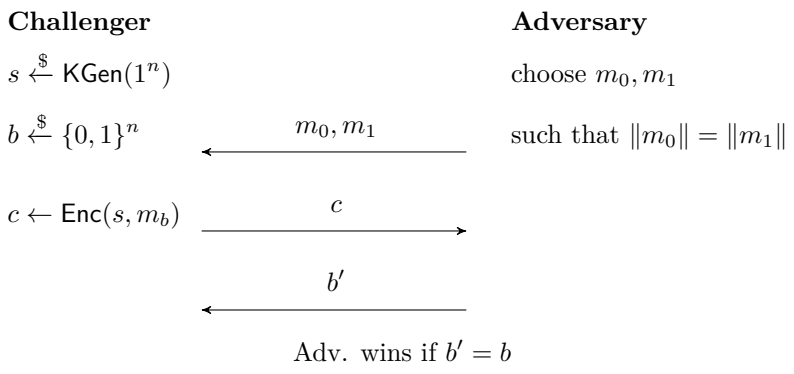
Correctness For every m, $\mathsf{Dec}(s, \mathsf{Enc}(s, m)) = m$, where $s \xleftarrow{\$} \mathsf{KGen}(1^n)$

Security Intuitively 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 IND-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 25 (IND-CPA) *A secret-key encryption scheme* $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *is IND-CPA secure if for all n.u. PPT adversaries* $\mathcal{A}$*, there exists a negligible function* $\mu(\cdot)$ *such that:*

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

Note that this is a game based definition of security, where the game operates in the following manner:



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:

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

5.2.1 One-Time Pads

Consider the following encryption scheme:

- $\mathsf{KGen}(1^n) := s \xleftarrow{\$} \{0,1\}^n$
- $\mathsf{Enc}(s, m) := s \oplus m = c$
- $\mathsf{Dec}(s, c) := s \oplus c = m$

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 distibuted. Thus two ciphertexts are more than just indistinguishable, they are in fact identically distributed.

$$\mathsf{Enc}(s \xleftarrow{\$} \{0,1\}^n, m_1) \equiv \mathsf{Enc}(s \xleftarrow{\$} \{0,1\}^n, m_2)$$

Note though that for two messages encrypted with the same key gives us $c_1 \oplus c_2 = (s \oplus m_1) \oplus (s \oplus m_1) = m_1 \oplus m_2$ which can break the security.

5.2.2 Encryption using PRGs

In the above scheme, 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) by relying on the fact that the output of a PRG is computationally indistinguishable from uniform random. Thus we can use PRGs to convert a random key of a fixed length into a pseudorandom key of the necessary length to encrypt a message of arbitrary polynomially length.

Consider the following encryption scheme:

- $\mathsf{KGen}(1^n) := s \xleftarrow{\$} \{0,1\}^n$

- $\mathsf{Enc}(s, m) := PRG(s) \oplus m = c$
- $\mathsf{Dec}(s, c) := PRG(s) \oplus c = m$

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

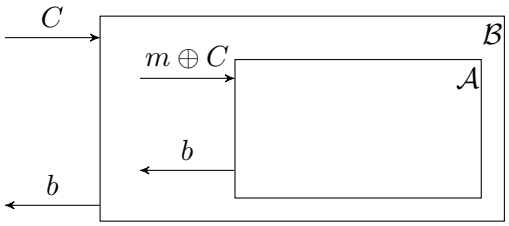
Proposition 2 (Security of Encryption using PRGs)

$$\mathsf{Enc}(s \xleftarrow{\$} \{0,1\}^n, m_1) \approx_c \mathsf{Enc}(s \xleftarrow{\$} \{0,1\}^n, m_2)$$

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:

$$\begin{aligned}
H_0: &\quad s \xleftarrow{\$} \mathsf{KGen}(1^n), \mathsf{Enc}(s, m_0) &&= m_0 \oplus PRG(s) \\
H_1: &\quad s \xleftarrow{\$} \mathsf{KGen}(1^n), \mathsf{Enc}(s, m_0) &&= m_0 \oplus R \xleftarrow{\$} \{0,1\}^n, \|R\| = \|m_0\| \\
H_2: &\quad s \xleftarrow{\$} \mathsf{KGen}(1^n), \mathsf{Enc}(s, m_1) &&= m_1 \oplus R \\
H_3: &\quad s \xleftarrow{\$} \mathsf{KGen}(1^n), \mathsf{Enc}(s, m_1) &&= m_1 \oplus PRG(s)
\end{aligned}$$

We claim that $H_0 \approx_c H_1$ and show this is true via a reduction argument. Assume that an adversary, $\mathcal{A}$, exists that can distinguish between H_0 and H_1. Then we construct an adversary $\mathcal{B}$ that can break the indistinguishability of the PRG.



The challenger to $\mathcal{B}$ flips a bit $b \xleftarrow{\$} \{0,1\}$ and then either sends $C \xleftarrow{\$} \{0,1\}^n$ or $C = PRG(s)$. Then $\mathcal{B}$ sends $C \oplus m$ to $\mathcal{A}$. Since $\mathcal{A}$ can distinguish between H_0 and H_0 it passes back the corresponding bit. This is then passed back to the challenger and is correct with non-negligible probability. The next pair of hybrids H_1 and H_2 are indistinguishable due to the indistinguishability of one-time pads. Finally H_2 and H_3 are indistinguishable by a symmetric argument to H_0 and H_1. Thus by transitivity, H_0 and H_3 are indistinguishable, which establishes that the scheme is IND-CPA secure.

■

5.3 Multi-message Secure Encryption

So far, we have only discussed encryption schemes where a key can be used to encrypt a *single* message. We now consider encryption schemes where a secret key can be used to encrypt multiple messages.

Definition 26 (Multi-message Secure Encryption) *A secret-key encryption scheme* $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *is multi-message IND-CPA secure if for all n.u. PPT adversaries* $\mathcal{A}$*, for all polynomials* $q(\cdot)$ *there exists a negligible function* $\mu(\cdot)$ *such that:*

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

This definition is very similar to the first but now in the security game the adversary sends two arrays of messages,$(m_0^1, m_0^2, \ldots, m_0^{q(n)})$ and $(m_1^1, m_1^2, \ldots, m_1^{q(n)})$ then the challenger encrypts one of them and returns an array of ciphertext $(c_1, c_2, \ldots, c_{q(n)})$. Here $q(\cdot)$ is an arbitrary polynomial chosen by the adversary $\mathcal{A}$.

Theorem 13 (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 PRG (as in the previous construction) and then use different "chunks" of the randomness to encrypt different messages. We need to keep track 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 14 (Randomized Encryption) *A multi-message secure encryption scheme cannot be deterministic and stateless.*

Proof. Suppose such a scheme existed. Then an adversary could send

$$\begin{matrix} m_0^1 & m_1^1 \\ m_0^2 & m_1^2 \end{matrix}$$

such that $m_0^1 = m_0^2$ and $m_1^1 \neq m_1^2$, but since no state is kept and the algorithm is entirely deterministic $\mathsf{Enc}(m_0^1) = \mathsf{Enc}(m_0^2)$. So the adversary could just check if $c_1 = c_2$. ■

5.3.1 Encryption using PRFs

To construct a stateless multi-message secure encryption scheme, we will use a family of PRFs. Consider the following encryption scheme:

Let $f_s : \{0,1\}^n \leftarrow \{0,1\}^n$ be a family of PRFs.

- $\mathsf{KGen}(1^n) := s \xleftarrow{\$} \{0,1\}^n$
- $\mathsf{Enc}(s, m) :=$ Pick $r \xleftarrow{\$} \{0,1\}^n$, Output $(r, c = m \oplus f_s(r))$
- $\mathsf{Dec}(s, (r, c)) := c \oplus f_s(r) = m$

Theorem 15 *Let* $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *be based in PRFs as above, then it is a multi-message secure encryption scheme.*

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

$$\begin{aligned}
H_0 &: \; s \xleftarrow{\$} \mathsf{KGen}(1^n),\ \text{compute } \forall i \in [q(n)], \mathsf{Enc}(s, m_0^i) &&= (r^i, m_0^i \oplus f_s(r^i)) \\
H_1 &: \; s \xleftarrow{\$} \mathsf{KGen}(1^n),\ \text{compute } \forall i \in [q(n)], \mathsf{Enc}(s, m_0^i) &&= (r^i, m_0^i \oplus RF(r^i)) \\
H_2 &: \; s \xleftarrow{\$} \mathsf{KGen}(1^n),\ \text{compute } \forall i \in [q(n)], \mathsf{Enc}(s, m_0^i) &&= (r^i, m_0^i \oplus R \xleftarrow{\$} \{0,1\}^n \\
H_3 &: \; s \xleftarrow{\$} \mathsf{KGen}(1^n),\ \text{compute } \forall i \in [q(n)], \mathsf{Enc}(s, m_1^i) &&= (r^i, m_1^i \oplus R \xleftarrow{\$} \{0,1\}^n \\
H_4 &: \; s \xleftarrow{\$} \mathsf{KGen}(1^n),\ \text{compute } \forall i \in [q(n)], \mathsf{Enc}(s, m_1^i) &&= (r^i, m_1^i \oplus RF(r^i)) \\
H_5 &: \; s \xleftarrow{\$} \mathsf{KGen}(1^n),\ \text{compute } \forall i \in [q(n)], \mathsf{Enc}(s, m_1^i) &&= (r^i, m_1^i \oplus f_s(r^i))
\end{aligned}$$

Using a similar argument to the one used in the PRG case, $H_0 \approx_c H_1$ because a PRF is computationally indistinguishable from a RF. If these two hybrids were distinguishable then we could build an adversary that could distinguish between having oracle access to f_s and RF. Next, note that H_1 and H_2 are statistically indistinguishable: the only difference between them is that in H_2, we might sample the same string R for two different messages; however, this can only happen with exponentially small probability. Now, H_2 and H_3 are indistinguishable by the security of one-time pads. Now, by symmetry the remaining hybrids are indistinguishable giving us our result. ■

Chapter 6

Public-Key Encryption

6.1 Semantic Security

We began lecture by formalizing the definition of Semantic Security for secret-key encryption as follows:

Definition 27 *A secret-key encryption scheme (Gen, Enc, Dec) is semantically secure if there exists a PPT simulator algorithm S s.t. the following two experiments generate computationally indistinguishable outputs:*

$$\left\{ \begin{array}{r} (m, z) \leftarrow M(1^n), \\ s \leftarrow Gen(1^n), \\ Output\ (Enc(s, m), z) \end{array} \right\} \approx \left\{ \begin{array}{l} (m, z) \rightarrow M(1^n), \\ Output\ S(1^n, z) \end{array} \right\}$$

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

Intuitively, semantic security promises that a PPT adversary does not learn any "new" information about a message m by looking at its ciphertext than what it already knew before (denoted as auxiliary information z). This is because adversary's view (consisting of the ciphertext and z) can be efficiently simulated given only z and no other information on m.

We now show that semantic security and IND-CPA security are, in fact, equivalent security notions.

Theorem 16 *Semantic security and IND-CPA security are equivalent.*

Proof. We prove each case separately:

- SS $\rightarrow$ IND: From SS, we have that for any m_1, $(Enc(m_1), z) \approx S(1^n, z)$. Also from SS, for any m_2, we have that $(Enc(m_2), z) \approx S(1^n, z)$. Then, by transitivity, we get IND-CPA security.

- IND-CPA $\rightarrow$ SS: From IND-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., $S(1^n, z) = (Enc(0^n), z)$. SS security immediately follows from IND-CPA security.

6.2 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.

Syntax. A public-key encryption scheme consists of three algorithms:

- $Gen(1^n) \to (pk, sk)$
- $Enc(pk, m) \to c$
- $Dec(sk, c) \to m'$ or $\perp$

Where $Gen(1^n)$ generates a public key pk and a secret key sk. Encryption takes as input a public key pk (which everyone including the adversary will have access to) and a message m and outputs a cipher text. Decryption takes as input a ciphertext c and a secret key sk, and outputs a message or $\perp$.

We say that the encryption scheme is *correct* if $Dec(sk, Enc(pk, m)) = m$, meaning if you encrypt and then decrypt a message with the public and secret key respectively, you should get the original message back.

We now provide our definition of security. We start with a weak definition (also called selective security):

Definition 28 *A public key encryption scheme is weakly-indistinguishably secure under chosen plaintext attack (IND-CPA) if for all n.u. PPT adversaries $\mathcal{A}$ there exists a negligible function $\mu(\cdot)$ such that the probability of*

$$Pr\left[\begin{array}{rl} (pk, sk) \leftarrow Gen(1^n), & \\ (m_0, m_1) \leftarrow \mathcal{A}(1^n), & : \mathcal{A}(pk, Enc(pk, m_b)) = b \\ b \xleftarrow{\$} \{0,1\} & \end{array}\right] \leq \frac{1}{2} + \mu(n)$$

What this means is that we generate two random keys, pk and sk, and allow the Adversary to send us any two messages, m_0 and m_1. The adversary $\mathcal{A}$ will be given the cipher text of a random choice of the two messages and the public key. Given this the adversary will then be unable to distinguish which message was encrypted, i.e., whether, $c = Enc(pk, m_1)$ or $Enc(pk, m_0)$.
We can then adapt this into the definition of strong security by changing a single point, we will give the Adversary the public key before we have it send us messages.

Definition 29 *A public key encryption is indistinguishably secure under chosen plaintext attack (IND-CPA) if for all n.u. PPT adversaries $\mathcal{A}$ there exists a negligible function $\mu(\cdot)$ s.t.:*

$$Pr\left[\begin{array}{rl} (pk, sk) \leftarrow Gen(1^n), & \\ (m_0, m_1) \leftarrow \mathcal{A}(1^n, pk), & : \mathcal{A}(pk, Enc(pk, m_b)) = b \\ b \xleftarrow{\$} \{0,1\} & \end{array}\right] \leq \frac{1}{2} + \mu(n)$$

This is stronger than the weak version because if you give A the public key before then it can attempt to pick messages based on generated cipher texts.

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

Lemma 17 *One-message security implies multi-message security for public-key encryption*

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: This corresponds to the case when $b = 0$, i.e., $c^1 = Enc(pk, m_0^1)$ and $c^2 = Enc(pk, m_0^2)$
- H_1: We now change the first ciphertext to encryption of m_1^1, but keep the second intact. That is, $c^1 = Enc(pk, m_1^1)$ and $c^2 = Enc(pk, m_0^2)$
- H_2: This corresponds to the case when $b = 1$, i.e., $c^1 = Enc(pk, m_1^1)$ and $c^2 = Enc(pk, m_1^2)$

We want to show that H_0 is indistinguishable from H_2, meaning that there is no PPT adversary $\mathcal{A}$ that can correctly predict the challenge bit b except with probability $\frac{1}{2}$. We start by proving that H_0 and H_1 are indistinguishable. The proof of indistinguishability of H_1 and H_2 is analogous and left as an exercise. Combining the two, we obtain that H_0 and H_2 are indistinguishable, as required.

Suppose that there is a PPT distinguisher $\mathcal{A}$ that distinguishes between H_0 and H_1 with noticeable probability. Using $\mathcal{A}$, we will construct an adversary $\mathcal{A}'$ that breaks one-message security of the encryption scheme.

First, $\mathcal{A}'$ receives public key pk from its challenger and forwards it to $\mathcal{A}$. Now, $\mathcal{A}$ sends two message pairs: (m_0^1, m_1^1) and (m_0^2, m_1^2). $\mathcal{A}'$ sets its own challenge message pair to $(m_0 = m_0^1, m_1 = m_1^1)$ and sends it to its challenger. Upon receiving a challenge ciphertext c, it sets $c^1 = c$ and computes $c^2 \leftarrow Enc(pk, m_0^2)$. It sends (c^1, c^2) to $\mathcal{A}$. Whenever $\mathcal{A}$ responds with its guess b', $\mathcal{A}'$ forwards it to its challenger.

Now, 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 $\mathcal{A}$ distinguishes with noticeable probability, then so does $\mathcal{A}'$. This contradicts the one-message security of the encryption scheme. ■

6.3 Trapdoor Permutations

We start by defining a collection of one-way functions and permutations, and then proceed to give a definition of trapdoor permutations.

Definition 30 *A collection of one way functions is a family of functions*

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

that satisfy the following conditions:

- *Sampling Function: There exists a PPT Gen such that $Gen(1^n)$ outputs $i \in I$*
- *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 $i \in D_i$ outputs $f_i(x)$*
- *Hard to invert: For every n.u. PPT adversary $\mathcal{A}$, there exists a negligible function $\mu(\cdot)$ s.t.*

$$Pr[i \leftarrow Gen(1^n),\ x \leftarrow D_i,\ y \leftarrow f_i(x) : f_i(A(1^n, y, i)) = y] \leq \mu(\cdot)$$

Theorem 18 *There exists a collection of one-way functions iff 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 the single function, and therefore collections of one way functions exist. The other direction we must construct a single one-way function given a collection F. To do this define $g(r_1, r_2)$ to be $i, f_i(x)$, where $f_i \in F$, i is random bits generated from 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 $f_i \in F$, which is not possible. ■

This leads us into our next definition, which is collections of One-way permutations.

Definition 31 *A collection*

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

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

Trapdoor Permutations. Next, we formalize the definition of a collection of Trapdoor permutations.

Definition 32 *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: $\exists$ a PPT Gen s.t. $Gen(1^n) \to (i, t) \in I$*
- *Sampling from Domain: : $\exists$ a PPT algorithm that on input i outputs a uniformly random element of D_i*
- *Evaluation: $\exists$ a PPT that on input $i, x \in D_i$ outputs $f_i(x)$*
- *Hard to invert: $\forall$ n.u. PPT adversary $A, \exists$ a negligble function $\mu(\cdot)$ s.t.:*

$$Pr[i \leftarrow Gen(1^n), x \leftarrow D_i, y \leftarrow f_i(x) : f_i(A(1^n, i, y)) = y] \leq \mu(\cdot)$$

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

Roughly speaking, a trapdoor permutation is essentially a one way permutation that has a "trapdoor" t, that allows you to invert. We will now show how to build a public key encryption scheme from a family of one way trapdoor permutations.

6.4 Public-key Encryption from Trapdoor Permutations

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

- $Gen(1^n) : (f_i, f_i^{-1}) \leftarrow Gen_T(1^n)$ Outputs $(pk, sk) \leftarrow ((f_i, h_i), f_i^{-1})$
- $Enc(pk, m)$: Pick $r \xleftarrow{\$} \{0,1\}^n$ Outputs $(f_i(r), h_i(r) \oplus m)$

- $Dec(sk, (c_1, c_2)) : r \leftarrow f^{-1}(c_1)$. Outputs $c_2 \oplus h_i(r)$

Theorem 19 *(Gen, Enc, Dec) is an IND-CPA secure public encryption scheme.*

Proof. Consider the following sequence of hybrids:

- $H_0 : b = 0, c = (f_i(r), h_i(r) \oplus m_0)$
- $H_1 : b = 0, c = (f_i(r), z \oplus m_0)$
- $H_2 : b = 1, c = (f_i(r), z \oplus m_1)$
- $H_3 : b = 1, c = (f_i(r), h_i(r) \oplus m_1)$

where $z \xleftarrow{\$} \{0, 1\}$ is a random bit. Note that in order to prove the IND-CPA security of the encryption scheme, it suffices to prove that H_0 and H_3 are computationally indistinguishable.

The indistinguishability of H_0 and H_1 follows from the security of the one-time pad. Indeed, note that the only difference between H_0 and H_1 is that in H_0, the message m_0 is masked using $h_i(r)$, whereas it is masked with a random bit z in H_1. Then, if there exists a PPT distinguisher between H_0 and H_1, we can use it to construct an adversary for hard-core predicate h_i. The reduction is left as an exercise.

Next, 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. ■

6.5 Trapdoor Permutations from RSA

The rest of the lecture was spent outlining trapdoor permutations from the RSA assumption. We begin by defining the RSA collection.

Definition 33 *$RSA = \{f_i : D_i \to R_i\}$ where:*

- $i = \{(N, e) | N = p * q \text{ s.t. } p, q \in \prod_n, e \in \mathbb{Z}^*_{\Phi(N)}\}$
- $D_i = \{x | x \in \mathbb{Z}^*_N\}$
- $R_i = \mathbb{Z}^*_N$
- $Gen(1^n) \to ((N, e), d)$ *where* $(N, e) \in I$ *and* $e * d \bmod \Phi(N)$
- $f_{N,e}(x) = x^e \bmod N$
- $f^{-1}_{N,e}(y) = y^d \bmod N$

It is easy to verify that that $f_{N,e}$ is a permutation.

Next, we describe the RSA assumption:

Definition 34 *For any n.u. PPT adversary $\mathcal{A}$, there exists a negligible function $\mu(\cdot)$ s.t.:*

$$Pr\left[\begin{array}{c} p, q \leftarrow_\$ \prod_n,\ N = p * q,\ e \leftarrow_\$ \mathbb{Z}^*_{\Phi(N)} \\ y \leftarrow_\$ \mathbb{Z}^*_N; x \leftarrow \mathcal{A}(N, e, y) : \quad x^e = y \bmod N \end{array}\right] \leq \frac{1}{2} + \mu(n)$$

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.*

Chapter 7

Authentication

7.1 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.

7.2 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.

7.2.1 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: $k \leftarrow \text{Gen}(1^n)$, generates a secret key
- Signing: $\sigma \leftarrow \text{Tag}_k(\text{m})$, computes a tag for the message m
- Verification: $\text{Ver}_k(\text{m}, \sigma) = 1$ 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 35 (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: $k \leftarrow \text{Gen}(1^n)$
2. A sends a message m_i to Ch
3. Ch generates a tag: $\sigma_i \leftarrow \text{Tag}_k(m_i)$
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 $\text{Ver}_k(m^*, \sigma^*) = 1$. Unforgeability under Chosen-Message Attack (UF-CMA) security states that the probability that A wins the above game is negligible.

7.3 Construction of MAC

We now construct a MAC scheme based on pseudorandom functions (PRFs).

Theorem 21 *$PRF \Rightarrow MAC$*

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. $\text{Gen}(1^n)$: Output k $\xleftarrow{\$}$ $\{0, 1\}^n$
2. Tag_k(m) : Output $\text{f}_k(m)$
3. Ver_k(m, σ) : Output $\text{f}_k(m) \stackrel{?}{=} \sigma$

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 $\sigma_i = y_i$ 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. ■

7.3.1 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.

7.4 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."

7.4.1 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: $(sk, pk) \leftarrow \text{Gen}(1^n)$
- Signing: $\sigma \leftarrow \text{Sign}_{sk}(m)$, a signature
- Verification: $\text{Ver}_{pk}(m, \sigma)$: $M \times S \rightarrow \{0, 1\}$, where M is the message space and S the signature space

Definition 36 (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 $m \wedge Ver_{pk}(m, \sigma) = 1] \leq \mu(n)$

7.5 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: $\text{Sign}_{sk}(m) : \sigma := (x_1^{m_1}, x_2^{m_2}, \dots, x_n^{m_n})$

  Where $\mid \sigma \mid = n$. Basically, for every column of sk, select one of the two x_i^b depending on m_i.

- Verification: $\text{Ver}_{pk}(m, \sigma) : \wedge_{i \in [n]} f(\sigma_i) \stackrel{?}{=} y_i^{m_i}$.

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 $i \stackrel{\$}{\leftarrow} [n]$, and set $b^* = m_i^*$.
4. Set $y_i^{b^*} = y$. For all $j \neq i, b \in \{0,1\}$, choose an $x_j^b \stackrel{\$}{\leftarrow} \{0,1\}^n$ and calculate $y_j^b = f(x_j^b$. Further, choose $x_i^{1-b^*} \stackrel{\$}{\leftarrow} \{0,1\}^n$ 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).

7.6 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 37 (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 38 (Universal One-Way Hash Function (UOWHF))

$$Pr[(x, state) \leftarrow A(1^n), i \xleftarrow{\$} Gen(1^n), x' \leftarrow A(i, state) : x \neq x' \wedge h_i(x) = h_i(x')] \leq \mu(n)$$

7.7 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 8

Zero-Knowledge Proofs

8.1 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!

8.2 Interactive Protocols

We start by establishing some notation and definitions related to interactive protocols.

Definition 39 (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 40 (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 41 (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 42 (Protocol Output) *The output of M_i in an execution e of (M_1, M_2) is denoted as $Out_{M_i}(e)$*

Definition 43 (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 $View_{M_i}(e)$.*

8.3 Interactive Proofs

Interactive proofs involve a pair of ITMs P and V, where P denotes the prover and V denotes the verifier.

Definition 44 (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 1 *In this definition, the prover does not have to be efficient. The restriction of efficient provers will be visited later.*

Remark 2 *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.

8.3.1 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.

8.4 Notation for Graphs

Definition 45 (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 46 Π_n *is the set of all permutations* π *over n vertices.*

Definition 47 (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 3 *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 48 (Graph Non-Isomorphism) G_0 *and* G_1 *are non-isomorphic if there exists no permutation* $\pi \in \Pi_n$ *s.t.* $G_1 = \pi(G_0)$

8.5 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 $H = \pi(G_b)$		
	send H →	
		Compute b' s.t. H and $G_{b'}$ are isomorphic
	← 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.

8.6 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 49 *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 4 *Even though honest P is efficient, we still require soundness guarantee against all possible adversarial provers.*

8.6.1 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.

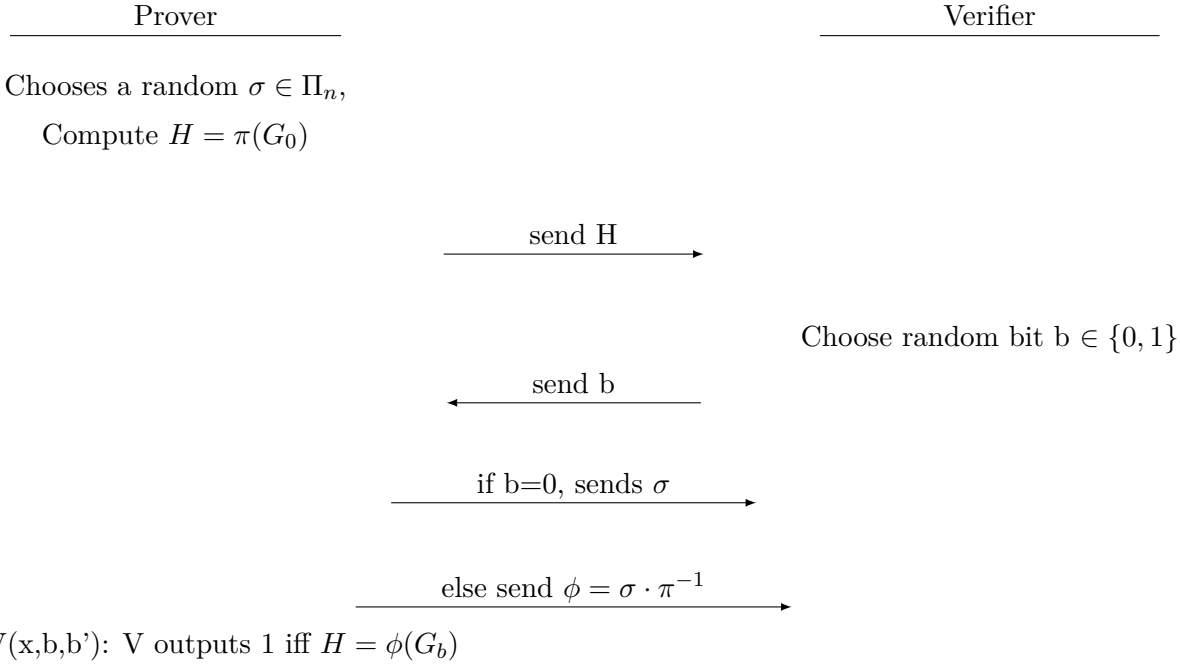
This raises the natural question whether there is a way to interactively prove isomorphism without revealing the witness. Even better yet, can we construct construct a proof that reveals nothing to V beyond the validity of the statement?

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.



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.

8.7 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 50 (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 5 *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 51 (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 6 *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 22 *Sequential repetition of any protocol is also .*

A proof sketch, which is skipped here, can be seen in section 7.2.1 of [**?**].

To prove a single iteration of the interactive proof is perfect , we need to perform the following steps:

- Construct a simulator for every PPT V^*.
- Prove that the expected run time of is polynomial.
- Prove that the output distribution of is indistinguishable from the real execution.

The simulator is defined below,

(x, z)
$b' \xleftarrow{\$} \{0,1\}, \sigma \xleftarrow{\$} \Pi_n$
$H = \sigma(G_{b'})$
Emulate execution of $V^*(x,z)$ by feeding it H. Let b be its response.
If $b = b'$
Feed σ 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 23 *In the execution of* (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 $\sigma \stackrel{\$}{\leftarrow} \Pi_n$, 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 has a success probability of $\frac{1}{2}$. Thus the expected number of iterations before 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 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 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. ■

8.8 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, is about transcripts while soundness is about “live" executions because of the random challenges.

8.9 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 . The formal statement is,

Theorem 24 *If one-way permutations exist, then every language in* NP *has a interactive proof.*

Remark 7 *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 proof for each language in NP? That would be ridiculously inefficient. Instead we focus on NP-complete languages, and rely on their ‘completeness’ property.

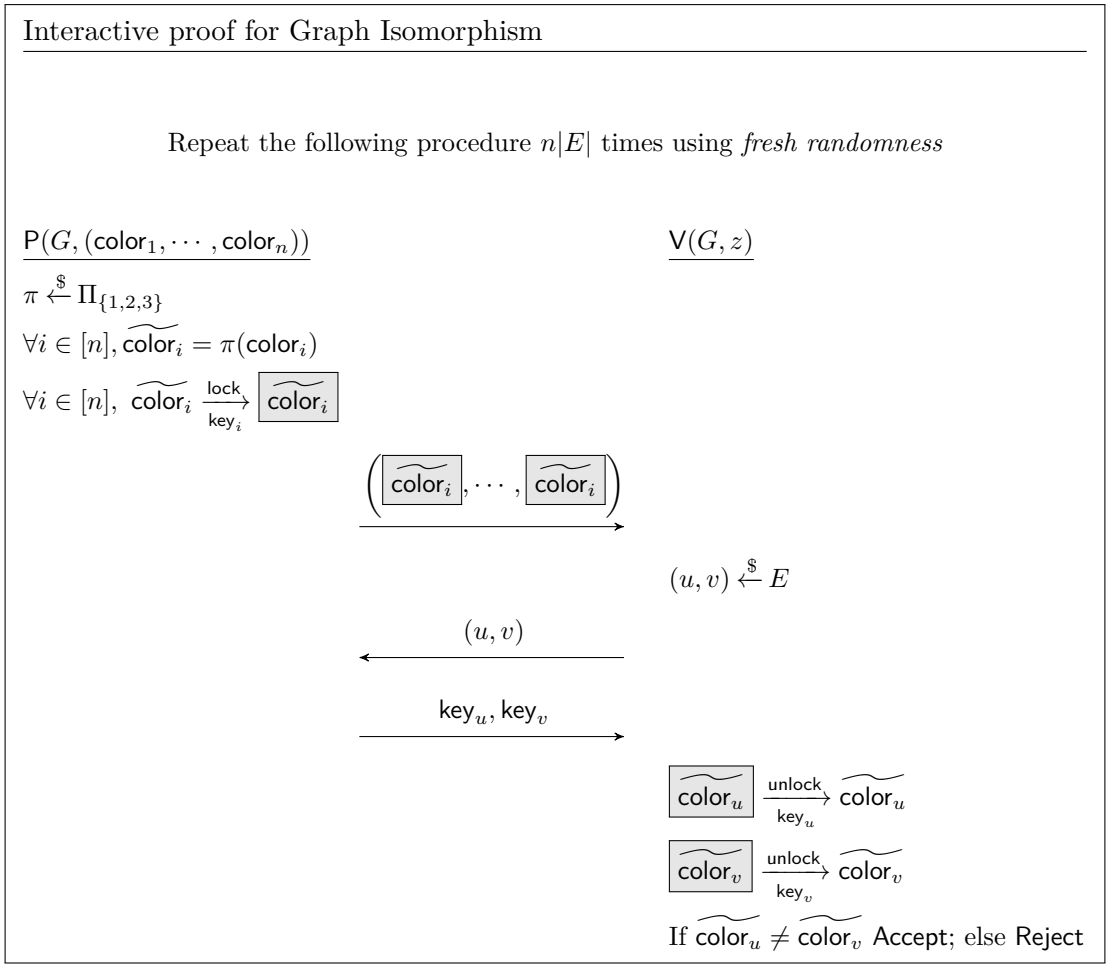
Proof. The proof proceeds in two steps:

Step 1: Construct a 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 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 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 $\boxed{\mathsf{color}_i}$ 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,



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 , 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.

8.10 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 52 (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*

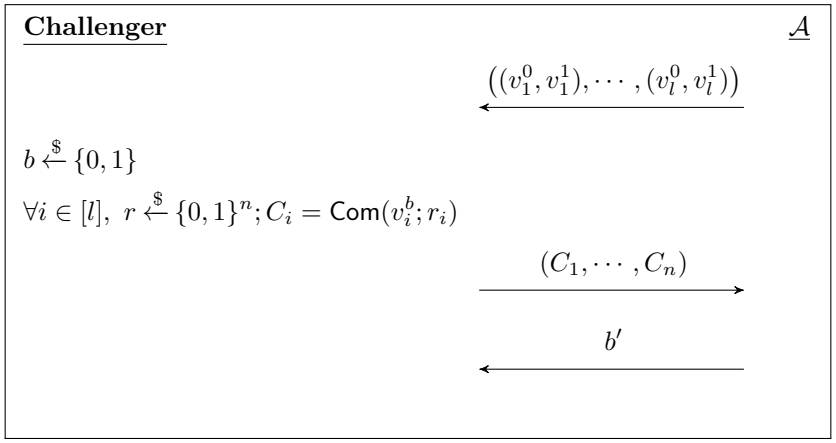
$$\mathsf{Com}(v_0; r_0) \neq \mathsf{Com}(v_1; r_1).$$

- **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 $\nu(n)$*

$$\left\{r \xleftarrow{\$} \{0,1\}^n : \mathsf{Com}(v_0; r)\right\} \text{ and } \left\{r \xleftarrow{\$} \{0,1\}^n : \mathsf{Com}(v_1; r)\right\}.$$

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.



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 25 *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 proof for *Graph 3-coloring*.

8.11 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*

$\mathsf{P}(G, (\mathsf{color}_1, \cdots, \mathsf{color}_n))$		$\mathsf{V}(G, z)$
$\pi \xleftarrow{\$} \Pi_{\{1,2,3\}}$		
$\forall i \in [n], \widetilde{\mathsf{color}_i} = \pi(\mathsf{color}_i)$		
$\forall i \in [n],\ C_i = \mathsf{Com}(\widetilde{\mathsf{color}_i};\ r_i)$		
	$\xrightarrow{(C_1, \cdots, C_n)}$	
		$(u, v) \xleftarrow{\$} E$
	$\xleftarrow{(u, v)}$	
	$\xrightarrow{(\widetilde{\mathsf{color}_u};\ r_u), (\widetilde{\mathsf{color}_v};\ r_v)}$	
		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 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 9

Secure Computation

9.1 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)?

9.2 Adversary Models

There are two types of adversaries:

Definition 53 (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 54 (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.

9.3 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 55 (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.

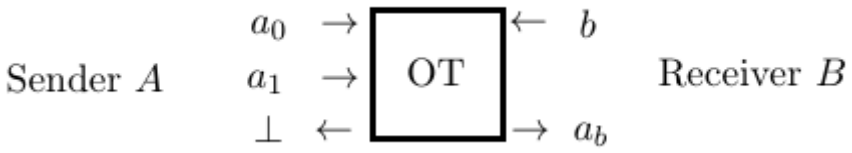
Remarks on Definition. Recall that in zero-knowledge, we only require indistinguishability of simulated view and adversary's view in the real execution. Here, indistinguishability is with respect to the joint distribution over the adversary's view and the honest party's output.

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.

9.4 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.



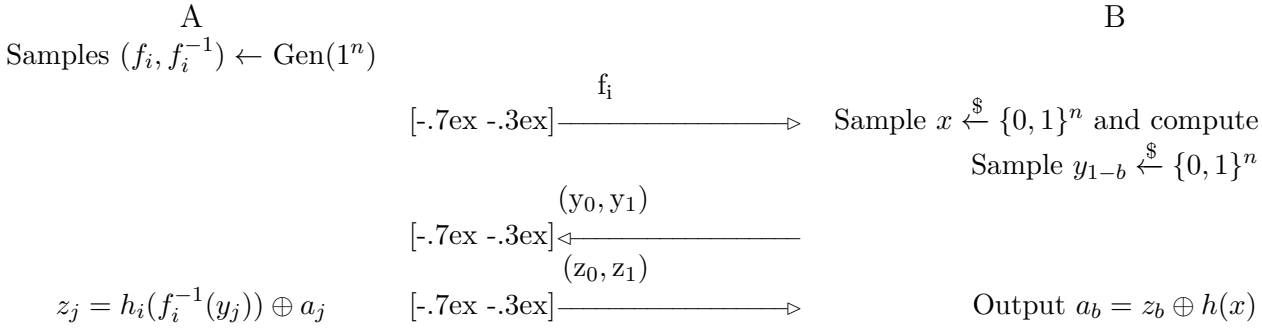
We define security for OT using Definition 55. Note that the definition promises that in a secure OT protocol, A does not learn b and B does not learn a_{1-b}.

9.5 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.

9.5.1 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.



9.6 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 8 (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 26 *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 27 *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 ■

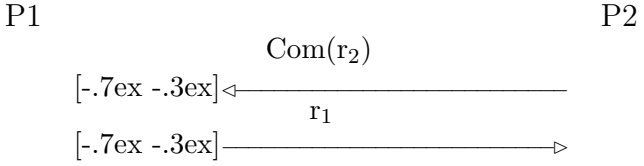
9.7 Remarks

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.

9.8 Goldreich- Micali-Wigderson (GMW) Protocol

9.8.1 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 $f(x, y)$.

9.8.2 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 56 (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

9.8.3 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. $\mathsf{Reconstruct}(w^A, w^B) = w$

9.8.4 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 $\mathsf{CircuitEval}$ 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

   $$\mathsf{out}_i = \mathsf{Reconstruct}(\mathsf{out}_i^A, \mathsf{out}_i^B) \tag{9.1}$$

   The output is $\mathsf{out} = \mathsf{out}_1, ..., \mathsf{out}_n$

9.8.5 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:
to 0.8 | X[2 cm] | X[2 cm] | X[8 cm] |

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 $b = 1 + 2u^B + v^B$

9.8.6 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.

9.9 Yao's Garbled Circuits

Definition 57 *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*
$$\widehat{G} = \{\widehat{g}_1, \ ... \ , \widehat{g}_{|c|}\}$$
$$\widehat{In} = \{\widehat{in}_1, \ ... \ , \widehat{in}_n\}$$
- *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 $(k_{x_1}^1, \ ... \ , k_{x_n}^n)$ 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 58 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* $m \ \in \{0,1\}^n$,

$$Pr[k \leftarrow Gen(1^n), k' \leftarrow Gen(1^n), Dec_k(Enc_k(m)) = \perp] \ < \ 1 - \nu(n)$$

Essentially this is saying if a ciphertext is decrypted using a different or "wrong" key, then answer is always $\perp$
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 $\perp$

9.9.1 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 $1, ..., n$.

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 $\widehat{In} = (\widehat{in}_1, \ ... \ , \widehat{in}_n)$
- For every gate g in C with input wire (i

First Input	Second Input	Output
k_0^i	k_0^j	$z_1 = Enc_{k_0^i}(Enc_{k_0^j}(k_{g(0,0)}^l))$
k_0^i	k_1^j	$z_2 = Enc_{k_0^i}(Enc_{k_1^j}(k_{g(0,1)}^l))$
k_1^i	k_0^j	$z_3 = Enc_{k_1^i}(Enc_{k_0^j}(k_{g(1,0)}^l))$
k_1^i	k_1^j	$z_4 = Enc_{k_1^i}(Enc_{k_1^j}(k_{g(1,1)}^l))$

Set $\widehat{g} = \text{RandomShuffle}(z_1, z_2, z_3, z_4)$. Output $(\widehat{G} = (\widehat{g}_1, \ ... \ , \widehat{g}_{|C|}), \widehat{In})$

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 $\widehat{G} = (\widehat{g}_1, \ ... \ , \widehat{g}_{|C|})$. $\widehat{In}_x = (k^1, \ ... \ , k^n)$
- Parse $\widehat{g}_i = (\widehat{g}_1, \ ... \ , \widehat{g}_4)$
- 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 $\exists \alpha_p \neq \perp$, set $k^l = \alpha_p$
- Let out_i be the value obtained for each output wire i. Output $out = (out_1, \ ... \ , out_n)$

9.9.2 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: $x = x_1, \ ... \ , x_n$

B's input: $y = y_1, \ ... \ , y_n$

Protocol $Pi = (A, B)$**:**

$A \rightarrow B$	A computes $(\widehat{G}, \widehat{In})$ Parse $\widehat{In} = (\widehat{in}_1, \ ... \ , \widehat{in}_{2n})$ where $\widehat{in}_i = (k_0^i, k_1^i)$. Set $\widehat{In}_x = (k_{x_1}, \ ... \ , k_{x_n}^n)$. 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 $\widehat{In}_y = (k_{y_1}^{n+1}, \ ... \ , k_{y_n}^{2n})$ 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 10

Non-Interactive Zero Knowledge

10.1 Introduction

So far we have discussed the case of interactive 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 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,

10.2 Non-Interactive Proofs

Definition 59 *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 60 (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 if there exists a PPT simulator s.t. for every* $x \in L, w \in R(x)$*, the output distribution of the following two experiments are computationally indistinguishable:*

$\mathsf{REAL}(1^n, x, w)$	$\mathsf{IDEAL}(1^n, x)$
$\sigma \leftarrow \mathsf{K}(1^n)$	$(\sigma, \pi) \leftarrow (1^n, x)$
$\pi \leftarrow \mathsf{P}(\sigma, x, w)$	
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 61 (Adaptive NIZK) *A non-interactive proof system* $(\mathsf{K}, \mathsf{P}, \mathsf{V})$ *for a language* L *with witness relation* R *is adaptive if there exists a PPT simulator* $= (_0, _1)$ *s.t. for every* $x \in L, w \in R(x)$*, the output distribution of the following two experiments are computationally indistinguishable:*

$\mathsf{REAL}(1^n, x, w)$	$\mathsf{IDEAL}(1^n, x)$
$\sigma \leftarrow \mathsf{K}(1^n)$	$(\sigma, \tau) \leftarrow {}_0(1^n)$
$\pi \leftarrow \mathsf{P}(\sigma, x, w)$	$\pi \leftarrow {}_1(\sigma, \tau, x)$
Output (σ, π)	*Output* (σ, π)

Here τ is the "trapdoor" for the simulated common random string σ that is used by $_1$ 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) , we don't define the 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 28 *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.*

10.3 NIZKs for NP

I. Non-adaptive Zero Knowledge: We first construct NIZKs for **NP** with non-adaptive 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.

10.4 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 62 *A non-interactive proof system for a language* L *with witness relation* R *in the hidden-bit model is a tuple of algorithms* $(\mathsf{K}_{\mathsf{HB}}, \mathsf{P}_{\mathsf{HB}}, \mathsf{V}_{\mathsf{HB}})$ *such that:*

1. **Setup:** $r \leftarrow \mathsf{K}(1^n)$ *outputs a common random string.*
2. **Prove:** $\pi \leftarrow \mathsf{P}_{\mathsf{HB}}(r, x, w)$ *generates the indices* $I \subseteq [|r|]$ *of* r *to reveal along with a proof* π*.*
3. **Verify:** $\mathsf{V}_{\mathsf{HB}}(I, \{r_i\}_{i \in I}, x, \pi)$ *outputs 1 if it accepts the proof and 0 otherwise.*

We define below. A non-interactive proof system must satisfy completeness and soundness properties defined earlier.

Definition 63 (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) if there exists a PPT simulator* HB *s.t. for every* $x \in L, w \in R(x)$*, the output distribution of the following two experiments are computationally indistinguishable:*

$\mathsf{REAL}(1^n, x, w)$	$\mathsf{IDEAL}(1^n, x)$
$r \leftarrow \mathsf{K_{HB}}(1^n)$	$(I, \{r_i\}_{i\in I}, \pi) \leftarrow \mathsf{HB}(1^n, x)$
$(I, \pi) \leftarrow \mathsf{P_{HB}}(r, x, w)$	
Output $(I, \{r_i\}_{i\in I}, \pi)$	*Output* $(I, \{r_i\}_{i\in I}, \pi)$

10.5 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 $(\mathsf{K_{HB}}, \mathsf{P_{HB}}, \mathsf{V_{HB}})$ 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.

$\mathsf{K}(1^n)$

1 : for every $i \in [n]$
2 : $\quad \sigma_i \xleftarrow{\$} \{0,1\}^n$
3 : Output $\sigma = \sigma_1\sigma_2 \cdots \sigma_n$

$\mathsf{P}(\sigma, x, w)$

1 : $(f, f^{-1}) \xleftarrow{\$} \mathcal{F}(1^n)$
2 : for every $i \in [n]$
3 : $\quad \alpha_i = f^{-1}(\sigma_i)$
4 : for every $i \in [n]$
5 : $\quad r_i = h(\alpha_i)$
6 : $(I, \Phi) \leftarrow \mathsf{P_{HB}}(r, x, w)$
7 : Output $\pi = (\sigma, f, I, \{\alpha_i\}_{i\in I}, \Phi)$

$\mathsf{V}(\sigma, x, \pi)$

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 : $\quad f(\alpha_i) = \sigma_i$
5 : for every $i \in I$
6 : $\quad r_i = h(\alpha_i)$
7 : Output $\mathsf{V}_{\mathsf{HB}}(I, \{\alpha_i\}_{i\in I}, x, \Phi)$

Theorem 29 *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

$$\mathsf{Pr}[\sigma \leftarrow K(1^n) : \mathsf{P}^* \text{ can cheat using } f_0] \leq \frac{1}{2^{2n}}$$

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 HB be the simulator for $(\mathsf{K}_{\mathsf{HB}}, \mathsf{P}_{\mathsf{HB}}, \mathsf{V}_{\mathsf{HB}})$. The simulator is,

$(1^n, x)$

1 : $(I, \{r_i\}_{i\in I}, \Phi) \xleftarrow{\$} \mathsf{HB}(1^n, x)$
2 : $(f, f^{-1}) \xleftarrow{\$} \mathcal{F}$
3 : for every $i \in I$
4 : $\quad \alpha_i = h^{-1}(r_i)$
5 : for every $i \in I$
6 : $\quad \sigma_i = f(\alpha_i)$
7 : for every $i \notin I$
8 : $\quad \sigma_i \xleftarrow{\$} \{0,1\}^n$
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 , 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 : $\sigma \xleftarrow{\$} K(1^n)$ where $\sigma = \sigma_1, \cdots, \sigma_n$
2 : $(f, f^{-1}) \xleftarrow{\$} \mathcal{F}$
3 : for every $i \in [n]$
4 : $\quad \alpha_i = f^{-1}(\sigma_i)$
5 : for every $i \in [n]$
6 : $\quad r_i = h(\alpha_i)$
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 : $\boxed{\alpha_i \xleftarrow{\$} \{0,1\}^n \text{ for every } i \in [n]}$
2 : $(f, f^{-1}) \xleftarrow{\$} \mathcal{F}$
3 : for every $i \in [n]$
4 : $\quad \boxed{\sigma_i = f(\alpha_i)}$
5 : for every $i \in [n]$
6 : $\quad r_i = h(\alpha_i)$
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 : $\boxed{r_i \xleftarrow{\$} \{0,1\}^n \text{ for every } i \in [n]}$
2 : $(f, f^{-1}) \xleftarrow{\$} \mathcal{F}$
3 : for every $i \in [n]$
4 : $\quad \boxed{\alpha_i = h^{-1}(r_i)}$
5 : for every $i \in [n]$
6 : $\quad \sigma_i = f(\alpha_i)$
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 : $r_i \xleftarrow{\$} \{0,1\}^n \text{ for every } i \in [n]$
2 : $(f, f^{-1}) \xleftarrow{\$} \mathcal{F}$
3 : for every $i \in [n]$
4 : $\quad \alpha_i = h^{-1}(r_i)$
5 : $(I, \Phi) \leftarrow \mathsf{P}_{\mathsf{HB}}(r, x, w)$
6 : $\boxed{\text{for every } i \in I}$
7 : $\quad \boxed{\sigma_i = f(\alpha_i)}$
8 : $\boxed{\text{for every } i \notin I}$
9 : $\quad \boxed{\sigma_i \xleftarrow{\$} \{0,1\}^n}$
10 : Output $(\sigma, f, I, \{\alpha_i\}_{i \in I}, \Phi)$

$H_4(1^n, x) = \mathsf{IDEAL}(1^n, x)$

1 : $\boxed{(I, \{r_i\}_{i \in I}, \Phi) \xleftarrow{\$} \mathsf{HB}(1^n, x)}$
2 : $(f, f^{-1}) \xleftarrow{\$} \mathcal{F}$
3 : $\boxed{\text{for every } i \in I}$
4 : $\quad \alpha_i = h^{-1}(r_i)$
5 : for every $i \in I$
6 : $\quad \sigma_i = f(\alpha_i)$
7 : for every $i \notin I$
8 : $\quad \sigma_i \xleftarrow{\$} \{0,1\}^n$
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 $\mathbf{_{HB}}$. Indistinguishability follows from the property of $(\mathsf{K_{HB}}, \mathsf{P_{HB}}, \mathsf{V_{HB}})$.

Thus $H_0 \approx H_4$. This gives us the proof. ■

Next we shall construct NIZKs for all languages in **NP** in the hidden-bit model.

10.6 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 64 (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 65 (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:*

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

Definition 66 (Cycle Matrix) *A cycle matrix is a boolean matrix that corresponds to a graph that contains a Hamiltonian cycle and no other edges.*

Definition 67 (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 $\frac{1}{n}$.

10.7 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.

10.7.1 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.

10.7.2 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 30 *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:

$$Pr[|X - n| > n] \leq \frac{1}{n}$$

- 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:

$$Pr[X = n] \geq \frac{\sum_{i=1}^{2n} Pr[X = i]}{2n}$$

$$\geq \frac{1}{3n}$$

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,

$$Pr[\text{new '1' entry is in different row and column}] = \left(1 - \frac{k}{n^2}\right)^2$$

- Multiplying all:

$$Pr[\text{no collision}] \geq \left(1 - \frac{1}{n^2}\right)^2 \cdots \left(1 - \frac{n-1}{n^2}\right)^2$$

$$\geq \frac{1}{n}$$

- Combining the above with Claim 1:

$$Pr[\text{M contains a permutation } n \times n \text{ submatrix}] \geq \frac{1}{3n^2}$$

Proof of Claim 3 We want to bound the probability that M contains an $n \times n$ cycle submatrix

- Observe:

$$Pr[n \times n \text{ permutation matrix is a cycle matrix }] = \frac{1}{n}$$

- Combining the above with Claim 2,

$$Pr[M \text{ contains a cycle } n \times n \text{ submatrix } \geq \frac{1}{3n^3}$$

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 $(M^i, M^i_C) \leftarrow Q(r_i)$. Compute$(I'_i, \varphi_i) \leftarrow P_1(M^i_C, x, w)$. 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^i_C, 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 $s_i = s^1_i, s^2_i$ and check that s^2_i is the all 0 string. Also, check that $V_1(I'_i, s^1_i, \pi_i) = 1$.

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 11

CCA Security

11.1 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 68 *(IND-CCA-1 Security) A public key encryption scheme $(,,)$ 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 $z \in \{0,1\}^*$:*

$$|Pr[\boldsymbol{Expt}_{\mathcal{A}}^{CCA1}(1,z)=1] - Pr[\boldsymbol{Expt}_{\mathcal{A}}^{CCA1}(0,z)=1]| \leq \mu(n)$$

where $\boldsymbol{Expt}_{\mathcal{A}}^{CCA1}(b,z)$ is defined as:
$\boldsymbol{Expt}_{\mathcal{A}}^{CCA1}(0,z)$

- $st = z$
- $(pk, sk) \leftarrow (1^n)$
- *Decryption query phase (repeated polynomial times)*
  - $c \leftarrow \mathcal{A}(pk, st)$
  - $m \leftarrow (sk, c)$
  - $st = (st, m)$
- $(m_0, m_1) \leftarrow \mathcal{A}(pk, st)$
- $c^* \leftarrow (pk, m_b)$
- *Output* $b' \leftarrow \mathcal{A}(pk, c^*, st)$

Definition 69 *(IND-CCA-2 Security) A public key encryption scheme $(,,)$ 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 $z \in \{0,1\}^*$:*

$$|Pr[\boldsymbol{Expt}_{\mathcal{A}}^{CCA2}(1,z)=1] - Pr[\boldsymbol{Expt}_{\mathcal{A}}^{CCA2}(0,z)=1]| \leq \nu(n)$$

where $\boldsymbol{Expt}_{\mathcal{A}}^{CCA2}(b,z)$ is defined as:
$\boldsymbol{Expt}_{\mathcal{A}}^{CCA2}(0,z)$

- $st = z$
- $(pk, sk) \leftarrow (1^n)$
- *Decryption query phase 1 (repeated polynomial times)*
  - $c \leftarrow \mathcal{A}(pk, st)$
  - $m \leftarrow (sk, c)$
  - $st = (st, m)$
- $(m_0, m_1) \leftarrow \mathcal{A}(pk, st)$
- $c^* \leftarrow (pk, m_b)$
- *Decryption query phase 2 (repeated polynomial times)*
  - $c \leftarrow \mathcal{A}(pk, c^*, st)$
  - *If* $c = c^*$*, output reject*
  - $m \leftarrow (sk, c)$
  - $st = (st, m)$
- *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^*.

11.2 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 31 *(Naor-Yung) Assuming that NIZKs in the CRS model and IND-CPA secure public-key encryption, the encryption scheme* $(',',')$ *below is IND-CCA-1 secure public-key encryption.*

Let $(,,)$ be an IND-CPA encryption scheme.

Let $(\mathsf{K}, \mathsf{P}, \mathsf{V})$ be an adaptive NIZK with Simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$.

$'(1^n)$:

- Compute (pk_1, sk_1) and (pk_2, sk_2) using (1^n)

- Compute $\sigma \leftarrow \mathsf{K}(1^n)$

- Output $pk' = (pk_1, pk_2, \sigma), sk' = sk_1$

$'(pk', m)$:

- Compute $c_i \leftarrow (pk_i, m; r_i)$ 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 $C =)c_1, c_2, \pi$

$'(sk', c')$: If $\mathsf{V}(\sigma, \pi) = 0$, output $\perp$. Else, output (sk_1, c_1).

Proof. We use Hybrid Lemma to prove the theorem. We construct hybrids as follows:

Hybrids H_0: $= \mathbf{Expt}^{CCA1}_{\mathcal{A}}(0, z)$

- $(pk_i, sk_i) \leftarrow (1^n)$ 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 $(sk' = sk_1, c_1)$
- $(m_0, m_1) \leftarrow \mathcal{A}(z, pk')$
- $c_1^* \leftarrow (pk_1, m_0; r_1^*)$
- $c_2^* \leftarrow (pk_2, m_0; r_2^*)$
- $\pi^* \leftarrow \mathsf{P}(\sigma, x^* = (c_1^*, c_2^*), w^* = (m_0, r_1, r_2))$
- Output $\mathcal{A}(z, pk', C = (c_1^*, c_2^*, \pi^*))$

Hybrids H_1:

- $(pk_i, sk_i) \leftarrow (1^n)$ 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 $(sk' = sk_1, c_1)$
- $(m_0, m_1) \leftarrow \mathcal{A}(z, pk')$
- $c_1^* \leftarrow (pk_1, m_0; r_1^*)$
- $c_2^* \leftarrow (pk_2, m_0; r_2^*)$
- $\pi^* \leftarrow \mathcal{S}_1(\sigma, \tau, x^* = (c_1^*, c_2^*))$
- Output $\mathcal{A}(z, pk', C = (c_1^*, c_2^*, \pi^*))$

Hybrids H_2:

- $(pk_i, sk_i) \leftarrow (1^n)$ 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 $(sk' = sk_1, c_1)$
- $(m_0, m_1) \leftarrow \mathcal{A}(z, pk')$
- $c_1^* \leftarrow (pk_1, m_0; r_1^*)$
- $c_2^* \leftarrow (pk_2, m_1; r_2^*)$
- $\pi^* \leftarrow \mathcal{S}_1(\sigma, \tau, x^* = (c_1^*, c_2^*))$
- Output $\mathcal{A}(z, pk', C = (c_1^*, c_2^*, \pi^*))$

Hybrids H_3:

- $(pk_i, sk_i) \leftarrow (1^n)$ 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 $(sk' = sk_2, c_2)$
- $(m_0, m_1) \leftarrow \mathcal{A}(z, pk')$
- $c_1^* \leftarrow (pk_1, m_0; r_1^*)$
- $c_2^* \leftarrow (pk_2, m_1; r_2^*)$
- $\pi^* \leftarrow \mathcal{S}_1(\sigma, \tau, x^* = (c_1^*, c_2^*))$
- Output $\mathcal{A}(z, pk', C = (c_1^*, c_2^*, \pi^*))$

Hybrids H_4:

- $(pk_i, sk_i) \leftarrow (1^n)$ 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 $(sk' = sk_2, c_2)$
- $(m_0, m_1) \leftarrow \mathcal{A}(z, pk')$
- $c_1^* \leftarrow (pk_1, m_1; r_1^*)$
- $c_2^* \leftarrow (pk_2, m_1; r_2^*)$
- $\pi^* \leftarrow \mathcal{S}_1(\sigma, \tau, x^* = (c_1^*, c_2^*))$
- Output $\mathcal{A}(z, pk', C = (c_1^*, c_2^*, \pi^*))$

Hybrids H_5:

- $(pk_i, sk_i) \leftarrow (1^n)$ 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 $(sk' = sk_1, c_1)$
- $(m_0, m_1) \leftarrow \mathcal{A}(z, pk')$
- $c_1^* \leftarrow (pk_1, m_1; r_1^*)$
- $c_2^* \leftarrow (pk_2, m_1; r_2^*)$
- $\pi^* \leftarrow \mathcal{S}_1(\sigma, \tau, x^* = (c_1^*, c_2^*))$
- Output $\mathcal{A}(z, pk', C = (c_1^*, c_2^*, \pi^*))$

Hybrids H_6: $= \mathbf{Expt}_{\mathcal{A}}^{CCA1}(0, z)$

- $(pk_i, sk_i) \leftarrow (1^n)$ 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 $(sk' = sk_1, c_1)$
- $(m_0, m_1) \leftarrow \mathcal{A}(z, pk')$
- $c_1^* \leftarrow (pk_1, m_1; r_1^*)$
- $c_2^* \leftarrow (pk_2, m_1; r_2^*)$
- $\pi^* \leftarrow \mathsf{P}(\sigma, x^* = (c_1^*, c_2^*), w^* = (m_1, r_1, r_2))$
- Output $\mathcal{A}(z, pk', C = (c_1^*, c_2^*, \pi^*))$

In short, the changes in hybrids are:

- H_0 : $\mathrm{Expt}_{\mathcal{A}}^{CCA1}(1, z)$.
- 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.

- $H_6 : \text{Expt}_{\mathcal{A}}^{CCA1}(0, z)$:

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:
$\mathcal{D}(\sigma, \pi)$

- $(pk_i, sk_i) \leftarrow (1^n)$ 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 $(sk' = sk_1, c_1)$
- $(m_0, m_1) \leftarrow \mathcal{A}(z, pk')$
- $c_1^* \leftarrow (pk_1, m_0; r_1^*)$
- $c_2^* \leftarrow (pk_2, m_0; r_2^*)$
- 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 $(,,)$ 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:

- $(pk_1, sk_1) \leftarrow (1^n)$.
- 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 $(sk' = sk_1, c_1)$
- Run $\mathcal{A}$ to get message query (m_0, m_1) and pass (m_0, m_1) to challenger.
- $c_1^* \leftarrow (pk_1, m_0; r_1^*)$
- 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:

- $(pk_i, sk_i) \leftarrow (1^n)$ 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 $(sk' = sk_1, c_1) \neq (sk' = sk_2, c_2)$ 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 $',',',$.

11.3 IND-CCA-2 Security

We begin by defining the challenge experiment $\mathbf{Expt}_{\mathcal{A}}^{\mathsf{CCA2}}(b, z)$ for an adversary in the CCA-2 Security model.

$\boxed{\mathbf{Expt}_{\mathcal{A}}^{\mathsf{CCA2}}(b, z)}$:

- $\mathsf{st} = z$
- $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$
- Decryption query phase 1 (repeated poly times):
  - $c \leftarrow \mathcal{A}(pk, \mathsf{st})$
  - $m \leftarrow \mathsf{Dec}(sk, c)$
  - $\mathsf{st} = (\mathsf{st}, m)$
- $(m_0, m_1) \leftarrow \mathcal{A}(pk, \mathsf{st})$
- $c^* \leftarrow \mathsf{Enc}(pk, m_b)$
- Decryption query phase 2 (repeated poly times):
  - $c \leftarrow \mathcal{A}(pk, c^* \ \mathsf{st})$
  - If $c = c^*$, output reject.
  - $m \leftarrow \mathsf{Dec}(sk, c)$
  - $\mathsf{st} = (\mathsf{st}, m)$
- Output $b' \leftarrow \mathcal{A}(pk, c^*, \mathsf{st})$

Definition 70 ***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.

11.4 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.

11.4.1 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.

11.4.2 Security

Theorem 32 *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 $\mathcal{A}(pk', c^*, z)$

$\mathbf{H_3}$:

- $(\sigma, \tau) \leftarrow \mathcal{S}_0(1^n)$
- $(SK^*, VK^*) \leftarrow \mathsf{Setup}(1^n)$, $VK^* = VK_1^*, ..., VK_n^*$
- $(pk_i^j, sk_i^j) \leftarrow \mathsf{Gen}(1^n)$ 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 $\mathsf{Dec}(sk_\ell^{\overline{VK_\ell^*}}, c_\ell)$
- $(m_0, m_1) \leftarrow \mathcal{A}(pk, z)$

- $c_i^* \leftarrow \mathsf{Enc}(pk_i^{VK_i^*}, m_0; r_i^*)$
- $\pi^* \leftarrow \mathcal{S}_1(\sigma, \tau, x^* = (\{pk_i^{VK_i^*}\}, \{c_i^*\}), w^* = (m_0, \{r_i^*\}))$
- $\Phi^* \leftarrow \mathsf{Sign}(SK^*, M* = (\{c_i\}, \pi))$
- $c^* = (VK^*, \{c_i^*\}, \pi^*, \Phi^*)$
- 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 $\mathcal{A}(pk', c^*, z)$

$\mathbf{H_4}$: 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 $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$

- $\mathbf{H_4} \approx \mathbf{H_5}$: 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. ■