SANJAM GARG

A COURSE IN THEORY OF CRYPTOGRAPHY

THIS DOCUMENT IS CONTINUALLY BEING UPDATED. PLEASE SEND US YOUR FEEDBACK.

This draft was compiled on Thursday 17th October, 2024.

Contents

Preface

Cryptography enables many paradoxical objects, such as public key encryption, verifiable electronic signatures, zero-knowledge protocols, and fully homomorphic encryption. The two main steps in developing such seemingly impossible primitives are (i) defining the desired security properties formally and (ii) obtaining a construction satisfying the security property provably. In modern cryptography, the second step typically assumes (unproven) computational assumptions, which are conjectured to be computationally intractable. In this course, we will define several cryptographic primitives and argue their security based on well-studied computational hardness assumptions. However, we will largely ignore the mathematics underlying the assumed computational intractability assumptions.

Acknowledgements

These lecture notes are based on scribe notes taken by students in CS 276 over the years. Also, thanks to Peihan Miao, Akshayaram Srinivasan, and Bhaskar Roberts for helping to improve these notes.

1
Mathematical Background

In modern cryptography, (1) we typically assume that our attackers cannot run in unreasonably large amounts of time, and (2) we allow security to be broken with a *very small*, but non-zero, probability.

Without these assumptions, we must work in the realm of information-theoretic cryptography, which is often unachievable or impractical for many applications. For example, the one-time pad [1] – an information-theoretically secure cipher – is not very useful because it requires very large keys.

In this chapter, we define items (1) and (2) more formally. We require our adversaries to run in polynomial time, which captures the idea that their runtime is not unreasonably large (sections 1.1). We also allow security to be broken with negligible – very small – probability (section 1.2).

1.1 *Probabilistic Polynomial Time*

A probabilistic Turing Machine is a generic computer that is allowed to make random choices during its execution. A probabilistic *polynomial time* Turing Machine is one which halts in time polynomial in its input length. More formally:

Definition 1.1 (Probabilistic Polynomial Time). *A probabilistic Turing Machine M is said to be PPT (a Probabilistic Polynomial Time Turing Machine) if $\exists c \in \mathbb{Z}^+$ such that $\forall x \in \{0,1\}^*$, $M(x)$ halts in $|x|^c$ steps.*

A *non-uniform* PPT Turing Machine is a collection of machines one for each input length, as opposed to a single machine that must work for all input lengths.

Definition 1.2 (Non-uniform PPT). *A non-uniform PPT machine is a sequence of Turing Machines $\{M_1, M_2, \cdots\}$ such that $\exists c \in \mathbb{Z}^+$ such that $\forall x \in \{0,1\}^*$, $M_{|x|}(x)$ halts in $|x|^c$ steps.*

[1] For a message $m \in \{0,1\}^n$ and a random key $k \in \{0,1\}^n$, the encryption of m is $c = m \oplus k$. The decryption is $m = c \oplus k$.

1.2 Noticeable and Negligible Functions

Noticeable and negligible functions are used to characterize the "largeness" or "smallness" of a function describing the probability of some event. Intuitively, a noticeable function is required to be larger than some inverse-polynomially function in the input parameter. On the other hand, a negligible function must be smaller than any inverse-polynomial function of the input parameter. More formally:

Definition 1.3 (Noticeable Function)**.** *A function $\mu(\cdot) : \mathbb{Z}^+ \to [0,1]$ is noticeable iff $\exists c \in \mathbb{Z}^+, n_0 \in \mathbb{Z}^+$ such that $\forall n \geq n_0,\ \mu(n) > n^{-c}$.*

Example. Observe that $\mu(n) = n^{-3}$ is a noticeable function. (Notice that the above definition is satisfied for $c = 4$ and $n_0 = 1$.)

Definition 1.4 (Negligible Function)**.** *A function $\mu(\cdot) : \mathbb{Z}^+ \to [0,1]$ is negligible iff $\forall c \in \mathbb{Z}^+\ \exists n_0 \in \mathbb{Z}^+$ such that $\forall n \geq n_0,\ \mu(n) < n^{-c}$.*

Example. $\mu(n) = 2^{-n}$ is an example of a negligible function. This can be observed as follows. Consider an arbitrary $c \in \mathbb{Z}^+$ and set $n_0 = c^2$. Now, observe that for all $n \geq n_0$, we have that $\frac{n}{\log_2 n} \geq \frac{n_0}{\log_2 n_0} > \frac{n_0}{\sqrt{n_0}} = \sqrt{n_0} = c$. This allows us to conclude that

$$\mu(n) = 2^{-n} = n^{-\frac{n}{\log_2 n}} < n^{-c}.$$

Thus, we have proved that for any $c \in \mathbb{Z}^+$, there exists $n_0 \in \mathbb{Z}^+$ such that for any $n \geq n_0$, $\mu(n) < n^{-c}$.

Gap between Noticeable and Negligible Functions. At first thought it might seem that a function that is not negligible (or, a non-negligible function) must be a noticeable. This is not true![2] Negating the definition of a negligible function, we obtain that a non-negligible function $\mu(\cdot)$ is such that $\exists c \in \mathbb{Z}^+$ such that $\forall n_0 \in \mathbb{Z}^+, \exists n \geq n_0$ such that $\mu(n) > n^{-c}$. Note that this requirement is satisfied as long as $\mu(n) > n^{-c}$ for infinitely many choices of $n \in \mathbb{Z}^+$. However, a noticeable function requires this condition to be true for every $n \geq n_0$.

Below we give example of a function $\mu(\cdot)$ that is neither negligible nor noticeable.

$$\mu(n) = \begin{cases} 2^{-n} & : x \mod 2 = 0 \\ n^{-3} & : x \mod 2 \neq 0 \end{cases}$$

This function is obtained by interleaving negligible and noticeable functions. It cannot be negligible (resp., noticeable) because it is greater (resp., less) than an inverse-polynomially function for infinitely many input choices.

[2] Mihir Bellare. A note on negligible functions. *Journal of Cryptology*, 15(4):271–284, September 2002

Properties of Negligible Functions. Sum and product of two negligible functions is still a negligible function. We argue this for the sum function below and defer the problem for products to Exercise 2.2. These properties together imply that any polynomial function of a negligible function is still negligible.

Exercise 1.1. *If $\mu(n)$ and $\nu(n)$ are negligible functions from domain $\mathbb{Z}^+$ to range $[0,1]$ then prove that the following functions are also negligible:*

1. $\psi_1(n) = \frac{1}{2} \cdot (\mu(n) + \nu(n))$
2. $\psi_2(n) = \min\{\mu(n) + \nu(n), 1\}$
3. $\psi_3(n) = \mu(n) \cdot \nu(n)$
4. $\psi_4(n) = \mathsf{poly}(\mu(n))$, *where* $\mathsf{poly}(\cdot)$ *is an unspecified polynomial function. (Assume that the output is also clamped to* $[0,1]$ *to satisfy the definition)*

function.

Proof.

1. We need to show that for any $c \in \mathbb{Z}^+$, we can find n_0 such that $\forall n \geq n_0$, $\psi_1(n) \leq n^{-c}$. Our argument proceeds as follows. Given the fact that μ and ν are negligible we can conclude that there exist n_1 and n_2 such that $\forall n \geq n_1$, $\mu(n) < n^{-c}$ and $\forall n \geq n_2$, $g(n) < n^{-c}$. Combining the above two facts and setting $n_0 = \max(n_1, n_2)$ we have that for every $n \geq n_0$,

$$\psi_1(n) = \frac{1}{2} \cdot (\mu(n) + \nu(n)) < \frac{1}{2} \cdot (n^{-c} + n^{-c}) = n^{-c}$$

   Thus, $\psi_1(n) \leq n^{-c}$ and hence is negligible.

2. We need to show that for any $c \in \mathbb{Z}^+$, we can find n_0 such that $\forall n \geq n_0$, $\psi_2(n) \leq n^{-c}$. Given the fact that μ and ν are negligible, there exist n_1 and n_2 such that $\forall n \geq n_1$, $\mu(n) \leq n^{-c-1}$ and $\forall n \geq n_2$, $g(n) \leq n^{-c-1}$. Setting $n_0 = \max(n_1, n_2, 3)$ we have that for every $n \geq n_0$,

$$\psi_2(n) = \min\{\mu(n) + \nu(n), 1\} < n^{-c-1} + n^{-c-1} < n^{-c}$$

□

1.3 Computationally Hard Problems

We will next provide certain number theoretical problems that are conjectured to be computationally intractable. We will use the conjectured hardness of these problems in subsequent chapters to o provide concrete instantiations.

1.3.1 The Discrete-Log Family of Problem

Consider a group $\mathbb{G}$ of prime order. For example, consider the group $\mathbb{Z}_p^*$ where p is a large prime. Let g be a generator of this group $\mathbb{G}$. In this group, given g^x for a random $x \in \{1, \ldots p-1\}$ consider the problem of finding x. This problem, referred to as the discrete-log problem, is believed to be computationally hard.

The asymptotic definition of the discrete-log problem needs to consider an infinite family of groups or what we will a group ensemble.

Group Ensemble. A group ensemble is a set of finite cyclic groups $\mathcal{G} = \{\mathbb{G}_n\}_{n \in \mathbb{N}}$. For the group G_n, we assume that given two group elements in G_n, their sum can be computed in polynomial in n time. Additionally, we assume that given n the generator g of $\mathbb{G}_n$ can be computed in polynomial time.

Definition 1.5 (Discrete-Log Assumption)**.** *We say that the discrete-log assumption holds for the group ensemble $\mathcal{G} = \{\mathbb{G}_n\}_{n \in \mathbb{N}}$, if for every non-uniform PPT algorithm $\mathcal{A}$ we have that*

$$\mu_{\mathcal{A}}(n) := \Pr_{x \leftarrow |G_n|}[\mathcal{A}(g, g^x) = x]$$

is a negligible function.

The Diffie-Hellman Problems. In addition to the discrete-log assumption, we also define the Computational Diffie-Hellman Assumption and the Decisional Diffie-Hellman Assumption.

Definition 1.6 (Computational Diffie-Hellman (CDH) Assumption)**.** *We say that the Computational Diffie-Hellman Assumption holds for the group ensemble $\mathcal{G} = \{\mathbb{G}_n\}_{n \in \mathbb{N}}$, if for every non-uniform PPT algorithm $\mathcal{A}$ we have that*

$$\mu_{\mathcal{A}}(n) := \Pr_{x,y \leftarrow |G_n|}[\mathcal{A}(g, g^x, g^y) = g^{xy}]$$

is a negligible function.

Definition 1.7 (Decisional Diffie-Hellman (DDH) Assumption)**.** *We say that the Computational Diffie-Hellman Assumption holds for the group ensemble $\mathcal{G} = \{\mathbb{G}_n\}_{n \in \mathbb{N}}$, if for every non-uniform PPT algorithm $\mathcal{A}$ we have that*

$$\mu_{\mathcal{A}}(n) = \mid \Pr_{x,y \leftarrow |G_n|}[\mathcal{A}(g, g^x, g^y, g^{xy}) = 1] - \Pr_{x,y,z \leftarrow |G_n|}[\mathcal{A}(g, g^x, g^y, g^z) = 1] \mid$$

is a negligible function.

It is not hard to observe that the discrete-log assumption is the weakest of the three assumptions above. In fact, it is not difficult to show that the Discrete-Log Assumption for $\mathcal{G}$ implies the CDH and the DDH Assumptions for $\mathcal{G}$. Additionally, we leave it as an exercise to show that the CDH Assumption for $\mathcal{G}$ implies the DDH Assumptions for $\mathcal{G}$.

Examples of Groups where these assumptions hold. Now we provide some examples of group where these assumptions hold.

1. Consider the group $\mathbb{Z}_p^*$ for a prime p.[3] For this group the CDH Assumption is conjectured to be true. However, using the Legendre symbol,[4] the DDH Assumption in this group can be shown to be false. Can you show how?[5]
2. Let $p = 2q + 1$ where both p and q are prime.[6] Next, let $\mathbb{Q}$ be the order-q subgroup of quadratic residues in $\mathbb{Z}_p^*$. For this group, the DDH assumption is believed to hold.
3. Let $N = pq$ where $p, q, \frac{p-1}{2}$ and $\frac{q-1}{2}$ are primes. Let $\mathbb{QR}_N$ be the cyclic subgroup of qudratic resides of order $\phi(N) = (p-1)(q-1)$. For this group $\mathbb{QR}_N$, the DDH assumption is also believed to hold.

Is DDH strictly stronger than Discrete-Log? In the example cases above, where DDH is known believed to be hard, the best known algorithms for DDH are no better than the best known algorithms for the discrete-log problem. Whether the DDH assumption is strictly stronger than the discrete-log assumption is an open problem.

1.3.2 CDH in $\mathbb{QR}_N$ implies Factoring

In this section, we will show that the CDH assumption in $\mathbb{QR}_N$ implies the factoring assumption.

Lemma 1.1. *Given an algorithm $\mathcal{A}$ that breaks the CDH assumption in $\mathbb{QR}_N$, we construct an non-uniform PPT adversary $\mathcal{B}$ that on input N outputs its prime factors p and q.*

Proof. Given that $\mathcal{A}$ is an algorithm that solves the CDH problem in $\mathbb{QR}_N$ with a non-negligible probability, we construct an algorithm $\mathcal{B}$ that can factor N. Specifically, $\mathcal{B}$ on input N proceeds as follows:

1. Sample $v \leftarrow \mathbb{QR}_N$ (such a v can be obtained by sampling a random value in $\mathbb{Z}_N^*$ and squaring it) and compute $g := v^2 \mod N$.
2. Sample $x, y \leftarrow [N]$.[7]
3. Let $u := \mathcal{A}(g, g^x \cdot v, g^y \cdot v)^8$ and compute $w := \frac{u}{g^{xy} \cdot v^{x+y}}$.

[3] Since the number of primes is infinite we can define an infinite family of such groups. For the sake of simplicity, here we only consider a single group.

[4] Let p be an odd prime number. An integer a is said to be a *quadratic residue* modulo p if it is congruent to a perfect square modulo p and is said to be a *quadratic non-residue* modulo p otherwise. The *Legendre symbol* is a function of a and p defined as

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{if } a \text{ is quadration residue mod } p \text{ and } a \not\equiv \\ -1 & \text{if } a \text{ is quadration non-residue mod } p \\ 0 & \text{if } a \equiv 0 \mod p \end{cases}$$

Legendre symbol can be efficiently computed as $\left(\frac{a}{p}\right) = a^{\frac{p-1}{2}} \mod p$.

[5] This is because given g^x, g^y one can easily compute deduce the Legendre symbol of g^{xy}. Observe that if $\left(\frac{g}{p}\right) = -1$ then we have that $\left(\frac{g^{xy}}{p}\right) = 1$ if and only if $\left(\frac{g^x}{p}\right) = 1$ or $\left(\frac{g^y}{p}\right) = 1$. Using this fact, we can construct an adversary that breaks the DDH problem with a non-negligible (in fact, noticeable) probability.

[6] By Dirichet's Theorem on primes in arithmetic progression, we have that there are infinite choices of primes (p, q) for which $p = 2q + 1$. This allows us to generalize this group to a group ensemble.

[7] Note that sampling x, y uniformly from $[N]$ is statistically close to sampling x, y uniformly from $[\phi(N)]$.

[8] Note that $g^x \cdot v$ where $x \leftarrow [N]$ is statistically close to g^x where $x \leftarrow [N]$.

4. If $w^2 = v^2 \mod N$ and $u \neq \pm v$, then compute the factors of N as $\mathsf{gcd}(N, u+v)$ and $N/\mathsf{gcd}(N, u+v)$. Otherwise, output $\perp$.

Observe that if $\mathcal{A}$ solves the CDH problem then the returned values $u = g^{(x+2^{-1})(y+2^{-1})} = v^{2xy+x+y+2^{-1}}$. Consequently, the computed value $w = v^{2^{-1}}$. Furthermore, with probability $\frac{1}{2}$ we have that $w \neq v$. In this case, $\mathcal{B}$ can factor N. □

2
One-Way Functions

Cryptographers often attempt to base cryptographic results on conjectured computational assumptions to leverage reduced adversarial capabilities. Furthermore, the security of these constructions is no better than the assumptions they are based on.

> *Cryptographers seldom sleep well.*[1]

Thus, basing cryptographic tasks on the *minimal* necessary assumptions is a key tenet in cryptography. Towards this goal, rather can making assumptions about specific computational problem in number theory, cryptographers often consider *abstract primitives*. The existence of these abstract primitives can then be based on one or more computational problems in number theory.

The weakest abstract primitive cryptographers consider is one-way functions. Virtually, every cryptographic goal of interest is known to imply the existence of one-way functions. In other words, most cryptographic tasks would be impossible if the existence of one-way functions was ruled out. On the flip side, the realizing cryptographic tasks from just one-way functions would be ideal.

2.1 *Definition*

A one-way function $f : \{0,1\}^n \to \{0,1\}^m$ is a function that is easy to compute but hard to invert. This intuitive notion is trickier to formalize than it might appear on first thought.

[1] Quote by Silvio Micali in personal communication with Joe Kilian.

Definition 2.1 (One-Way Functions). *A function $f : \{0,1\}^* \to \{0,1\}^*$ is said to be one-way function if:*

- ***Easy to Compute:*** *$\exists$ a (deterministic) polynomial time machine M such that $\forall x \in \{0,1\}^*$ we have that*

$$M(x) = f(x)$$

- ***Hard to Invert:*** *$\forall$ non-uniform PPT adversary $\mathcal{A}$ we have that*

$$\mu_{\mathcal{A},f}(n) = \Pr_{x \xleftarrow{\$} \{0,1\}^n} [\mathcal{A}(1^n, f(x)) \in f^{-1}(f(x))] \tag{2.1}$$

is a negligible function, $x \xleftarrow{\$} \{0,1\}^n$ denotes that x is drawn uniformly at random from the set $\{0,1\}^n$, $f^{-1}(f(x)) = \{x' \mid f(x) = f(x')\}$, and the probability is over the random choices of x and the random coins of $\mathcal{A}$[2].

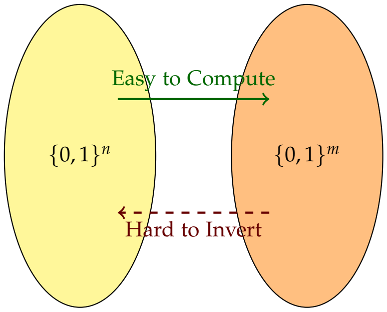


Figure 2.1: Visulizing One-way Funcations

We note that the function is not necessarily one-to-one. In other words, it is possible that $f(x) = f(x')$ for $x \neq x'$ – and the adversary is allowed to output any such x'.

The above definition is rather delicate. We next describe problems in the slight variants of this definition that are insecure.

1. What if we require that $\Pr_{x \xleftarrow{\$} \{0,1\}^n}[\mathcal{A}(1^n, f(x)) \in f^{-1}(f(x))] = 0$ instead of being negligible?

   This condition is false for every function f. An adversary $\mathcal{A}$ that outputs an arbitrarily fixed value x_0 succeeds with probability at least $1/2^n$, as $x_0 = x$ with at least the same probability.

2. What if we drop the input 1^n to $\mathcal{A}$ in Equation 2.1?

   Consider the function $f(x) = |x|$. In this case, we have that $m = \log_2 n$, or $n = 2^m$. Intuitively, f should not be considered a one-way function, because it is easy to invert f. Namely, given a value y any x such that $|x| = y$ is such that $x \in f^{-1}(y)$. However, according to this definition the adversary gets an m bit string as input, and hence is restricted to running in time polynomial in m. Since each possible x is of size $n = 2^m$, the adversary doesn't even have enough time to write down the answer! Thus, according to the flawed definition above, f would be a one-way function.

   Providing the attacker with 1^n (n repetitions of the 1 bit) as additional input avoids this issue. In particular, it allows the attacker to run in time polynomial in m and n.

Candidate One-way Functions. It is not known whether one-way functions exist. In fact, the existence of one-way functions would

[2] Typically, the probability is only taken over the random choices of x, since we can fix the random coins of the adversary $\mathcal{A}$ that maximize its advantage.

imply that $P \neq NP$ (see Exercise 2.3).

However, there are candidates of functions that could be one-way functions, based on the difficulty of certain computational problems. (See Section 1.3)

Let's suppose that the discrete-log assumption hold for group ensemble $\mathcal{G} = \{\mathbb{G}_n\}$ then we have that the function family $\{f_n\}$ where $f_n : \{1, \ldots |\mathbb{G}_n|\} \rightarrow \mathbb{G}_n$ is a one-way function family. In particular, $f_n(x) = g^x$ where g is the generator of the group $\mathbb{G}_n$. The proof that $\{f_n\}$ is one-way based on the Discrete-Log Assumption (see Definition 1.5) is left as as an exercise.

2.2 *Robustness and Brittleness of One-way Functions*

What operations can we perform on one-way functions and still have a one-way function? In this section, we explore the robustness and brittleness of one-way functions and some operations that are safe or unsafe to perform on them.

2.2.1 *Robustness*

Consider having a one-way function f. Can we use this function f in order to make a more structured one-way function g such that $g(x_0) = y_0$ for some constants x_0, y_0, or would this make the function no longer be one-way?

Intuitively, the answer is yes - we can specifically set $g(x_0) = y_0$, and otherwise have $g(x) = f(x)$. In this case, the adversary gains the knowledge of how to invert y_0, but that will only happen with negligible probability, and so the function is still one-way.

In fact, this can be done for an exponential number of x_0, y_0 pairs. To illustrate that, consider the following function:

$$g(x_1 \| x_2) = \begin{cases} x_1 \| x_2 & : x_1 = 0^{n/2} \\ f(x_1 \| x_2) & : \text{otherwise} \end{cases}$$

However, this raises an apparent contradiction - according to this theorem, given a one-way function f, we could keep fixing each of its values to 0, and it would continue to be a one-way function. If we kept doing this, we would eventually end up with a function which outputs 0 for *all* of the possible values of x. How could this still be one-way?

The resolution of this apparent paradox is by noticing that a one-way function is only required to be one-way in the limit where n grows very large. So, no matter how many times we fix the values of f to be 0, we are still only setting a finite number of x values to 0. However, this will still satisfy the definition of a one-way function

- it is just that we will have to use larger and larger values of n_0 in order to prove that the probability of breaking the one-way function is negligible.

2.2.2 Brittleness

Example: OWFs do not always compose securely. Given a one-way function $f : \{0,1\}^n \rightarrow \{0,1\}^n$, is the function $f^2(x) = f(f(x))$ also a one-way function? Intuitively, it seems that if it is hard to invert $f(x)$, then it would be just as hard to invert $f(f(x))$. However, this intuition is incorrect and highlights the delicacy when working with cryptographic assumptions and primitives. In particular, assuming one-way functions exists we describe a one-way function $f : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^{2n}$ such that f^2 can be efficiently inverted. Let $g : \{0,1\}^n \rightarrow \{0,1\}^n$ be a one-way function then we set f as follows:

$$f(x_1, x_2) = 0^n \| g(x_1)$$

Two observations follow:

1. f^2 is not one-way. This follows from the fact that for all inputs x_1, x_2 we have that $f^2(x_1, x_2) = 0^{2n}$. This function is clearly not one-way!

2. f is one-way. This can be argued as follows. Assume that there exists an adversary $\mathcal{A}$ such that $\mu_{\mathcal{A},f}(n) = \Pr_{x \xleftarrow{\$} \{0,1\}^n}[\mathcal{A}(1^{2n}, f(x)) \in f^{-1}(f(x))]$ is non-negligible. Using such an $\mathcal{A}$ we will describe a construction of adversary $\mathcal{B}$ such that $\mu_{\mathcal{B},g}(n) = \Pr_{x \xleftarrow{\$} \{0,1\}^n}[\mathcal{B}(1^n, g(x)) \in g^{-1}(g(x))]$ is also non-negligible. This would be a contradiction thus proving our claim.

   Description of $\mathcal{B}$: $\mathcal{B}$ on input $y \in \{0,1\}^n$ outputs the n lower-order bits of $\mathcal{A}(1^{2n}, 0^n \| y)$.

   Observe that if $\mathcal{A}$ successfully inverts f then we have that $\mathcal{B}$ successfully inverts g. More formally, we have that:

   $$\mu_{\mathcal{B},g}(n) = \Pr_{x \xleftarrow{\$} \{0,1\}^n} \left[\mathcal{A}(1^{2n}, 0^n || g(x)) \in \{0,1\}^n || g^{-1}(g(x)) \right].$$

   But

   $$\begin{aligned} \mu_{\mathcal{A},f}(2n) = & \Pr_{x_1, x_2 \xleftarrow{\$} \{0,1\}^{2n}} [\mathcal{A}(1^{2n}, f(x_1, x_2)) \in f^{-1}(f(\tilde{x}))] \\ = & \Pr_{x_1 \xleftarrow{\$} \{0,1\}^n} [\mathcal{A}(1^{2n}, 0^n || g(x_2)) \in \{0,1\}^n || g^{-1}(g(x_2))] \\ = & \mu_{\mathcal{B},g}(n). \end{aligned}$$

   Hence, we have that $\mu_{\mathcal{B},g}(n) = \mu_{\mathcal{A},f}(2n)$ which is non-negligible as long as $\mu_{\mathcal{A},f}(2n)$ is non-negligible.

Example: Dropping a bit is not always secure. Below is another example of a transformation that does not work. Given any one-way function g, let $g'(x)$ be $g(x)$ with the first bit omitted.

Claim 2.1. *g' is not necessarily one-way. In other words, there exists a OWF function g for which g' is not one-way.*

Proof. We must (1) construct a function g, (2) show that g is one-way, and (3) show that g' is not one-way.

Step 1: Construct a OWF g. To do this, we first want to come up with a (contrived) function g and prove that it is one-way. Let us assume that there exists a one-way function $h : \{0,1\}^n \to \{0,1\}^n$. We define the function $g : \{0,1\}^{2n} \to \{0,1\}^{2n}$ as follows:

$$g(x\|y) = \begin{cases} 0^n\|y & \text{if } x = 0^n \\ 1\|0^{n-1}\|g(y) & \text{otherwise} \end{cases}$$

Step 2: Prove that g is one-way.

Claim 2.2. *If h is a one-way function, then so is g.*

Proof. Assume for the sake of contradiction that g is not one-way. Then there exists a polynomial time adversary $\mathcal{A}$ and a non-negligible function $\mu(\cdot)$ such that:

$$\Pr_{x,y}[\mathcal{A}(1^n, g(x\|y)) \in g^{-1}(g(x\|y))] = \mu(n)$$

We will use such an adversary $\mathcal{A}$ to invert h with some non-negligible probability. This contradicts the one-wayness of h and thus our assumption that g is not one-way function is false.

Let us now construct an $\mathcal{B}$ that uses $\mathcal{A}$ and inverts h. $\mathcal{B}$ is given $1^n, h(y)$ for a randomly chosen y and its goal is to output $y' \in h^{-1}(h(y))$ with some non-negligible probability. $\mathcal{B}$ works as follows:

1. It samples $x \leftarrow \{0,1\}^n$ randomly.
2. If $x = 0^n$, it samples a random $y' \leftarrow \{0,1\}^n$ and outputs it.
3. Otherwise, it runs $\mathcal{A}(10^{n-1}\|h(y))$ and obtains $x'\|y'$. It outputs y'.

Let us first analyze the running time of $\mathcal{B}$. The first two steps are clearly polynomial (in n) time. In the third step, $\mathcal{B}$ runs $\mathcal{A}$ and uses its output. Note that the running time of since $\mathcal{A}$ runs in polynomial (in n) time, this step also takes polynomial (in n) time. Thus, the overall running time of $\mathcal{B}$ is polynomial (in n).

Let us now calculate the probability that $\mathcal{B}$ outputs the correct inverse. If $x = 0^n$, the probability that y' is the correct inverse is at least $\frac{1}{2^n}$ (because it guesses y' randomly and probability that a random y' is the correct inverse is $\geq 1/2^n$). On the other hand, if $x \neq 0^n$, then the probability that $\mathcal{B}$ outputs the correct inverse is $\mu(n)$. Thus,

$$\begin{aligned}
\Pr[\mathcal{B}(1^n, h(y)) \in h^{-1}(h(y))] &\geq \Pr[x = 0^n](\frac{1}{2^n}) + \Pr[x \neq 0^n]\mu(n) \\
&= \frac{1}{2^{2n}} + (1 - \frac{1}{2^n})\mu(n) \\
&\geq \mu(n) - (\frac{1}{2^n} - \frac{1}{2^{2n}})
\end{aligned}$$

Since $\mu(n)$ is a non-negligible function and $(\frac{1}{2^n} - \frac{1}{2^{2n}})$ is a negligible function, their difference is non-negligible.[3] This contradicts the one-wayness of h.

□

Step 3: Prove that g' is not one-way. We construct the new function $g' : \{0,1\}^{2n} \to \{0,1\}^{2n-1}$ by dropping the first bit of g. That is,

$$g'(x\|y) = \begin{cases} 0^{n-1}\|y & \text{if } x = 0^n \\ 0^{n-1}\|g(y) & \text{otherwise} \end{cases}$$

We now want to prove that g' is not one-way. That is, we want to design an adversary $\mathcal{C}$ such that given 1^{2n} and $g'(x\|y)$ for a randomly chosen x, y, it outputs an element in the set $g^{-1}(g(x\|y)$. The description of $\mathcal{C}$ is as follows:

- On input 1^{2n} and $g'(x\|y)$, the adversary $\mathcal{C}$ parses $g'(x\|y)$ as $0^{n-1}\|\overline{y}$.

- It outputs $0^n\|\overline{y}$ as the inverse.

Notice that $g'(0^n\|\overline{y}) = 0^{n-1}\|\overline{y}$. Thus, $\mathcal{C}$ succeeds with probability 1 and this breaks the one-wayness of g'.

□

2.3 *Hardness Amplification*

In this section, we show that even a very *weak* form of one-way functions suffices from constructing one-way functions as defined previously. For this section, we refer to this previously defined notion as strong one-way functions.

[3] Exercise: Prove that if $\alpha(\cdot)$ is a non-negligible function and $\beta(\cdot)$ is a negligible function, then $(\alpha - \beta)(\cdot)$ is a non-negligible function.

Definition 2.2 (Weak One-Way Functions). *A function $f : \{0,1\}^n \rightarrow \{0,1\}^m$ is said to be a weak one-way function if:*

- *f is computable by a polynomial time machine, and*
- *There exists a noticeable function $\alpha_f(\cdot)$ such that $\forall$ non-uniform PPT adversaries $\mathcal{A}$ we have that*

$$\mu_{\mathcal{A},f}(n) = \Pr_{x \overset{\$}{\leftarrow} \{0,1\}^n}[\mathcal{A}(1^n, f(x)) \in f^{-1}(f(x))] \leq 1 - \alpha_f(n).$$

Theorem 2.1. *If there exists a weak one-way function, then there exists a (strong) one-way function.*

Proof. We prove the above theorem constructively. Suppose $f : \{0,1\}^n \rightarrow \{0,1\}^m$ is a weak one-way function, then we prove that the function $g : \{0,1\}^{nq} \rightarrow \{0,1\}^{mq}$ for $q = \lceil \frac{2n}{\alpha_f(n)} \rceil$ where

$$g(x_1, x_2, \cdots, x_q) = f(x_1)||f(x_2)|| \cdots ||f(x_q),$$

is a strong one-way function. Let us discuss the intuition. A weak one-way function is "strong" in a small part of its domain. For this construction to result in a strong one-way function, we need just one of the q instantiations to be in the part of the domain where our weak one-way function is strong. If we pick a large enough q, this is guaranteed to happen.

Assume for the sake of contradiction that there exists an adversary $\mathcal{B}$ such that $\mu_{\mathcal{B},g}(nq) = \Pr_{x \overset{\$}{\leftarrow} \{0,1\}^{nq}}[\mathcal{B}(1^{nq}, g(x)) \in g^{-1}(g(x))]$ is non-negligible. Then we use $\mathcal{B}$ to construct $\mathcal{A}$ (see Figure 2.2) that breaks f, namely $\mu_{\mathcal{A},f}(n) = \Pr_{x \overset{\$}{\leftarrow} \{0,1\}^n}[\mathcal{A}(1^n, f(x)) \in f^{-1}(f(x))] > 1 - \alpha_f(n)$ for sufficiently large n.

```
1. $i \overset{\$}{\leftarrow} [q]$.
2. $x_1, \cdots, x_{i-1}, x_i, \cdots, x_q \overset{\$}{\leftarrow} \{0,1\}^n$.
3. Set $y_j = f(x_j)$ for each $j \in [q]\backslash\{i\}$ and $y_i = y$.
4. $(x'_1, x'_2, \cdots, x'_q) := \mathcal{B}(f(x_1), f(x_2), \cdots, f(x_q))$.
5. $f(x'_i) = y$ then output $x'_i$ else $\perp$.
```

Figure 2.2: Construction of $\mathcal{A}(1^n, y)$

Note that: (1) $\mathcal{A}(1^n, y)$ iterates at most $T = \frac{4n^2}{\alpha_f(n)\mu_{\mathcal{B},g}(nq)}$ times each call is polynomial time. (2) $\mu_{\mathcal{B},g}(nq)$ is a non-negligible function. This implies that for infinite choices of n this value is greater than some noticeable function. Together these two facts imply that for infinite choices of n the running time of $\mathcal{A}$ is bounded by a polynomial function in n.

It remains to show that $\Pr_{x \overset{\$}{\leftarrow} \{0,1\}^n}[\mathcal{A}(1^n, f(x)) = \perp] < \alpha_f(n)$ for arbitrarily large n. A natural way to argue this is by showing that at least one execution of $\mathcal{B}$ should suffice for inverting $f(x)$. However, the technical challenge in proving this formally is that these calls to $\mathcal{B}$ aren't independent. Below we formalize this argument even when these calls aren't independent.

Define the set S of "bad" x's, which are hard to invert:

$$S := \left\{ x \,\middle|\, \Pr_{\mathcal{B}}[\mathcal{A} \text{ inverts } f(x) \text{ in a single iteration}] \leq \frac{\alpha_f(n)\mu_{\mathcal{B},g}(nq)}{4n} \right\}.$$

Lemma 2.1. *Let A be any an efficient algorithm such that $\Pr_{x,r}[A(x,r) = 1] \geq \epsilon$. Additionally, let $G = \{x \mid \geq \Pr_r[A(x,r) = 1] \geq \frac{\epsilon}{2}\}$. Then, we have $\Pr_x[x \in G] \geq \frac{\epsilon}{2}$.*

Proof. The proof of this lemma follows by a very simple counting argument. Let's start by assuming that $\Pr_x[x \in G] < \frac{\epsilon}{2}$. Next, observe that

$$\begin{aligned}
&\Pr_{x,r}[A(x,r) = 1] \\
&\quad = \Pr_x[x \in G] \cdot \Pr_{x,r}[A(x,r) = 1 \mid x \in G] \\
&\quad + \Pr_x[x \notin G] \cdot \Pr_{x,r}[A(x,r) = 1 \mid x \notin G] \\
&\quad < \frac{\epsilon}{2} \cdot 1 + 1 \cdot \frac{\epsilon}{2} \\
&\quad < \epsilon,
\end{aligned}$$

which is a contradiction. □

We start by proving that the size of S is small. More formally,

$$\Pr_{x \xleftarrow{\$} \{0,1\}^n} [x \in S] \leq \frac{\alpha_f(n)}{2}.$$

Assume, for the sake of contradiction, that $\Pr_{x \xleftarrow{\$} \{0,1\}^n} [x \in S] > \frac{\alpha_f(n)}{2}$. Then we have that:

$$\begin{aligned}
\mu_{\mathcal{B},g}(nq) = & \Pr_{(x_1,\cdots,x_q) \xleftarrow{\$} \{0,1\}^{nq}} [\mathcal{B}(1^{nq}, g(x_1,\cdots,x_q)) \in g^{-1}(g(x_1,\cdots,x_q))] \\
= & \Pr_{x_1,\cdots,x_q} [\mathcal{B}(1^{nq}, g(x_1,\cdots,x_q)) \in g^{-1}(g(x_1,\cdots,x_q)) \wedge \forall i : x_i \notin S] \\
& + \Pr_{x_1,\cdots,x_q} [\mathcal{B}(1^{nq}, g(x_1,\cdots,x_q)) \in g^{-1}(g(x_1,\cdots,x_q)) \wedge \exists i : x_i \in S] \\
\leq & \Pr_{x_1,\cdots,x_q} [\forall i : x_i \notin S] + \sum_{i=1}^{q} \Pr_{x_1,\cdots,x_q} [\mathcal{B}(1^{nq}, g(x_1,\cdots,x_q)) \in g^{-1}(g(x_1,\cdots,x_q)) \wedge x_i \in S] \\
\leq & \left(1 - \frac{\alpha_f(n)}{2}\right)^q + q \cdot \Pr_{x_1,\cdots,x_q,i} [\mathcal{B}(1^{nq}, g(x_1,\cdots,x_q)) \in g^{-1}(g(x_1,\cdots,x_q)) \wedge x_i \in S] \\
= & \left(1 - \frac{\alpha_f(n)}{2}\right)^{\frac{2n}{\alpha_f(n)}} + q \cdot \Pr_{x \xleftarrow{\$} \{0,1\}^n, \mathcal{B}} [\mathcal{A} \text{ inverts } f(x) \text{ in a single iteration} \wedge x \in S] \\
\leq & e^{-n} + q \cdot \Pr_x[x \in S] \cdot \Pr[\mathcal{A} \text{ inverts } f(x) \text{ in a single iteration} \mid x \in S] \\
\leq & e^{-n} + \frac{2n}{\alpha_f(n)} \cdot 1 \cdot \frac{\mu_{\mathcal{B},g}(nq) \cdot \alpha_f(n)}{4n} \\
\leq & e^{-n} + \frac{\mu_{\mathcal{B},g}(nq)}{2}.
\end{aligned}$$

Hence $\mu_{\mathcal{B},g}(nq) \leq 2e^{-n}$, contradicting with the fact that $\mu_{\mathcal{B},g}$ is non-negligible. Then we have

$$\begin{aligned}
& \Pr_{x \xleftarrow{\$} \{0,1\}^n} [\mathcal{A}(1^n, f(x)) = \perp] \\
& = \Pr_x[x \in S] + \Pr_x[x \notin S] \cdot \Pr[\mathcal{B} \text{ fails to invert } f(x) \text{ in every iteration} | x \notin S] \\
& \leq \frac{\alpha_f(n)}{2} + (\Pr[\mathcal{B} \text{ fails to invert } f(x) \text{ a single iteration} | x \notin S])^T \\
& \leq \frac{\alpha_f(n)}{2} + \left(1 - \frac{\mu_{\mathcal{A},g}(nq) \cdot \alpha_f(n)}{4n}\right)^T \\
& \leq \frac{\alpha_f(n)}{2} + e^{-n} \leq \alpha_f(n)
\end{aligned}$$

for sufficiently large n. This concludes the proof. □

Lemma 2.2. *Let A be any an efficient algorithm such that $\Pr_{x,r}[A(x_1, \ldots x_n, r) = 1] \geq \epsilon$. Additionally, let $G = \{x \mid\geq \Pr_{x_1,\ldots x_n,r}[A(x,r) = 1 \mid \exists i, x = x_i] \geq \frac{\epsilon}{2}\}$. Then, we have $\Pr_x[x \in G] \geq \frac{\epsilon}{2}$.*

Proof. The proof of this lemma follows by a very simple counting argument. Let's start by assuming that $\Pr_x[x \in G] < \frac{\epsilon}{2}$. Next, observe that

$$\begin{aligned}
& \Pr_{x,r}[A(x,r) = 1] \\
& \quad = \Pr_x[x \in G] \cdot \Pr_{x,r}[A(x,r) = 1 \mid x \in G] \\
& \quad + \Pr_x[x \notin G] \cdot \Pr_{x,r}[A(x,r) = 1 \mid x \notin G] \\
& \quad < \frac{\epsilon}{2} \cdot 1 + 1 \cdot \frac{\epsilon}{2} \\
& \quad < \epsilon,
\end{aligned}$$

which is a contradiction. □

2.4 *Levin's One-Way Function*

In this section, we discuss Levin's one-way function, which is an explicit construction of a one-way function that is secure as long as a one-way function exists. This is interesting because unlike a typical cryptographic primitive that relies on a specific hardness assumption (which may or may not hold in the future), Levin's one-way function is future-proof in the sense that it will be secure as long as atleast one hardness assumption holds (which we may or may not discover).

The high-level intuition behind Levin's construction is as follows: since we assume one-way functions exist, there exists a uniform machine $\tilde{M}$ such that $|\tilde{M}|$ is a constant and $\tilde{M}(x)$ is hard to invert for a random input x. Now, consider a function h that parses the first $\log(n)$ bits of its n-bit input as the code of a machine M and the remaining bits as the input to M. For a large enough n that is exponential in $|\tilde{M}|$, note that we will hit the code of $\tilde{M}$ with noticeable probability in n, and for those instances, h will be hard to invert. It is easy to see that this gives us a weak one-way function which has a noticeable probability of being hard to invert, and we can amplify the hardness of this weak one-way function to get an explicit construction of a one-way function.

Theorem 2.2. *If there exists a one-way function, then there exists an explicit function f that is one-way (constructively).*

Before we look at the construction and the proof in detail, we first prove a lemma that will be useful in the proof. In particular, we need a bound on the running time of the one-way function $\tilde{M}$ so that we can upper bound the execution time of h, since there could be inputs to g that do not terminate in polynomial time. To this end, we prove the following lemma which shows that if a one-way function exists, then there is also a one-way function that runs in time n^2, and thus, we can bound h to n^2 steps.

Lemma 2.3. *If there exists a one-way function computable in time n^c for a constant c, then there exists a one-way function computable in time n^2.*

Proof. Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way function computable in time n^c. Construct $g : \{0,1\}^{n+n^c} \to \{0,1\}^{n+n^c}$ as follows:

$$g(x,y) = f(x)||y$$

where $x \in \{0,1\}^n, y \in \{0,1\}^{n^c}$. $g(x,y)$ takes time $2n^c$, which is linear in the input length.

We next show that $g(\cdot)$ is one-way. Assume for the purpose of contradiction that there exists an adversary $\mathcal{A}$ such that $\mu_{\mathcal{A},g}(n + n^c) = \Pr_{(x,y) \xleftarrow{\$} \{0,1\}^{n+n^c}}[\mathcal{A}(1^{n+n^c}, g(x,y)) \in g^{-1}(g(x,y))]$ is non-negligible. Then we use $\mathcal{A}$ to construct $\mathcal{B}$ such that $\mu_{\mathcal{B},f}(n) = \Pr_{x \xleftarrow{\$} \{0,1\}^n}[\mathcal{B}(1^n, f(x)) \in f^{-1}(f(x))]$ is also non-negligible.

$\mathcal{B}$ on input $z \in \{0,1\}^n$, samples $y \xleftarrow{\$} \{0,1\}^{n^c}$, and outputs the n higher-order bits of $\mathcal{A}(1^{n+n^c}, z||y)$. Then we have

$$\begin{aligned}\mu_{\mathcal{B},g}(n) = &\Pr_{x\xleftarrow{\$}\{0,1\}^n, y\xleftarrow{\$}\{0,1\}^{n^c}}\left[\mathcal{A}(1^{n+n^c}, f(x)||y) \in f^{-1}(f(x))||\{0,1\}^{n^c}\right]\\ \geq &\Pr_{x,y}\left[\mathcal{A}(1^{n+n^c}, g(x,y)) \in f^{-1}(f(x))||y\right]\\ = &\Pr_{x,y}\left[\mathcal{A}(1^{n+n^c}, g(x,y)) \in g^{-1}(g(x,y))\right]\end{aligned}$$

is non-negligible. □

Now, we provide the explicit construction of h and prove that it is a weak one-way function. Since h is an (explicit) weak one-way function, we can construct an (explicit) one-way function from h as we discussed in Section 2.3, and this would prove Theorem 2.2.

Proof of Theorem 2.2. $h : \{0,1\}^n \to \{0,1\}^n$ is defined as follows:

$$h(M,x) = \begin{cases} M||M(x) & \text{if } M(x) \text{ takes no more than } |x|^2 \text{ steps}\\ M||0 & \text{otherwise}\end{cases}$$

where $|M| = \log n, |x| = n - \log n$ (interpreting M as the code of a machine and x as its input).

It remains to show that if one-way functions exist, then h is a weak one-way function, with $\alpha_h(n) = \frac{1}{n^2}$. Assume for the purpose of contradiction that there exists an adversary $\mathcal{A}$ such that $\mu_{\mathcal{A},h}(n) = \Pr_{(M,x)\xleftarrow{\$}\{0,1\}^n}[\mathcal{A}(1^n, h(M,x)) \in h^{-1}(h(M,x))] \geq 1 - \frac{1}{n^2}$ for all sufficiently large n. By the existence of one-way functions and Lemma 2.3, there exists a one-way function $\tilde{M}$ that can be computed in time n^2. Let $\tilde{M}$ be the uniform machine that computes this one-way function. We will consider values n such that $n > 2^{|\tilde{M}|}$. In other words for these choices of n, $\tilde{M}$ can be described using $\log n$ bits. We construct $\mathcal{B}$ to invert $\tilde{M}$: on input y outputs the $(n - \log n)$ lower-order bits of $\mathcal{A}(1^n, \tilde{M}||y)$. Then

$$\begin{aligned}\mu_{\mathcal{B},\tilde{M}}(n-\log n) = &\Pr_{x\xleftarrow{\$}\{0,1\}^{n-\log n}}\left[\mathcal{A}(1^n, \tilde{M}||\tilde{M}(x)) \in \{0,1\}^{\log n}||\tilde{M}^{-1}(\tilde{M}((x))\right]\\ \geq &\Pr_{x\xleftarrow{\$}\{0,1\}^{n-\log n}}\left[\mathcal{A}(1^n, \tilde{M}||\tilde{M}(x)) \in \tilde{M}||\tilde{M}^{-1}(\tilde{M}((x))\right].\end{aligned}$$

Observe that for sufficiently large n it holds that

$$\begin{aligned}1 - \frac{1}{n^2} \leq &\mu_{\mathcal{A},h}(n)\\ = &\Pr_{(M,x)\xleftarrow{\$}\{0,1\}^n}\left[\mathcal{A}(1^n, h(M,x)) \in h^{-1}(h(M,x))\right]\\ \leq &\Pr_M[M = \tilde{M}] \cdot \Pr_x\left[\mathcal{A}(1^n, \tilde{M}||\tilde{M}(x)) \in \tilde{M}||\tilde{M}^{-1}(\tilde{M}((x))\right] + \Pr_M[M \neq \tilde{M}]\\ \leq &\frac{1}{n} \cdot \mu_{\mathcal{B},\tilde{M}}(n - \log n) + \frac{n-1}{n}.\end{aligned}$$

Hence $\mu_{\mathcal{B},\tilde{M}}(n - \log n) \geq \frac{n-1}{n}$ for sufficiently large n which is a contradiction. □

2.5 Hardness Concentrate Bit

We start by asking the following question: Is it possible to concentrate the strength of a one-way function into one bit? In particular, given a one-way function f, does there exist one bit that can be computed efficiently from the input x, but is hard to compute given $f(x)$?

Definition 2.3 (Hard Concentrate Bit). *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way function. $B : \{0,1\}^n \to \{0,1\}$ is a hard concentrate bit of f if:*

- *B is computable by a polynomial time machine, and*
- *$\forall$ non-uniform PPT adversaries $\mathcal{A}$ we have that*

$$\Pr_{x \xleftarrow{\$} \{0,1\}^n} [\mathcal{A}(1^n, f(x)) = B(x)] \leq \frac{1}{2} + \mathsf{negl}(n).$$

A simple example. Let f be a one-way function. Consider the one-way function $g(b,x) = 0||f(x)$ and a hard concentrate bit $B(b,x) = b$. Intuitively, the value $g(b,x)$ does not reveal any information about the first bit b, thus no information about the value $B(b,x)$ can be ascertained. Hence $\mathcal{A}$ cannot predict the first bit with a non-negligible advantage than a random guess. However, we are more interested in the case where the hard concentrate bit is hidden because of computational hardness and not information theoretic hardness.

Remark 2.1. *Given a one-way function f, we can construct another one-way function g with a hard concentrate bit. However, we may not be able to find a hard concentrate bit for f. In fact, it is an open question whether a hard concentrate bit exists for every one-way function.*

Intuitively, if a function f is one-way, it seems that there should be a particular bit in the input x that is hard to compute given $f(x)$. However, we show that is not true:

Claim 2.3. *If $f : \{0,1\}^n \to \{0,1\}^n$ is a one-way function, then there exists a one-way function $g : \{0,1\}^{n+\log n} \to \{0,1\}^{n+\log n}$ such that $\forall i \in [1, n + \log n]$, $B_i(x) = x_i$ is not a hard concentrate bit, where x_i is the i^{th} bit of x.*

Proof. Define $g : \{0,1\}^{n+\log(n)} \to \{0,1\}^{n+\log(n)}$ as follows.

$$g(x,y) = f(x_{\bar{y}})||x_y||y,$$

where $|x| = n$, $|y| = \log n$, $x_{\bar{y}}$ is all bits of x except the y^{th} bit, and x_y is the y^{th} bit of x.

First, one can show that g is still a one-way function. (We leave this as an exercise!) Next, we show that B_i is not a hard concentrate bit for $\forall i \in [1, n]$ (clearly B_i is not a hard concentrate bit for $i \in [n+1, n+\log n]$). Construct an adversary $\mathcal{A}_i(1^{n+\log n}, f(x_{\bar{y}})||x_y||y)$ that "breaks" B_i:

- If $y \neq i$ then output a random bit;
- Otherwise output x_y.

$$\begin{aligned}
&\Pr_{x,y}[\mathcal{A}(1^{n+\log n}, g(x,y)) = B_i(x)] \\
=&\Pr_{x,y}[\mathcal{A}(1^{n+\log n}, f(x_{\bar{y}})||x_y||y) = x_i] \\
=&\frac{n-1}{n} \cdot \frac{1}{2} + \frac{1}{n} \cdot 1 = \frac{1}{2} + \frac{1}{2n}.
\end{aligned}$$

Hence $\mathcal{A}_i$ can guess the output of B_i with greater than $\frac{1}{2} + \mathsf{negl}(n)$ probability. □

2.5.1 *Hard Concentrate Bit of any One-Way Permutation*

We now show that a slight modification of every one-way function has a hard concentrate bit. More formally,

Theorem 2.3. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way function. Define a function $g : \{0,1\}^{2n} \to \{0,1\}^{2n}$ as follows:*

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

where $|x| = |r| = n$. Then we have that g is one-way and that it has a hard concentrate bit, namely $B(x,r) = \sum_{i=1}^n x_i r_i \mod 2$.

Remark 2.2. *If f is a (one-to-one) one-way function, then g is also a (one-to-one) one-way function with hard concentrate bit $B(\cdot)$.*

Proof. We leave it as an exercise to show that g is a one-way function and below we will prove that the function $B(\cdot)$ describe a hard concentrate bit of g. More specifically, we need to show that if there exists a non-uniform PPT $\mathcal{A}$ s.t. $\Pr_{x,r}[\mathcal{A}(1^{2n}, g(x,r)) = B(x,r)] \geq \frac{1}{2} + \epsilon(n)$, where ϵ is non-negligible, then there exists a non-uniform PPT $\mathcal{B}$ such that $\Pr_{x,r}[\mathcal{B}(1^{2n}, g(x,r)) \in g^{-1}(g(x,r))]$ is non-negligible. Below we use E to denote the event that $\mathcal{A}(1^{2n}, g(x,r)) = B(x,r)$. We will present our proof in three steps, where each step progressively increases in complexity: (1) the super simple case where we restrict to $\mathcal{A}$ such that $\Pr_{x,r}[E] = 1$, (2) the simple case where we restrict to $\mathcal{A}$ such that $\Pr_{x,r}[E] \geq \frac{3}{4} + \epsilon(n)$, and finally (3) the general case where $\Pr_{x,r}[E] \geq \frac{1}{2} + \epsilon(n)$.

Super simple case. Suppose $\mathcal{A}$ guesses $B(\cdot)$ with perfect accuracy:

$$\Pr_{x,r}[E] = 1.$$

We now construct $\mathcal{B}$ that inverts g with perfect accuracy. Let e^i denote the one-hot n-bit string $0\cdots010\cdots0$, where only the i-th bit is 1, the rest are all 0. $\mathcal{B}$ gets $f(x)||r$ as input, and its algorithm is described in Figure 2.3.

```
for $i = 1$ to $n$ do
    $x'_i \leftarrow \mathcal{A}(1^{2n}, f(x)||e^i)$
end for
return $x'_1 \cdots x'_n||r$
```

Figure 2.3: Super-Simple Case $\mathcal{B}$

Observe that $B(x, e^i) = \sum_{j=1}^{n} x_j e^i_j = x_i$. Therefore, the probability that $\mathcal{B}$ inverts a single bit successfully is,

$$\Pr_x\left[\mathcal{A}(1^{2n}, f(x)||e^i) = x_i\right] = \Pr_x\left[\mathcal{A}(1^{2n}, f(x)||e^i) = B(x, e^i)\right] = 1.$$

Hence $\Pr_{x,r}[\mathcal{B}(1^{2n}, g(x,r)) = (x,r)] = 1$.

Simple case. Next moving on to the following more demanding case.

$$\Pr_{x,r}[E] \geq \frac{3}{4} + \epsilon(n),$$

where $\epsilon(\cdot)$ is non-negligible. We describe $\mathcal{B}$'s algorithm for inverting g in Figure 2.4. Here we can no longer use the super simple case algorithm because we no longer know if $\mathcal{A}$ outputs the correct bit on input $f(x)\|e^i$. Instead, we introduce randomness to $\mathcal{A}$'s input expecting that it should be able to guess the right bit on majority of those inputs since it has a high probability of guessing $B(\cdot)$ in general. We now also need to make two calls to $\mathcal{A}$ to isolate the i-th bit of x. Note that an iteration of $\mathcal{B}$ outputs the right bit if calls to $\mathcal{A}$ output the correct bit because $B(x,s) \oplus B(x, s \oplus e^i) = x_i$:

```
for $i = 1$ to $n$ do
    for $t = 1$ to $T = \frac{n}{2\epsilon(n)^2}$ do
        $s \xleftarrow{\$} \{0,1\}^n$
        $x_i^t \leftarrow \mathcal{A}(f(x)||s)$
                 $\oplus\ \mathcal{A}(f(x)||(s \oplus e^i))$
    end for
    $x'_i \leftarrow$ the majority of $\{x_i^1, \cdots, x_i^T\}$
end for
return $x'_1 \cdots x'_n||R$
```

Figure 2.4: Simple Case $\mathcal{B}$

$$\begin{aligned} B(x,s) \oplus B(x, s \oplus e^i) &= \sum_j x_j s_j \oplus \sum_j x_j (s_j \oplus e^i_j) \\ &= \sum_{j \neq i} (x_j s_j \oplus x_j s_j) \oplus x_i s_i \oplus x_i (s_i \oplus 1) \\ &= x_i \end{aligned}$$

The key technical challenge in proving that $\mathcal{B}$ inverts g with non-negligible probability arises from the fact that the calls to $\mathcal{A}$ made during one iteration of $\mathcal{B}$ are not independent. In particular, all calls to $\mathcal{A}$ share the same x and the calls $\mathcal{A}(f(x)||s)$ and $\mathcal{A}(f(x)||(s \oplus e^i))$ use correlated randomness as well.

We solve the first issue by showing that there exists a large set of x values for which $\mathcal{A}$ still works with large probability. The latter issue of lack of independence between $\mathcal{A}(f(x)||s)$ and $\mathcal{A}(f(x)||(s \oplus e^i))$ can be solved using a union bound since the success probability of the adversary $\mathcal{A}$ is high enough.

Formally, define the set G of "good" x's, for which it is easy for $\mathcal{A}$ to predict the right bit:

$$G := \left\{ x \,\middle|\, \Pr_r[E] \geq \frac{3}{4} + \frac{\epsilon(n)}{2} \right\}.$$

Now we prove that G is not a small set. More formally, we claim that:

$$\Pr_{x \xleftarrow{\$} \{0,1\}^n} [x \in G] \geq \frac{\epsilon(n)}{2}.$$

Assume that $\Pr_{x \xleftarrow{\$} \{0,1\}^n} [x \in G] < \frac{\epsilon(n)}{2}$. Then we have the following contradiction:

$$\begin{aligned}
\frac{3}{4} + \epsilon(n) &\leq \Pr_{x,r}[E] \\
&= \Pr_x[x \in G] \Pr_r[E | x \in G] + \Pr_x[x \notin G] \Pr_r[E | x \notin G] \\
&< \frac{\epsilon(n)}{2} \cdot 1 + 1 \cdot \left(\frac{3}{4} + \frac{\epsilon(n)}{2} \right) = \frac{3}{4} + \epsilon(n).
\end{aligned}$$

Now consider a single iteration for a fixed $x \in G$:

$$\begin{aligned}
&\Pr_s \left[\mathcal{A}(f(x), s) \oplus \mathcal{A}(f(x), s \oplus e^i) = x_i \right] \\
= \;& \Pr_s [\text{Both } \mathcal{A}\text{'s are correct}] + \Pr_s [\text{Both } \mathcal{A}\text{'s are wrong}] \\
\geq \;& \Pr_s [\text{Both } \mathcal{A}\text{'s are correct}] = 1 - \Pr_s [\text{Either } \mathcal{A} \text{ is wrong}] \\
\geq \;& 1 - 2 \cdot \Pr_s [\mathcal{A} \text{ is wrong}] \\
\geq \;& 1 - 2 \left(\frac{1}{4} - \frac{\epsilon(n)}{2} \right) = \frac{1}{2} + \epsilon(n).
\end{aligned}$$

Let Y_i^t be the indicator random variable that $x_i^t = x_i$ (namely, $Y_i^t = 1$ with probability $\Pr[x_i^t = x_i]$ and $Y_i^t = 0$ otherwise). Note that $Y_i^1, \cdots, Y_i^T$ are independent and identical random variables, and for all $t \in \{1, \ldots, T\}$, we have $\Pr[Y_i^t = 1] = \Pr[x_i^t = x_i] \geq \frac{1}{2} + \epsilon(n)$. Next we argue that majority of x_i^t coincide with x_i with high probability.

$$\begin{aligned}
\Pr[x_i' \neq x_i] &= \Pr \left[\sum_{t=1}^{T} Y_i^t \leq \frac{T}{2} \right] \\
&= \Pr \left[\sum_{t=1}^{T} Y_i^t - \left(\frac{1}{2} + \epsilon(n) \right) T \leq \frac{T}{2} - \left(\frac{1}{2} + \epsilon(n) \right) T \right] \\
&\leq \Pr \left[\left| \sum_{t=1}^{T} Y_i^t - \left(\frac{1}{2} + \epsilon(n) \right) T \right| \geq \epsilon(n) T \right]
\end{aligned}$$

Let $X_1, \cdots, X_m$ be i.i.d. random variables taking values 0 or 1. Let $\Pr[X_i = 1] = p$.

By Chebyshev's Inequality, $\Pr\left[\left|\sum X_i - pm\right| \geq \delta m\right] \leq \dfrac{1}{4\delta^2 m}$.

$$\leq \frac{1}{4\epsilon(n)^2 T} = \frac{1}{2n}.$$

Then, completing the argument, we have

$$\begin{aligned}
&\Pr_{x,r}[\mathcal{B}(1^{2n}, g(x,r)) = (x,r)] \\
\geq\ & \Pr_x[x \in G] \Pr[x'_1 = x_1, \cdots x'_n = x_n | x \in G] \\
\geq\ & \frac{\epsilon(n)}{2} \cdot \left(1 - \sum_{i=1}^{n} \Pr[x'_i \neq x_i | x \in G]\right) \\
\geq\ & \frac{\epsilon(n)}{2} \cdot \left(1 - n \cdot \frac{1}{2n}\right) = \frac{\epsilon(n)}{4}.
\end{aligned}$$

<u>Real Case.</u> Now, we describe the final case where $\Pr_{x,r}[E] \geq \frac{1}{2} + \epsilon(n)$ and $\epsilon(\cdot)$ is a non-negligible function. The key technical challenge in this case is that we cannot make two related calls to $\mathcal{A}$ as was done in the simple case above since we can't argue that both calls to $\mathcal{A}$ will be correct with high enough probability. However, just using one call to $\mathcal{A}$ seems insufficient. The key idea is to just guess one of those values. Very surprisingly, this idea along with careful analysis magically works out. Just like the previous two cases, we start by describing the algorithm $\mathcal{B}$ in Figure 2.5.

```
$T = \frac{2n}{\epsilon(n)^2}$
for $\ell = 1$ to $\log T$ do
    $s_\ell \xleftarrow{\$} \{0,1\}^n$
    $b_\ell \xleftarrow{\$} \{0,1\}$
end for
for $i = 1$ to $n$ do
    for all $L \subseteq \{1, 2, \cdots, \log T\}$ do
        $S_L := \bigoplus_{j \in L} s_j$
        $B_L := \bigoplus_{j \in L} b_j$
        $x_i^L \leftarrow B_L \oplus \mathcal{A}(f(x)||S_L \oplus e^i)$
    end for
    $x'_i \leftarrow$ majority of $\{x_i^{\emptyset}, \cdots, x_i^{[\log T]}\}$
end for
return $x'_1 \cdots x'_n || R$
```

Figure 2.5: Real Case $\mathcal{B}$

In the beginning of the algorithm, $\mathcal{B}$ samples $\log T$ random strings $\{s_\ell\}_\ell$ and bits $\{b_\ell\}_\ell$. Since there are only $\log T$ values, with probability $\frac{1}{T}$ (which is polynomial in n) all the b_ℓ's are correct, i.e., $b_\ell = B(x, s_\ell)$. In the rest of this proof, we denote this event as F. Now note that if F happens, then B_L as defined in the algorithm is also equal to $B(x, S_L)$ (we denote the k^{th}-bit of s with $(s)_k$):

$$\begin{aligned}
B(x, S_L) &= \sum_{k=1}^{n} x_k (\bigoplus_{j \in L} s_j)_k \\
&= \sum_{k=1}^{n} x_k \sum_{j \in L} (s_j)_k \\
&= \sum_{j \in L} \sum_{k=1}^{n} x_k (s_j)_k \\
&= \sum_{j \in L} B(x, s_j) \\
&= \sum_{j \in L} b_j \\
&= B_L
\end{aligned}$$

Thus, with probability $\frac{1}{T}$, we have all the right guesses for one of the invocations, and we just need to bound the probability that $\mathcal{A}(f(x)||S_L \oplus e^i) = B(x, S_L \oplus e^i)$. However there is a subtle issue.

Now the events $Y_i^{\emptyset}, \cdots, Y_i^{[\log T]}$ are no longer independent. Nevertheless, we can still show that they are pairwise independent, and the Chebyshev's Inequality still holds. Now we give the formal proof.

Just as in the simple case, we define the set G as

$$G := \left\{ x \,\middle|\, \Pr_r [E] \geq \frac{1}{2} + \frac{\epsilon(n)}{2} \right\},$$

and with an identical argument we obtain that:

$$\Pr_{x \xleftarrow{\$} \{0,1\}^n} [x \in G] \geq \frac{\epsilon(n)}{2}$$

Next, given $\{b_\ell = B(x, s_\ell)\}_{\ell \in [\log T]}$ and $x \in G$, we have:

$$\begin{aligned}
&\Pr_r \left[B_L \oplus \mathcal{A}(f(x)||S_L \oplus e^i) = x_i \right] \\
=\ & \Pr_r \left[B(x, S_L) \oplus \mathcal{A}(f(x)||S_L \oplus e^i) = x_i \right] \\
=\ & \Pr_r \left[\mathcal{A}(f(x)||S_L \oplus e^i) = B(x, S_L \oplus e^i) \right] \\
\geq\ & \frac{1}{2} + \frac{\epsilon(n)}{2}
\end{aligned}$$

For the same $\{b_\ell\}_\ell$ and $x \in G$, let Y_i^L be the indicator random variable that $x_i^L = x_i$. Notice that $Y_i^{\emptyset}, \cdots, Y_i^{[\log T]}$ are pairwise independent and $\Pr[Y_i^L = 1] = \Pr[x_i^L = x_i] \geq \frac{1}{2} + \frac{\epsilon(n)}{2}$.

$$\begin{aligned}
\Pr[x_i' \neq x_i] &= \Pr \left[\sum_{L \subseteq [\log T]} Y_i^L \leq \frac{T}{2} \right] \\
&= \Pr \left[\sum_{L \subseteq [\log T]} Y_i^L - \left(\frac{1}{2} + \frac{\epsilon(n)}{2} \right) T \leq \frac{T}{2} - \left(\frac{1}{2} + \frac{\epsilon(n)}{2} \right) T \right] \\
&\leq \Pr \left[\left| \sum_{L \subseteq [\log T]} Y_i^L - \left(\frac{1}{2} + \frac{\epsilon(n)}{2} \right) T \right| \geq \frac{\epsilon(n)}{2} T \right] \\
&\qquad \text{(By Theorem 2.4)} \\
&\leq \frac{1}{4 \left(\frac{\epsilon(n)}{2} \right)^2 T} = \frac{1}{2n}.
\end{aligned}$$

Completing the proof, we have that:

$$\begin{aligned}
&\Pr_{x,r} [\mathcal{B}(1^{2n}, g(x,r)) = (x,r)] \\
\geq\ & \Pr_{\{b_\ell, s_\ell\}_\ell} [F] \cdot \Pr_x [x \in G] \cdot \Pr[x_1' = x_1, \cdots x_n' = x_n \mid x \in G \ \wedge \ F] \\
\geq\ & \frac{1}{T} \cdot \frac{\epsilon(n)}{2} \cdot \left(1 - \sum_{i=1}^{n} \Pr[x_i' \neq x_i \mid x \in G \ \wedge \ F] \right) \\
\geq\ & \frac{\epsilon(n)^2}{2n} \cdot \frac{\epsilon(n)}{2} \cdot \left(1 - n \cdot \frac{1}{2n} \right) = \frac{\epsilon(n)^3}{8n}
\end{aligned}$$

Pairwise Independence and Chebyshev's Inequality. For the sake of completeness, we prove the Chebyshev's Inequality here.

Definition 2.4 (Pairwise Independence). *A collection of random variables $\{X_1, \cdots, X_m\}$ is said to be* pairwise independent *if for every pair of random variables $(X_i, X_j), i \neq j$ and every pair of values (v_i, v_j), it holds that*

$$\Pr[X_i = v_i, X_j = v_j] = \Pr[X_i = v_i] \Pr[X_j = v_j]$$

Theorem 2.4 (Chebyshev's Inequality). *Let $X_1, \ldots, X_m$ be pairwise independent and identically distributed binary random variables. In particular, for every $i \in [m]$, $\Pr[X_i = 1] = p$ for some $p \in [0,1]$ and $\Pr[X_i = 0] = 1 - p$. Then it holds that*

$$\Pr \left[\left| \sum_{i=1}^{m} X_i - pm \right| \geq \delta m \right] \leq \frac{1}{4\delta^2 m}.$$

Proof. Let $Y = \sum_i X_i$. Then

$$\begin{aligned}
&\Pr \left[\left| \sum_{i=1}^{m} X_i - pm \right| > \delta m \right] \\
=\ & \Pr \left[\left(\sum_{i=1}^{m} X_i - pm \right)^2 > \delta^2 m^2 \right] \\
\leq\ & \frac{\mathbb{E}\left[|Y - pm|^2 \right]}{\delta^2 m^2} = \frac{\mathrm{Var}(Y)}{\delta^2 m^2}
\end{aligned}$$

Observe that

$$\begin{aligned}
\mathrm{Var}(Y) &= \mathbb{E}\left[Y^2 \right] - (\mathbb{E}[Y])^2 \\
&= \sum_{i=1}^{m} \sum_{j=1}^{m} \left(\mathbb{E}\left[X_i X_j \right] - \mathbb{E}\left[X_i \right] \mathbb{E}\left[X_j \right] \right)
\end{aligned}$$

By pairwise independence, for $i \neq j$, $\mathbb{E}\left[X_i X_j \right] = \mathbb{E}\left[X_i \right] \mathbb{E}\left[X_j \right]$.

$$\begin{aligned}
&= \sum_{i=1}^{m} \mathbb{E}\left[X_i^2 \right] - \mathbb{E}\left[X_i \right]^2 \\
&= mp(1-p).
\end{aligned}$$

Hence

$$\Pr \left[\left| \sum_{i=1}^{m} X_i - pm \right| \geq \delta m \right] \leq \frac{mp(1-p)}{\delta^2 m^2} \leq \frac{1}{\delta^2 m}.$$

□

□

Exercises

Exercise 2.1. *If $\mu(\cdot)$ and $\nu(\cdot)$ are negligible functions then show that $\mu(\cdot) \cdot \nu(\cdot)$ is a negligible function.*

Exercise 2.2. *If $\mu(\cdot)$ is a negligible function and $f(\cdot)$ is a function polynomial in its input then show that $\mu(f(\cdot))$*[4] *are negligible functions.*

Exercise 2.3. *Prove that the existence of one-way functions implies $P \neq NP$.*

Exercise 2.4. *Prove that there is no one-way function $f : \{0,1\}^n \to \{0,1\}^{\lfloor \log_2 n \rfloor}$.*

Exercise 2.5. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be any one-way function then is $f'(x) \overset{def}{=} f(x) \oplus x$ necessarily one-way?*

Exercise 2.6. *Prove or disprove: If $f : \{0,1\}^n \to \{0,1\}^n$ is a one-way function, then $g : \{0,1\}^n \to \{0,1\}^{n-\log n}$ is a one-way function, where $g(x)$ outputs the $n - \log n$ higher order bits of $f(x)$.*

Exercise 2.7. *Explain why the proof of Theorem 2.1 fails if the attacker $\mathcal{A}$ in Figure 2.2 sets $i = 1$ and not $i \overset{\$}{\leftarrow} \{1, 2, \cdots, q\}$.*

Exercise 2.8. *Given a (strong) one-way function construct a weak one-way function that is not a (strong) one-way function.*

Exercise 2.9. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a weak one-way permutation (a weak one way function that is a bijection). More formally, f is a PPT computable one-to-one function such that $\exists$ a constant $c > 0$ such that $\forall$ non-uniform PPT machine A and $\forall$ sufficiently large n we have that:*

$$\Pr_{x,A}[A(f(x)) \notin f^{-1}(f(x))] > \frac{1}{n^c}$$

Show that $g(x) = f^T(x)$ is not a strong one way permutation. Here f^T denotes the T times self composition of f and T is a polynomial in n.

Interesting follow up reading if interested: With some tweaks the function above can be made a strong one-way permutation using explicit constructions of expander graphs. See Section 2.6 in `http://www.wisdom.weizmann.ac.il/~oded/PSBookFrag/part2N.ps`

[4] Assume that μ and f are such that $\mu(f(\cdot))$ takes inputs from $\mathbb{Z}^+$ and outputs values in $[0,1]$.

3
Pseudorandomness

In this chapter, our objective is to transform a small amount of entropy into a distribution that closely resembles randomness. The idea is to start with a small amount of entropy, known as the “seed", and use a deterministic process to generate a new distribution that appears “indistinguishable" from random. However, before we dive into the specifics of how to achieve this, we need to clarify what we mean by “indistinguishable."

3.1 *Statistical Indistinguishability*

The first definition of indistinguishability we will focus on is that of statistical indistinguishability. It turns out that defining what it means for two distributions to be indistinguishable by an adversary is tricky. In particular, it is tricky to define indistinguishability for a single pair of distributions because the length of the output of a random variable is a constant. Therefore, in order for our definition to make sense, we will work with collections of distributions, called *ensembles*

Definition 3.1 (Ensemble of Probability Distributions). *An* ensemble *of probability distributions is a sequence of random variables* $\{X_n\}_{n \in \mathbb{N}}$.

In this definition, n is a parameter. Sometimes, we write $\{X_n\}_n$ or even simply X_n, when it is clear from context that we are talking about an ensemble.

Definition 3.2 (Statistical Indistinguishability). *Two ensembles of probability distributions $\{X_n\}_n$ and $\{Y_n\}_n$ are said to be* statistically indistinguishable *if for all adversaries $\mathcal{A}$, the quantities*

$$p(n) := \Pr[\mathcal{A}(X_n) = 1] = \sum_x \Pr[X_n = x] \Pr[\mathcal{A}(1^n, x) = 1]$$

and

$$q(n) := \Pr[\mathcal{A}(Y_n) = 1] = \sum_y \Pr[Y_n = y] \Pr[\mathcal{A}(1^n, y) = 1]$$

differ by a negligible amount. In particular, the ensembles are said to be statistically indistinguishable if

$$\Delta_{\mathcal{A}}(n) = |p(n) - q(n)| = |\Pr[\mathcal{A}(X_n) = 1] - \Pr[\mathcal{A}(Y_n) = 1]|$$

is negligible in n. This equivalence is denoted by

$$\{X_n\}_n \approx_S \{Y_n\}_n$$

Note that our attacker in this scenario is not computationally bounded, as is usual[1]. We also do not require the ensemble to be efficiently samplable.

This definition is closely related to the concept of the *statistical distance* between two probability distributions.

Definition 3.3 (Statistical Distance). *The* statistical distance *between two distributions X and Y is defined as*

$$SD(X, Y) = \frac{1}{2} \sum_{v \in S} |\Pr[X_n = v] - \Pr[Y_n = v]|$$

where $S = Support(X_n) \cup Support(Y_n)$.

In fact, we can show that $\Delta_{\mathcal{A}}(n) \leq SD(X_n, Y_n)$.

Lemma 3.1 (Relationship between SD and $\Delta_{\mathcal{A}}$). *For any adversary $\mathcal{A}$,*

$$\Delta_{\mathcal{A}}(n) \leq SD(X_n, Y_n)$$

Proof. Let Ω be the sample space for X_n and Y_n.

Let $T = \{v \in \Omega | \Pr[v \leftarrow X_n] > \Pr[v \leftarrow Y_n]\}$.

First, we will prove that $SD(X_n, Y_n) = \sum\limits_{v \in \Omega} |\Pr[v \leftarrow X_n] - \Pr[v \leftarrow Y_n]|$.

[1] Statistical indistinguishability is a very strong requirement, and it makes use of a very powerful adversary, so it will serve mostly as an illustrative example.

$$
\begin{aligned}
\sum_{v\in\Omega} \Pr[v \leftarrow X_n] &= \sum_{v\in\Omega} \Pr[v \leftarrow Y_n] = 1 \\
\sum_{v\in T} \Pr[v \leftarrow X_n] + \sum_{v\in\Omega\setminus T} \Pr[v \leftarrow X_n] &= \sum_{v\in T} \Pr[v \leftarrow Y_n] + \sum_{v\in\Omega\setminus T} \Pr[v \leftarrow Y_n] \\
\sum_{v\in T} (\Pr[v \leftarrow X_n] - \Pr[v \leftarrow Y_n]) &= \sum_{v\in\Omega\setminus T} (\Pr[v \leftarrow Y_n] - \Pr[v \leftarrow X_n]) \\
\sum_{v\in T} |\Pr[v \leftarrow X_n] - \Pr[v \leftarrow Y_n]| &= \sum_{v\in\Omega\setminus T} |\Pr[v \leftarrow Y_n] - \Pr[v \leftarrow X_n]| \\
\sum_{v\in T} |\Pr[v \leftarrow X_n] - \Pr[v \leftarrow Y_n]| &= \sum_{v\in\Omega\setminus T} |\Pr[v \leftarrow X_n] - \Pr[v \leftarrow Y_n]| \\
\sum_{v\in\Omega} |\Pr[v \leftarrow X_n] - \Pr[v \leftarrow Y_n]| &= \sum_{v\in T} |\Pr[v \leftarrow X_n] - \Pr[v \leftarrow Y_n]| \\
&\quad + \sum_{v\in\Omega\setminus T} |\Pr[v \leftarrow X_n] - \Pr[v \leftarrow Y_n]| \\
2SD(X_n, Y_n) &= 2 \cdot \sum_{v\in T} |\Pr[v \leftarrow X_n] - \Pr[v \leftarrow Y_n]| \\
SD(X_n, Y_n) &= \sum_{v\in T} |\Pr[v \leftarrow X_n] - \Pr[v \leftarrow Y_n]|
\end{aligned}
$$

Now we will show the main result of the lemma.

$$
\begin{aligned}
\Delta_{\mathcal{A}}(n) &= |\Pr[\mathcal{A}(X_n) = 1] - \Pr[\mathcal{A}(Y_n) = 1]| \\
&= |\sum_{v\in\Omega} (\Pr[\mathcal{A}(v) = 1] \cdot \Pr[v \leftarrow X_n]) - (\Pr[\mathcal{A}(v) = 1] \cdot \Pr[v \leftarrow Y_n])| \\
&= |\sum_{v\in\Omega} \Pr[\mathcal{A}(v) = 1] \cdot (\Pr[v \leftarrow X_n]) - \Pr[v \leftarrow Y_n])| \\
&= |\sum_{v\in T} \Pr[\mathcal{A}(v) = 1] \cdot (\Pr[v \leftarrow X_n]) - \Pr[v \leftarrow Y_n]) \\
&\quad + \sum_{v\in\Omega\setminus T} \Pr[\mathcal{A}(v) = 1] \cdot (\Pr[v \leftarrow X_n]) - \Pr[v \leftarrow Y_n])| \\
&= \sum_{v\in T} \Pr[\mathcal{A}(v) = 1] \cdot (\Pr[v \leftarrow X_n]) - \Pr[v \leftarrow Y_n]) \\
&\quad + \sum_{v\in\Omega\setminus T} \Pr[\mathcal{A}(v) = 1] \cdot (\Pr[v \leftarrow X_n]) - \Pr[v \leftarrow Y_n]) \\
&= \sum_{v\in T} \Pr[\mathcal{A}(v) = 1] \cdot |\Pr[v \leftarrow X_n]) - \Pr[v \leftarrow Y_n]| \\
&\quad + \sum_{v\in\Omega\setminus T} \Pr[\mathcal{A}(v) = 1] \cdot |\Pr[v \leftarrow X_n]) - \Pr[v \leftarrow Y_n]| \\
&\leq \sum_{v\in T} |\Pr[v \leftarrow X_n] - \Pr[v \leftarrow Y_n]| \\
&= SD(X_n, Y_n)
\end{aligned}
$$

□

3.2 Computational Indistinguishability

We now turn to a more reasonable definition of indistinguishability. In particular, this definition imposes the usual computational limits on the adversary $\mathcal{A}$. It also requires that the ensembles of distributions in question be efficiently samplable. Besides those changes, however, the definition of *computational indistinguishability* is quite similar to that of *statistical indistinguishability*.

Definition 3.4 (Computational Indistinguishability). *Two ensembles of probability distributions $\{X_n\}_n$ and $\{Y_n\}_n$ (which are* samplable *in time polynomial in n) are said to be* computationally indistinguishable *if for all (non-uniform) PPT adversaries $\mathcal{A}$, the quantities*

$$p(n) := \Pr[\mathcal{A}(1^n, X_n) = 1] = \sum_x \Pr[X_n = x] \Pr[\mathcal{A}(1^n, x) = 1]$$

and

$$q(n) := \Pr[\mathcal{A}(1^n, Y_n) = 1] = \sum_y \Pr[Y_n = y] \Pr[\mathcal{A}(1^n, y) = 1]$$

differ by a negligible amount; i.e. $|p(n) - q(n)|$ is negligible in n. This equivalence is denoted by

$$\{X_n\}_n \approx_C \{Y_n\}_n$$

However, since this is the main form of indistinguishability that we are concerned with, we will simply write

$$\{X_n\}_n \approx \{Y_n\}_n$$

We now prove some properties of computationally indistinguishable ensembles that will be useful later on.

Lemma 3.2 (Sunglass Lemma). *If $\{X_n\}_n \approx \{Y_n\}_n$ and P is a PPT machine, then*

$$\{P(X_n)\}_n \approx \{P(Y_n)\}_n$$

Proof. Consider an adversary $\mathcal{A}$ that can distinguish $\{P(X_n)\}_n$ from $\{P(Y_n)\}_n$ with non-negligible probability. Then the adversary $\mathcal{A} \circ P$ can distinguish $\{X_n\}_n$ from $\{Y_n\}_n$ with the same non-negligible probability. Since P and $\mathcal{A}$ are both PPT machines, the composition is also a PPT machine. This proves the contrapositive of the lemma. □

The name of the lemma comes from the idea that if two objects are indistinguishable without putting on sunglasses, then they should remain indistinguishable after putting on sunglasses.

Lemma 3.3 (Multicopy Lemma). *For a polynomial $t : \mathbb{Z}^+ \to \mathbb{Z}^+$ let the t-product of $\{Z_n\}_n$ be*

$$\{Z_n^{(1)}, Z_n^{(2)}, \ldots, Z_n^{(t(n))}\}_n$$

where the $Z_n^{(i)}$s are independent copies of Z_n. If

$$\{X_n\}_n \approx \{Y_n\}_n$$

then

$$\{X_n^{(1)}, \ldots, X_n^{(t)}\}_n \approx \{Y_n^{(1)}, \ldots, Y_n^{(t)}\}_n$$

as well.

Intuitively, if you can't tell apart a red ball and a blue ball, then you can't tell apart multiple copies of the red and blue balls.

Proof. We proceed by what is known as a hybrid argument. Consider the set of tuple random variables

$$H_n^{(i,t)} = (Y_n^{(1)}, \ldots, Y_n^{(i)}, X_n^{(i+1)}, X_n^{(i+2)}, \ldots, X_n^{(t)})$$

for integers $0 \leq i \leq t$. For instance, when $i = 0$:

$$H_n^{(0,t)} = (X_n^{(1)}, X_n^{(2)}, \ldots, X_n^{(t)}) = \overline{X}_n$$

Similarly, when $i = t$:

$$H_n^{(t,t)} = (Y_n^{(1)}, Y_n^{(2)}, \ldots, Y_n^{(t)}) = \overline{Y}_n$$

Assume, for the sake of contradiction, that there is a PPT adversary $\mathcal{A}$ that can distinguish between $\{H_n^{(0,t)}\}_n$ and $\{H_n^{(t,t)}\}_n$ with non-negligible probability difference $\varepsilon(n)$. Suppose that $\mathcal{A}$ returns 1 with probability P_i when it runs on samples from $H_n^{(i,t)}$. That is, $P_i = \Pr[\mathcal{A}(H_n^{(i,t)} = 1)]$ By definition, $|P_0 - P_t| \geq \varepsilon(n)$.

Using the common add-one-subtract-one trick, we can find that

$$\begin{aligned} |P_0 - P_t| &= |P_0 - P_1 + P_1 - P_2 + \ldots + P_{t-1} - P_t| \\ &= |(P_0 - P_1) + (P_1 - P_2) + \ldots + (P_{t-1} - P_t)| \\ &\leq |P_0 - P_1| + |P_1 - P_2| + \ldots + |P_{t-1} - P_t| \end{aligned}$$

Since $|P_0 - P_t| \geq \varepsilon(n)$, it follows that $|P_0 - P_1| + |P_1 - P_2| + \ldots + |P_{t-1} - P_t| \geq \varepsilon(n)$. Then there must exist some index k for which

$$|P_k - P_{k+1}| \geq \frac{\varepsilon(n)}{t}$$

Note that $\frac{\varepsilon(n)}{t}$ is non-negligible because t is polynomial. This implies that $\{H_n^{(k,t)}\}_n$ and $\{H_n^{(k+1,t)}\}_n$ are distinguishable.

Using this information, we can construct an adversary $\mathcal{B}$ that can distinguish X_n from Y_n. Given an input Z_n, which is either X_n or Y_n, $\mathcal{B}$ works as follows:

$$\mathcal{B}(Z_n) = \mathcal{A}(X_1, ..., X_{k-1}, Z, Y_{k+1}, ..., Y_t)$$

By the argument above, for some value[2] of k, this computation gives $|\Pr[\mathcal{B}(X_n) = 1] - \Pr[\mathcal{B}(Y_n) = 1]| \geq \frac{\varepsilon(n)}{t}$.

This is a contradiction. □

Intuitively, the idea behind proofs by hybrid argument is to create a chain of polynomially many hybrids such that the hybrids are pairwise indistinguishable at each step. Visually:

$$H_n^{(0,t)} \approx H_n^{(1,t)} \approx H_n^{(2,t)} \approx ... \approx H_n^{(t-1,t)} \approx H_n^{(t,t)}$$

This implies that

$$H_n^{(0,t)} \approx H_n^{(t,t)}$$

which is the same thing as saying that

$$\overline{X}_n \approx \overline{Y}_n$$

□

3.3 *Pseudorandom Generators*

Now, we can define pseudorandom generators, which intuitively generates a polynomial number of bits that are computationally indistinguishable from being uniformly random:

Definition 3.5. *A function $G : \{0,1\}^n \to \{0,1\}^{n+m}$ with $m = poly(n)$ is called a* pseudorandom generator *if*

- *G is computable in polynomial time.*
- *$U_{n+m} \approx G(U_n)$, where U_k denotes the uniform distribution on $\{0,1\}^k$.*

3.3.1 *PRG Extension*

In this section we show that any pseudorandom generator that produces one bit of randomness can be extended to create a polynomial number of bits of randomness.

Construction 3.1. *Given a PRG $G : \{0,1\}^n \to \{0,1\}^{n+1}$, we construct a new PRG $F : \{0,1\}^n \to \{0,1\}^{n+l}$ as follows (l is polynomial in n).*

(a) Input: $S_0 \xleftarrow{\$} \{0,1\}^n$.

(b) $\forall i \in [l] = \{1, 2, \cdots, l\}$, $(\sigma_i, S_i) := G(S_{i-1})$, where $\sigma_i \in \{0,1\}, S_i \in \{0,1\}^n$.

(c) Output: $\sigma_1\sigma_2 \cdots \sigma_l S_l$.

[2] $\mathcal{B}$ is non-uniform, so it can “know" which value of k it should use.

Theorem 3.1. *The function F constructed above is a PRG.*

Proof. We prove this by hybrid argument. Define the hybrid H_i as follows.

(a) Input: $S_0 \xleftarrow{\$} \{0,1\}^n$.

(b) $\sigma_1, \sigma_2, \cdots, \sigma_i \xleftarrow{\$} \{0,1\}$, $S_i \leftarrow S_0$.
$\forall j \in \{i+1, i+2, \cdots, l\}$, $(\sigma_j, S_j) := G(S_{j-1})$, where $\sigma_j \in \{0,1\}, S_j \in \{0,1\}^n$.

(c) Output: $\sigma_1 \sigma_2 \cdots \sigma_l S_l$.

Note that $H_0 \equiv F$, and $H_l \equiv U_{n+l}$.

Assume for the sake of contradiction that there exits a non-uniform PPT adversary $\mathcal{A}$ that can distinguish H_0 form H_l. Define $\epsilon_i := \Pr[\mathcal{A}(1^n, H_i) = 1]$ for $i = 0, 1, \cdots, l$. Then there exists a non-negligible function $v(n)$ such that $|\epsilon_0 - \epsilon_l| \geq v(n)$. Since

$$|\epsilon_0 - \epsilon_1| + |\epsilon_1 - \epsilon_2| + \cdots + |\epsilon_{l-1} - \epsilon_l| \geq |\epsilon_0 - \epsilon_l| \geq v(n),$$

there exists $k \in \{0, 1, \cdots, l-1\}$ such that

$$|\epsilon_k - \epsilon_{k+1}| \geq \frac{v(n)}{l}.$$

l is polynomial in n, hence $\frac{v(n)}{l}$ is also a non-negligible function. That is to say, $\mathcal{A}$ can distinguish H_k from H_{k+1}. Then we use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that can distinguish U_{n+1} from $G(U_n)$ (which leads to a contradiction): On input $T \in \{0,1\}^{n+1}$ (T could be either from U_{n+1} or $G(U_n)$), $\mathcal{B}$ proceeds as follows:

- $\sigma_1, \sigma_2, \cdots, \sigma_k \xleftarrow{\$} \{0,1\}$, $(\sigma_{k+1}, S_{k+1}) \leftarrow T$.
- $\forall j \in \{k+2, k+3, \cdots, l\}$, $(\sigma_j, S_j) := G(S_{j-1})$, where $\sigma_j \in \{0,1\}, S_j \in \{0,1\}^n$.
- Output: $\mathcal{A}(1^n, \sigma_1 \sigma_2 \cdots \sigma_l S_l)$.

First, since $\mathcal{A}$ and G are both PPT computable, $\mathcal{B}$ is also PPT computable.

Second, if $T \leftarrow G(U_n)$, then $\sigma_1 \sigma_2 \cdots \sigma_l S_l$ is the output of H_k; if $T \xleftarrow{\$} U_{n+1}$, then $\sigma_1 \sigma_2 \cdots \sigma_l S_l$ is the output of H_{k+1}. Hence

$$\begin{aligned} & \big| \Pr[\mathcal{B}(1^n, G(U_n)) = 1] - \Pr[\mathcal{B}(1^n, U_{n+1}) = 1] \big| \\ = & \big| \Pr[\mathcal{A}(1^n, H_k) = 1] - \Pr[\mathcal{A}(1^n, H_{k+1}) = 1] \big| \\ = & |\epsilon_k - \epsilon_{k+1}| \geq \frac{v(n)}{l}. \end{aligned}$$

□

3.3.2 PRG from OWP (One-Way Permutations)

In this section we show how to construct pseudorandom generators under the assumption that one-way permutations exist.

Construction 3.2. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a OWP. We construct $G : \{0,1\}^{2n} \to \{0,1\}^{2n+1}$ as*

$$G(x,r) = f(x)||r||B(x,r),$$

where $x, r \in \{0,1\}^n$, and $B(x,r)$ is a hard concentrate bit for the function $g(x,r) = f(x)||r$.

Remark 3.1. *The hard concentrate bit $B(x,r)$ always exists. Recall Theorem 2.3,*

$$B(x,r) = \left(\sum_{i=1}^{n} x_i r_i\right) \mod 2$$

is a hard concentrate bit.

Theorem 3.2. *The G constructed above is a PRG.*

Proof. Assume for the sake of contradiction that G is not PRG. We construct three ensembles of probability distributions:

$$H_0 := G(U_{2n}) = f(x)||r||B(x,r), \text{ where } x, r \overset{\$}{\leftarrow} \{0,1\}^n;$$

$$H_1 := f(x)||r||\sigma, \text{ where } x, r \overset{\$}{\leftarrow} \{0,1\}^n, \sigma \overset{\$}{\leftarrow} \{0,1\};$$

$$H_2 := U_{2n+1}.$$

Since G is not PRG, there exists a non-uniform PPT adversary $\mathcal{A}$ that can distinguish H_0 from H_2. Since f is a permutation, H_1 is uniformly distributed in $\{0,1\}^{2n+1}$, i.e., $H_1 \equiv H_2$. Therefore, $\mathcal{A}$ can distinguish H_0 from H_1, that is, there exists a non-negligible function $v(n)$ satisfying

$$\big|\Pr[\mathcal{A}(H_0) = 1] - \Pr[\mathcal{A}(H_1) = 1]\big| \geq v(n).$$

Next we will construct an adversary $\mathcal{B}$ that "breaks" the hard concentrate bit (which leads to a contradiction). Define a new ensemble of probability distribution

$$H_1' = f(x)||r||(1 - B(x,r)), \text{ where } x, r \overset{\$}{\leftarrow} \{0,1\}^n.$$

Then we have

$$\begin{aligned}\Pr[\mathcal{A}(H_1) = 1] =& \Pr[\sigma = B(x,r)] \Pr[A(H_0) = 1] + \Pr[\sigma = 1 - B(x,r)] \Pr[A(H_1') = 1] \\ =& \frac{1}{2} \Pr[A(H_0) = 1] + \frac{1}{2} \Pr[A(H_1') = 1].\end{aligned}$$

Hence

$$\begin{aligned}
&\Pr[A(H_1)=1]-\Pr[A(H_0)=1]=\frac{1}{2}\Pr[A(H_1')=1]-\frac{1}{2}\Pr[A(H_0)=1],\\
&\frac{1}{2}\left|\Pr[A(H_0)=1]-\Pr[A(H_1')=1]\right|=|\Pr[A(H_1)=1]-\Pr[A(H_0)=1]|\geq v(n),\\
&\left|\Pr[A(H_0)=1]-\Pr[A(H_1')=1]\right|\geq 2v(n).
\end{aligned}$$

Without loss of generality, we assume that

$$\Pr[A(H_0)=1]-\Pr[A(H_1')=1]\geq 2v(n).$$

Then we construct $\mathcal{B}$ as follows:

$$\mathcal{B}(f(x)||r):=\begin{cases}\sigma, & \text{if } \mathcal{A}(f(x)||r||\sigma)=1\\ 1-\sigma, & \text{if } \mathcal{A}(f(x)||r||\sigma)=0\end{cases},$$

where $\sigma \overset{\$}{\leftarrow} \{0,1\}$. Then we have

$$\begin{aligned}
&\Pr[\mathcal{B}(f(x)||r)=B(x,r)]\\
=&\Pr[\sigma=B(x,r)]\Pr[\mathcal{A}(f(x)||r||\sigma)=1|\sigma=B(x,r)]+\\
&\Pr[\sigma=1-B(x,r)]\Pr[\mathcal{A}(f(x)||r||\sigma)=0|\sigma=1-B(x,r)]+\\
=&\frac{1}{2}\big(\Pr[\mathcal{A}(f(x)||r||B(x,r))=1]+1-\Pr[\mathcal{A}(f(x)||r||1-B(x,r))=1]\big)\\
=&\frac{1}{2}+\frac{1}{2}\big(\Pr[A(H_0)=1]-\Pr[A(H_1')=1]\big)\\
\geq&\frac{1}{2}+v(n).
\end{aligned}$$

This contradicts the fact that B must be a hardness concentrate bit.

□

3.4 Pseudorandom Functions

In this section, we first define pseudorandom functions, and then show how to construct a pseudorandom function from a pseudorandom generator.

Considering the set of all functions $f : \{0,1\}^n \rightarrow \{0,1\}^n$, there are $(2^n)^{2^n}$ of them. To describe a random function in this set we need $n \cdot 2^n$ bits. Intuitively, a pseudorandom function is one that cannot be distinguished from a random one, but needs much fewer bits (e.g., polynomial in n) to be described. Note that we restrict the distinguisher to only being allowed to ask the function poly(n) times and decide whether it is random or pseudorandom.

3.4.1 Definitions

Definition 3.6 (Function Ensemble). *A* function ensemble *is a sequence of random variables $F_1, F_2, \cdots, F_n, \cdots$ denoted as $\{F_n\}_{n\in\mathbb{N}}$ such that F_n assumes values in the set of functions mapping n-bit input to n-bit output.*

Although we will only focus on the functions where the input and output bit-length is the same, the definition can be generalized to functions mapping n-bit inputs to m-bit outputs as $\{F_{n,m}\}_{n,m\in\mathbb{N}}$.

Definition 3.7 (Random Function Ensemble)**.** *We denote a random function ensemble by* $\{R_n\}_{n\in\mathbb{N}}$.

A sampling of the random variable R_n requires $n \cdot 2^n$ bits to describe.

Definition 3.8 (Efficiently Computable Function Ensemble)**.** *A function ensemble is called* efficiently computable *if*

(a) ***Succinct***: *$\exists$ a PPT algorithm I and a mapping ϕ from strings to functions such that $\phi(I(1^n))$ and F_n are identically distributed. Note that we can view the output of $I(\cdot)$ as the description of the function.*

(b) ***Efficient***: *$\exists$ a poly-time machine V such that $V(i,x) = f_i(x)$ for every $x \in \{0,1\}^n$, where i is in the range of $I(1^n)$, and $f_i = \phi(i)$.*

Note that the succinctness condition implies that a sample from F_n can be equivalently generated by first sampling a random string k from $\{0,1\}^n$, and then outputting f_k. Here k is often called the "key" of the function[3]. More generally, the key can be a string of length m where n is polynomial in m; here I uses a random tape of length m and outputs n bits.

Definition 3.9 (Pseudorandom Function Ensemble)**.** *A function ensemble $F = \{F_n\}_{n\in\mathbb{N}}$ is* pseudorandom *if for every non-uniform PPT oracle adversary $\mathcal{A}$, there exists a negligible function $\epsilon(n)$ such that*

$$\left|\Pr[\mathcal{A}^{F_n}(1^n) = 1] - \Pr[\mathcal{A}^{R_n}(1^n) = 1]\right| \le \epsilon(n).$$

Here by saying "oracle" it means that $\mathcal{A}$ has "oracle access" to a (fixed) function (in our definition, the function is a sampling of F_n or R_n), and each call to that function costs 1 unit of time.

Note that we will only consider efficiently computable pseudorandom ensembles in the following. Therefore, each function in F_n is defined by a PRF key $k \in \{0,1\}^n$.

3.4.2 *Construction of PRF from PRG*

Construction 3.3. *Given a PRG $G : \{0,1\}^n \to \{0,1\}^{2n}$, let $G_0(x)$ be the first n bits of $G(x)$, $G_1(x)$ be the last n bits of $G(x)$. We construct $F^{(K)} : \{0,1\}^n \to \{0,1\}^n$ as follows.*

$$F_n^{(K)}(x_1 x_2 \cdots x_n) := G_{x_n}(G_{x_{n-1}}(\cdots (G_{x_1}(K))\cdots)),$$

where $K \in \{0,1\}^n$ is the key to the pseudorandom function. In Figure 3.1, $i = K$.

[3] An efficiently computable function requires only n bits (the key) to describe, while a random function requires $n.2^n$ bits.

The construction can be viewed as a binary tree of depth n, as shown in Figure 3.1[4].

Theorem 3.3. *The function ensemble $\{F_n\}_{n\in\mathbb{N}}$ constructed above is pseudorandom.*

Proof. Assume for the sake of contradiction that $\{F_n\}_{n\in\mathbb{N}}$ is not a PRF. Then there exists a non-uniform PPT oracle adversary $\mathcal{A}$ that can distinguish $\{F_n\}_{n\in\mathbb{N}}$ from $\{R_n\}_{n\in\mathbb{N}}$. Below, via a hybrid argument, we prove that this contradicts the fact that G is a PRG; we will construct an adversary $\mathcal{B}$ that can distinguish between a sample from U_{2n} and $G(U_n)$. We will prove for a fixed n, and the proof can be easily extended to all $n \in \mathbb{N}$.

Hybrids. Consider the sequence of hybrids H_i for $i \in \{0, 1, \cdots, n\}$ where the hybrid i is defined as follows:

$$H_i^{(K_i)}(x_1x_2\dots x_n) := G_{x_n}(G_{x_{n-1}}(\cdots(G_{x_{i+1}}(K_i(x_1\dots x_{i-1}x_i)))\cdots)),$$

where K_i is a random function from $\{0,1\}^i$ to $\{0,1\}^n$. Intuitively, hybrid H_i corresponds to a binary tree of depth n where the nodes of levels 0 to i correspond to random values and the nodes at levels $i+1$ to n correspond to pseudorandom values. By inspection, observe that hybrids H_0 and H_n are identical to a pseudorandom function and a random function, respectively. Note that we cannot yet reduce the computational indistinguishability of H_i and H_{i+1} to security of the PRG G because the adversary can make multiple oracle queries at different inputs.

Sub-hybrids. We show that H_i and H_{i+1} are indistinguishable by considering a sequence of sub-hybrids $H_{i,j}$ for $j \in \{0, \dots q\}$, where q is the number of oracle queries made by $\mathcal{A}$[5]. Intuitively, with each sub-hybrid $H_{i,j}$, at level $i+1$ in the tree, we will fix the first j oracle queries made by $\mathcal{A}$ to be output of random functions and the rest to be output of PRG. Let $R_i : \{0,1\}^i \to \{0,1\}^n$ and $S_i : \{0,1\}^{i+1} \to \{0,1\}^n$ be two random functions. We define sub-hybrid $H_{i,j}^{(R_i,S_i)}(x_1x_2\dots x_n)$ algorithmically as follows:

1. Initialize a list $L \leftarrow \{\}$ to store the i-bit prefixes of the queries made by $\mathcal{A}$.

2. If $|L| < j$ or $(x_1 \dots x_i) \in L$[6]:
   (a) Set $y \leftarrow S_i(x_1 \dots x_i x_{i+1})$.
   (b) Append $(x_1 \dots x_i)$ to L.
   (c) For $a \in i+2 \dots n$: update $y \leftarrow G_{x_a}(y)$.

3. Else:
   (a) Set $y \leftarrow R_i(x_1 \dots x_i)$.

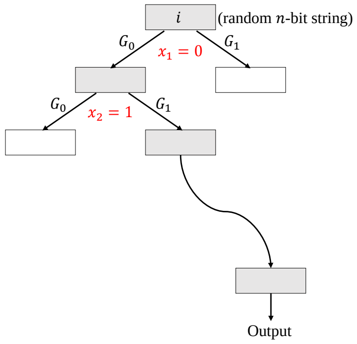


Figure 3.1: View the construction as a binary tree

[4] Algorithmically, $F_n^{(K)}(x)$ is computed as:

1. Set $y \leftarrow K \in \{0,1\}^n$.
2. For $i = 1 \dots n$: update $y \leftarrow G_{x_i}(y)$.
3. Output y.

[5] Observe that $\mathcal{A}$ can make at most polynomial in n oracle queries. Looking ahead, our outer adversary $\mathcal{B}$ can either take q as the max queries allowed to $\mathcal{A}$, or guess the number, and double the guess each time if it's an underestimate.

[6] Captures the first j queries or any query with repeated i-bit prefix to a previous query.

   (b) For $a \in i+1 \ldots n$: update $y \leftarrow G_{x_a}(y)$.

4. Output y.

Note that $H_{i,0}$ is the same as H_i and $H_{i,q}$ is the same as H_{i+1}. Since we assumed that $\mathcal{A}$ can distinguish between H_0 and H_n, by triangle inequality, there exists a i^*, j^* such that it can distinguish H_{i^*,j^*} and H_{i^*,j^*+1}. We now focus on these two sub-hybrids[7]. Consider the j^*+1-th query made by $\mathcal{A}$ (i.e. the first query where $|L| = j$). Observe that this query cannot have the same i-bit prefix as any of the previous queries. Because if it did, then the output distribution of the two hybrids would be identical, and that contradicts our assumption about $\mathcal{A}$'s distinguishing power. Therefore, the j^*+1-th query has to be a new query, and this query is the only place where the two hybrids differ.

Outer adversary $\mathcal{B}$. Now we are ready to construct our outer adversary $\mathcal{B}$ that can distinguish between U_{2n} and $G(U_n)$. $\mathcal{B}^{\mathcal{A},i^*,j^*}(1^n, z)$, where $z \in \{0,1\}^{2n}$ (z could be either from U_{2n} or $G(U_n)$) and we assume the knowledge of i^*, j^*[8], operates as follows:

1. Parse z as $z_0||z_1$, where $z_0, z_1 \in \{0,1\}^n$.

2. For all the oracle queries from $\mathcal{A}$ except the j^*+1-th query, respond as H_{i^*,j^*}[9].

3. For the j^*+1-th query $(x_1 \ldots x_n)$, do the following:

   (a) Set $y \leftarrow z_{x_{i^*+1}}$.

   (b) For $a \in i^*+2 \ldots n$: update $y \leftarrow G_{x_a}(y)$.

   (c) Respond with y.

4. Output whatever $\mathcal{A}$ outputs.

We assumed that $\mathcal{A}$ can distinguish between H_{i^*,j^*} and H_{i^*,j^*+1}, so by contrapositive of the Sunglass Lemma, $\mathcal{B}$ can distinguish between U_{2n} and $G(U_n)$. This contradicts that G is a PRG.

□

3.5 PRFs from DDH: Naor-Reingold PRF

We will now describe a PRF function family $F_n : \mathcal{K} \times \{0,1\}^n \to \mathbb{G}_n$ where DDH is assumed to be hard for $\{\mathbb{G}_n\}$ and $\mathcal{K}$ is the key space. The key for the PRF F_n will be $K = (h, u_1, \ldots u_n)$, where $u, u_0 \ldots u_n$ are sampled uniformly from $|\mathbb{G}_n|$, g is the generator of $\mathbb{G}_n$ and $h = g^u$. Compared to the previous construction (Theorem 3.3), there are two differences to note already: the key is polynomially longer and the output space is $\mathbb{G}_n$ instead of $\{0,1\}^n$.

[7] Looking ahead, the outer adversary $\mathcal{B}$ can guess i^*, j^*; total choices are bounded by polynomial in n. To simplify the proof, we will assume that $\mathcal{B}$ already knows this i^*, j^*.

[8] As mentioned before, it can be guessed with slight loss in distinguishing advantage.

[9] The outer adversary $\mathcal{B}$ runs a random function in polynomial time in n via lazy sampling. It generates a random output on a new input and caches responses to previous inputs.

$$F_n(K, x) = h^{\prod_i u_i^{x_i}}$$

Next, we will prove that the function F_n is a pseudo-random function or that $\{F_n\}$ is a pseudo-random function ensemble.[10]

Lemma 3.4. *Assuming the DDH Assumption (see Definition 1.7) for $\{\mathbb{G}_n\}$ is hard, we have that $\{F_n\}$ is a pseudorandom function ensemble.*

Proof. The proof of this lemma is similar to the proof of Theorem 3.3 except for some subtle differences that arise from number theory[11].

Let R_n be random function from $\{0,1\}^n \to \mathbb{G}_n$. Then we want to prove that for all non-uniform PPT adversaries $\mathcal{A}$ we have that:

$$\mu(n) = \left|\Pr[\mathcal{A}^{F_n}(1^n) = 1] - \Pr[\mathcal{A}^{R_n}(1^n) = 1]\right|$$

is a negligible function.

Hybrids. For the sake of contradiction, we assume that the function F_n is not pseudorandom. Next, towards a contradiction, we consider a sequence of hybrid functions $H_n^0 \dots H_n^n$. For $j \in \{0, \dots, n\}$, let $S_n^j : \{0,1\}^j \to \{0, 1, \dots, |\mathbb{G}_n| - 1\}$, then hybrid H_n^j is defined as[12]:

$$H_n^j((u, u_{j+1} \dots u_n), x) = \left(g^{S_n^j(x_1 \dots x_j)}\right)^{\prod_{i=j+1}^n u_i^{x_i}}$$

where $S_n^0(\cdot)$ is the constant function with output u. Observe that H_n^0 is the same as the function F_n and H_n^n is the same as the function R_n[13]. Thus, by a hybrid argument and triangle inequality, we conclude that there exists $j^* \in \{0, \dots n-1\}$, such that

$$\left|\Pr[\mathcal{A}^{H_n^{j^*}}(1^n) = 1] - \Pr[\mathcal{A}^{H_n^{j^*+1}}(1^n) = 1]\right|$$

is a non-negligible function. Now all we are left to show is that this implies an attacker that refutes the DDH assumption.

Sub-hybrids. The proof of this claim follows by a sequence of $q+1$ sub-hybrids $H_n^{j,0}, \dots, H_n^{j,q}$, where q is the (polynomially bounded by n) running time of $\mathcal{A}$. For the simplicity of exposition, we abuse the notation and denote $q(n)$ by q. Let $C_n^j : \{0,1\}^j \to \{0, \dots, |\mathbb{G}_n| - 1\}$ and $D_n^j : \{0,1\}^{j+1} \to \{0, \dots, |\mathbb{G}_n| - 1\}$ be two random functions, and $C_n^0(\cdot) = u$. We define sub-hybrid $H_n^{j,k}\big((u, u_{j+1} \dots u_n), (x_1 \dots x_n)\big)$ for $k \in \{0, \dots, q\}$ as follows:

1. Initialize a list $L \leftarrow \{\}$ to store the j-bit prefixes of the queries made by $\mathcal{A}$.

2. If $|L| < k$ or $(x_1 \cdots x_j) \in L$:

   (a) Set $y \leftarrow D_n^j(x_1 \dots x_{j+1})$.

   (b) Append $(x_1 \dots x_j)$ to L.

[10] Here, we require that adversary distinguish the function F_n from a random function from $\{0,1\}^n$ to $\mathbb{G}_n$. Note that the output range of the function is $\mathbb{G}_n$. Moreover, note that the distribution of random group elements in $\mathbb{G}_n$ might actually be far from uniformly random strings.

[11] At a high-level, we can no longer fix nodes in the same level of the tree arbitrarily. Fixing one node has implications for how other nodes will be changed. This is because we have a fixed basis in the key.

[12] Algorithmically, $H_n^j((u, u_{j+1} \dots u_n), x)$ is computed as:

1. Set $y \leftarrow S_n^j(x_1 \dots x_j)$.
2. For $i = j+1 \dots n$: update $y \leftarrow y \cdot u_i^{x_i}$.
3. Output g^y.

[13] A uniform group element is equivalently sampled by first sampling an exponent in the order of the group.

```
   (c) For $i = j+2 \dots n$: update $y \leftarrow y \cdot u_i^{x_i}$.
3. Else
   (a) Set $y \leftarrow C_n^j(x_1 \dots x_j)$.
   (b) For $i = j+1 \dots n$: update $y \leftarrow y \cdot u_i^{x_i}$.
4. Output $g^y$.
```

It is easy to see that $H_n^{j,0}$ is the same as H_n^j and $H_n^{j,q}$ is the same as H_n^{j+1}. Again, we use hybrid argument to conclude that there exists j^*, k^* such that $\mathcal{A}$ can distinguish between $H_n^{j^*,k^*}$ and $H_n^{j^*,k^*+1}$ with non-negligible probability. We now focus on these two sub-hybrids. Consider the k^*+1-th oracle query made by $\mathcal{A}$. Following an identical argument we used in the proof of Theorem 3.3, this query cannot be a repeat of a query made before, and this query is the only place where the two sub-hybrids differ.

Outer adversary $\mathcal{B}$. The construction of the outer adversary $\mathcal{B}$ is a bit different from the proof of Theorem 3.3. Intuitively, unlike Theorem 3.3, outer adversary cannot simply replace the k^*+1-th query with the DDH challenge in isolation from the rest of the queries made by $\mathcal{A}$. This is because the pseudorandom nodes in the tree are tied together by the DDH relation, and are not independent, i.e., all pseudorandom sibling nodes on the same level of the tree are set apart by a common exponent.

$\mathcal{B}$ gets as challenge either a DDH tuple $(g, A = g^a, B = g^b, C = g^{ab})$ or a uniform tuple $(g, A = g^a, B = g^b, C = g^c)$ where a, b, c are uniform in $\{0, \dots, |\mathbb{G}| - 1\}$. We construct $\mathcal{B}^{\mathcal{A},j^*,k^*}(1^n, (g, A, B, C))$ as follows:

```
1. Sample $u, u_{j^*+1}, \dots u_n$ uniformly from $\{0, \dots, |\mathbb{G}_n| - 1\}$.
2. For first $k^*$ queries from $\mathcal{A}$, respond as $H_n^{j^*,k^*}((u, u_{j^*+1}, \dots u_n), \cdot)$.
3. For the $k^*+1$-th query $(x_1 \dots x_n)$, do the following:
   (a) Set $y \leftarrow A$ if $x_{j^*+1} = 0$ and $y \leftarrow C$ if $x_{j^*+1} = 1$.
   (b) For $i = j^*+2 \dots n$: update $y \leftarrow y \cdot u_i^{x_i}$.
   (c) Output $g^y$.
4. For the rest of the queries $(x_1 \dots x_n)$, do the following:
   (a) Set $y \leftarrow C_n^j(x_1 \dots x_j)$.
   (b) For $i = j^*+2 \dots n$: update $y \leftarrow y \cdot u_i^{x_i}$.
   (c) If $x_{j^*+1} = 0$, output $g^y$, else output[14] $B^y$.
5. Output whatever $\mathcal{A}$ outputs.
```

By the construction of $\mathcal{B}$, if (g, A, B, C) is a DDH tuple, then the distribution of oracle responses seen by $\mathcal{A}$ are exactly the same as the

[14] Recall that $B = g^b$, so $B^y = g^{y \cdot b} = g^{y \cdot b^x_{j^*+1}}$. Therefore, the DDH relation is properly set for all pseudorandom nodes.

responses seen in the hybrid $H_n^{j^*,k^*}$. Otherwise, they are the same as hybrid $H_n^{j^*,k^*+1}$. We assumed that $\mathcal{A}$ can distinguish between $H_n^{j^*,k^*}$ and $H_n^{j^*,k^*+1}$, therefore $\mathcal{B}$ can distinguish between a DDH tuple and a uniform tuple. This contradicts our assumption that DDH is hard.

□

Exercises

Exercise 3.1. *Prove or disprove: If f is a one-way function, then the following function $B : \{0,1\}^* \to \{0,1\}$ is a hardconcentrate predicate for f. The function $B(x)$ outputs the inner product modulo 2 of the first $\lfloor |x|/2 \rfloor$ bits of x and the last $\lfloor |x|/2 \rfloor$ bits of x.*

Exercise 3.2. *Let $\phi(n)$ denote the first n digits of $\pi = 3.141592653589\ldots$ after the decimal in binary (π in its binary notation looks like $11.00100100001111110110101010001000100001\ldots$).*

Prove the following: if one-way functions exist, then there exists a one-way function f such that the function $B : \{0,1\}^* \to \{0,1\}$ is not a hard concentrate bit of f. The function $B(x)$ outputs $\langle x, \phi(|x|) \rangle$, where

$$\langle a, b \rangle := \sum_{i=1}^{n} a_i b_i \mod 2$$

for the bit-representation of $a = a_1 a_2 \cdots a_n$ and $b = b_1 b_2 \cdots b_n$.

Exercise 3.3. *If $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is PRF, then in which of the following cases is $g : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ also a PRF?*

1. $g(K, x) = f(K, f(K, x))$
2. $g(K, x) = f(x, f(K, x))$
3. $g(K, x) = f(K, f(x, K))$

Exercise 3.4 (Puncturable PRFs.). *Puncturable PRFs are PRFs for which a key can be given out such that, it allows evaluation of the PRF on all inputs, except for one designated input.*

A puncturable pseudo-random function F is given by a triple of efficient algorithms (Key_F,$\mathsf{Puncture}_F$, and Eval_F), satisfying the following conditions:

- ***Functionality preserved under puncturing**: For every $x^*, x \in \{0,1\}^n$ such that $x^* \neq x$, we have that:*

  $\Pr[\mathsf{Eval}_F(K, x) = \mathsf{Eval}_F(K_{x^*}, x) : K \leftarrow \mathsf{Key}_F(1^n), K_{x^*} = \mathsf{Puncture}_F(K, x^*)] = 1$

- ***Pseudorandom at the punctured point**: For every $x^* \in \{0,1\}^n$ we have that for every polysize adversary $\mathcal{A}$ we have that:*

  $|\Pr[\mathcal{A}(K_{x^*}, \mathsf{Eval}_F(K, x^*)) = 1] - \Pr[\mathcal{A}(K_{x^*}, \mathsf{Eval}_F(K, U_n)) = 1]| = \mathsf{negl}(n)(n)$

  where $K \leftarrow \mathsf{Key}_F(1^n)$ and $K_S = \mathsf{Puncture}_F(K, x^*)$. U_n denotes the uniform distribution over n bits.

Prove that: If one-way functions exist, then there exists a puncturable PRF family that maps n bits to n bits.

***Hint:** The GGM tree-based construction of PRFs from a length doubling pseudorandom generator (discussed in class) can be adapted to construct a puncturable PRF. Also note that K and K_{x^*} need not be the same length.*

4
Private-Key Cryptography

4.1 Private-Key Encryption

The first primitive that we will study in private-key cryptography is that of private-key encryption. When talking about private-key encryption, we will be working in a setting where two players, Alice and Bob, are attempting to communicate with each other.

Alice and Bob want to communicate with each other. For simplicity, let's assume that only Alice wants to send a message to Bob. The crucial property that they want is that no eavesdropper attempting to listen to the conversation should be able to decipher the contents of the message being sent.

To achieve this, the two employ the following communication protocol:

1. A priori, Alice and Bob generate a key k and distribute it in such a way that only the two of them know what k is.

2. Using k, Alice can encrypt her message m, to turn it into a ciphertext c, which she sends over to Bob.

3. Upon receiving c, Bob can decrypt its contents and recover m by using k.

This meta-scheme implies a couple of requirements. First of all, we want Bob to indeed be able to recover m when decrypting c with k. It is no use having a communication scheme where the message received is not the one sent. We will call this requirement *correctness*. The second requirement, which we have already mentioned, is *confidentiality*. To reiterate, *confidentiality* means that no eavesdropper that manages to get a hold of c should be able to learn anything about c that they do not already know (assuming they have no knowledge of the key k). In addition to these two fundamental requirements, we might also impose that our private-key encryption scheme guarantees *integrity* and *authenticity*. By *integrity*, we mean that Bob should be able to detect that the message c has been tampered with prior to him receiving it. By *authenticity*, we mean that Bob should be able to verify that the message he received was indeed sent by Alice, and not some adversary interfering with the conversation.

Now that we have some intuitive understanding of what we are trying to achieve, let us attempt to ground it in mathematics.

Definition 4.1 (Private-Key Encryption Scheme). *A private-key encryption scheme* Π *is a tuple* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, *where* $\mathsf{Gen}, \mathsf{Enc}$, *and* Dec *are algorithms such that:*

1. $\mathsf{Gen}(1^n) \to k$
2. $\mathsf{Enc}(k, m) \to c$
3. $\mathsf{Dec}(k, c) \to m'$

where n *is a security parameter and* $k, c, m, m' \in \{0,1\}^*$

Now, we will formalize the requirements of our cryptosystem. Our first requirement is correctness, which is defined below:

Definition 4.2 ((Perfect) Correctness). *We say that a private-key encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is* (perfectly) correct *if* $\forall n, k \in \mathsf{Gen}(1^n), m \in \{0,1\}^*$,

$$\Pr[\mathsf{Dec}(k, \mathsf{Enc}(k, m)) = m] = 1$$

That is, if $c = \mathsf{Enc}(k, m)$, then Bob is guaranteed to recover m by running $\mathsf{Dec}(k, c)$. Note that for a fixed-length encryption scheme, we require that $m \in \{0,1\}^{l(n)}$.

Next, we will formalize what we mean by *confidentiality*. We will often use the terms *confidentiality* and *security* interchangeably in the context of private-key encryption schemes. Our first definition of confidentiality is called IND Security, stated below:

Definition 4.3 (IND Security). $\forall m_0, \forall m_1$ *s.t.* $|m_0| = |m_1| = l(n)$ *and* $\forall$ *nu-PPT* $\mathcal{A}$ *we have*

$$|\Pr[\mathcal{A}(1^n, \mathsf{Enc}(k, m_0)) = 1 \mid k \leftarrow \mathsf{Gen}(1^n)] - |\Pr[\mathcal{A}(1^n, \mathsf{Enc}(k, m_1)) = 1 \mid k \leftarrow \mathsf{Gen}(1^n)]| = neg(n)$$

Note that this is not a particularly good definition of security, in the sense that the attacker is very limited in what they are allowed to do. Specifically, all that $\mathcal{A}$ can do is take a look at the encryption of m_0 and m_1 and must decide which one is the plaintext. We need a more usable and realistic definition of security. For this reason, we will allow the attacker to have oracle access to the encryption function, $\mathsf{Enc}(k, \cdot)$. In other words, $\mathcal{A}$ will be able to craft their own ciphertexts, which it can then use to break the security of the encryption scheme.

We shall dub this new definition of security *Chosen Plaintext Attack Security*, or *CPA Security* for short.

In defining *CPA Security*, we will also introduce a new method for defining private-key encryption schemes: the game-style definition. The rationale behind this change in style is that probabilistic definitions, while precise and rigorous, are rather cumbersome to work with, especially in the context of secure communication. Therefore, we will adopt this new paradigm, which will make it easier to work with and reason about private-key encryption schemes.

Priv-IND-CPA$_{\Pi}^{\mathcal{A}}(n)$

```
$b \xleftarrow{\$} \{0,1\}$
$\mathsf{k} \xleftarrow{\$} \mathsf{Gen}(1^n)$
$(\mathsf{state}, m_0, m_1) \xleftarrow{\$} \mathcal{A}^{\mathsf{Enc}(\mathsf{k},\cdot)}(1^n)$
$c \xleftarrow{\$} \mathsf{Enc}(\mathsf{k}, m_b)$
$b' \xleftarrow{\$} \mathcal{A}^{\mathsf{Enc}(\mathsf{k},\cdot)}(\mathsf{state}, c)$
**return** $b = b' \wedge |m_0| = |m_1| = l(n)$
```

Definition 4.4 (CPA Security). *A private-key encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is CPA-secure if* $\forall$ *nu-PPT* $\mathcal{A}$

$$\mathsf{Adv}_{\Pi,\mathcal{A}}^{\text{ind-cpa}}(n) = \left| \Pr[\text{Priv-IND-CPA}_{\Pi}^{\mathcal{A}}(n) = 1] - \frac{1}{2} \right|$$

is a negligible function.

Observe that in this new game-style definition, we have a concrete notion of the order in which each action is taken. One important detail to note (and that is more evident in a game-style definition) is that in our *CPA Security* definition, the key k is sampled *before* m_0 and m_1 are fixed. This is in contrast to *IND Security*, in which the messages m_0 and m_1 are chosen before the key k is sampled[1].

To further illustrate this point, consider the following scheme, which is secure in IND but insecure in CPA:

- $\mathsf{Gen}(1^n)$:
  1. $k \leftarrow \mathsf{Gen}(1^n)$
  2. $x \xleftarrow{\$} \{0,1\}^n$
  3. $k' = (k, x)$
- $\mathsf{Enc}'(k', m)L$:
  1. if $m = x$, then output x
  2. else, output $\mathsf{Enc}(k, m)||x$

The final security notion we will define is CCA (chosen-ciphertext attack) security. Here, the attacker is allowed oracle access to both the encryption and the decryption functions. Let L be the working list of queries that $\mathcal{A}$ has made to $Dec(k, \cdot)$. Then Priv-IND-CCA$_{\Pi}^{\mathcal{A}}(n)$ is defined as:

Priv-IND-CCA$_{\Pi}^{\mathcal{A}}(n)$

```
$b \xleftarrow{\$} \{0,1\}$
$\mathsf{k} \xleftarrow{\$} \mathsf{Gen}(1^n)$
$(\mathsf{state}, m_0, m_1) \xleftarrow{\$} \mathcal{A}^{\mathsf{Enc}(\mathsf{k},\cdot),\mathsf{Dec}(k,\cdot)}(1^n)$
$c \xleftarrow{\$} \mathsf{Enc}(\mathsf{k}, m_b)$
$b' \xleftarrow{\$} \mathcal{A}^{\mathsf{Enc}(\mathsf{k},\cdot),\mathsf{Dec}(k,\cdot)}(\mathsf{state}, c)$
**return** $b = b' \wedge |m_0| = |m_1| \wedge c \notin L$
```

Definition 4.5 (CCA Security). *A private-key encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is CCA-secure if* $\forall$ *nu-PPT* $\mathcal{A}$

$$\mathsf{Adv}_{\Pi,\mathcal{A}}^{\text{ind-cca}}(n) = \left| \Pr[\text{Pri-IND-CCA}_{\Pi}^{\mathcal{A}}(n) = 1] - \frac{1}{2} \right|$$

is a negligible function.

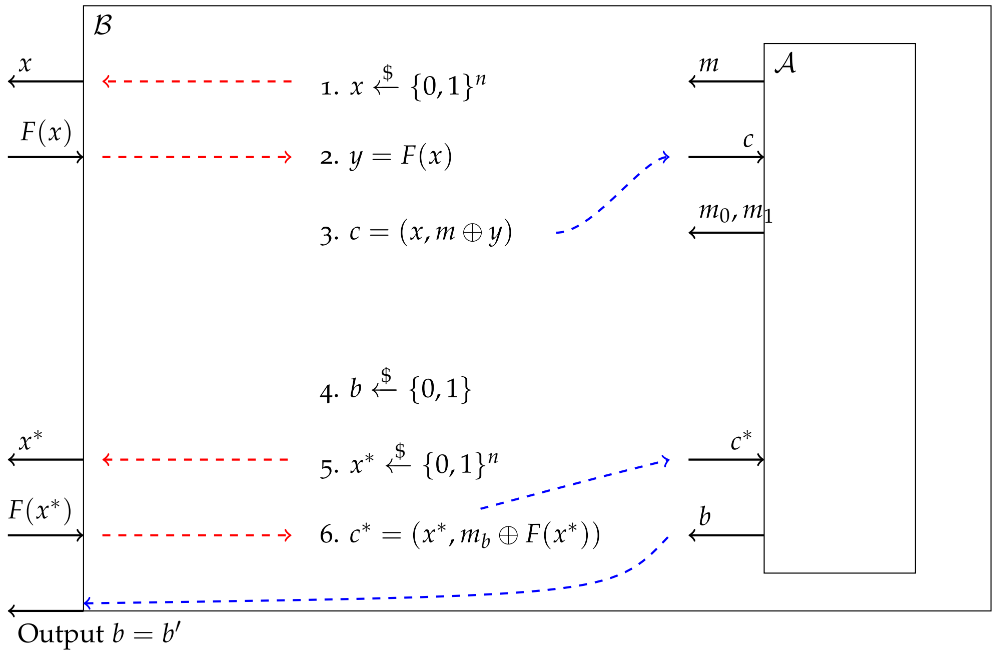
[1] This is an important detail because if m_0 and m_1 are chosen before k is sampled, then giving oracle access to $\mathcal{A}$ is not much help.

Π is a fixed-length encryption scheme for length $l(n)$ if $l(n)$ is polynomial in n and $|m_0| = |m_1| = l(n)$.

Theorem 4.1. *If F is a PRF then the scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ given below is a secure encryption scheme for length n.*

- $\mathsf{Gen}(1^n)$:
  1. *output* $k \xleftarrow{\$} \{0,1\}^n$
- $\mathsf{Enc}(k, m)$:
  1. $r \xleftarrow{\$} \{0,1\}^n$
  2. *output* $(r, F_k(r) \oplus m)$
- $\mathsf{Dec}(k, c = (c_1, c_2))$:
  1. *output* $c_2 \oplus F_k(c_1)$

Proof. Assume there exists a nu-PPT $\mathcal{A}$ that is able to break CPA security of Π. Then we can construct a nu-PPT adversary $\mathcal{B}$ that breaks the PRF F. The strategy is outlined in the figure below:


$\mathcal{B}$
x
$F(x)$
1. $x \xleftarrow{\$} \{0,1\}^n$
2. $y = F(x)$
3. $c = (x, m \oplus y)$
4. $b \xleftarrow{\$} \{0,1\}$
5. $x^* \xleftarrow{\$} \{0,1\}^n$
6. $c^* = (x^*, m_b \oplus F(x^*))$
m
$\mathcal{A}$
c
m_0, m_1
c^*
b
x^*
$F(x^*)$
Output $b = b'$

After running this procedure, we guess "Pseudorandom" if $b = b'$. Else, we guess random.

Now we argue that

$$|\Pr[B^{F_n(\cdot)}(1^n) = 1] - \Pr[\mathcal{B}^{F_n(\cdot)}(1^n) = 1]|$$

is non-negligible.

$$\begin{aligned} |\Pr[B^{F_n(\cdot)}(1^n)=1] - \Pr[\mathcal{B}^{F_n(\cdot)}(1^n)=1]| &\geq \frac{1}{2} + \epsilon(n) - (\frac{1}{2} + \frac{q(n)}{2^n}) \\ &= \epsilon(n) - \frac{q(n)}{2^n} \end{aligned}$$

□

Theorem 4.2. *No deterministic encryption scheme Π can be CPA Secure.*

Proof. The proof of this claim is simple. If we have a deterministic encryption scheme, then when we get c^*, we can again try to encrypt a message and check if $c = c^*$ □

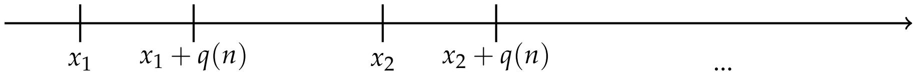
4.1.1 Counter Mode Encryption

One construction of a CCA-secure cipher is by the use of the counter mode.

- $\mathsf{Enc}(k, (m_1, ..., m_\ell))$:

```
$r \xleftarrow{\$} \{0,1\}^n$
Output $c = (r, m_1 \oplus F_k(r+1), m_2 \oplus F_k(r+2), ..., m_\ell \oplus F_k(r+\ell))$
```

Consider the following picture:



Then the probability of breaking this cipher is

$$\frac{2q(n)-1}{2^n} \cdot q(n)$$

In practice, we use block ciphers, which are stronger primitives.

4.2 Message Authentication Codes

Now we address the question of how we can guarantee the *integrity* of a message. To achieve this, we will construct a new primitive, called a *message authentication code*, or MAC for short. MACs generate a verifiable tag t for a message m that cannot be forged.

When sending a message, Alice sends the pair (m, t). Once Bob receives the message, he runs $\mathsf{Verify}(k, m, t)$. He accepts the message if $\mathsf{Verify}(k, m, t) = 1$, otherwise he rejects the message. The formal definition is stated below:

Definition 4.6 (Private-Key Encryption Scheme)**.** *A MAC scheme Π is a tuple of algorithms $\Pi = (\mathsf{Gen}, \mathsf{MAC}, \mathsf{Verify})$, with the following syntax:*

1. $k \leftarrow \mathsf{Gen}(1^n)$
2. $t \leftarrow \mathsf{MAC}(k, m)$
3. $0/1 \leftarrow \mathsf{Verify}(k, m, t)$

where n is a security parameter and $k, m \in \{0,1\}^{l(n)}$

We impose the following *correctness* requirement on our MACs:

Definition 4.7 (MAC Correctness)**.**

$$\forall n, k \in \mathsf{Gen}(1^n), m \in \{0,1\}^*, \Pr[\mathsf{Verify}(k, m, \mathsf{MAC}(k, m)) = 1] = 1$$

We also want the message authentication codes to be *unforgeable*. That is, given a message m, a nu-PPT attacker $\mathcal{A}$ should only be able to forge a tag t for m with negligible probability.

Definition 4.8 (EUF-CMA Security)**.** *A MAC scheme $\Pi = (\mathsf{Gen}, \mathsf{MAC}, \mathsf{Verify})$ is EUF-CMA-secure if $\forall$ nu-PPT $\mathcal{A}$,*

$$\left| \Pr\left[\mathsf{MAC\text{-}forge}_{\mathcal{A},\Pi}(n) = 1\right] \right| = \mathsf{negl}(n)$$

Definition 4.9 ($\mathsf{MAC\text{-}forge}_{\mathcal{A},\Pi}(n)$)**.**

1. ***Setup:*** *The challenger samples k uniformly from the key space. $\mathcal{A}$ is given 1^n.*
2. ***Query:*** *The adversary submits a message $m^{(i)}$; then the challenger computes a tag $t^{(i)} \leftarrow \mathsf{MAC}(k, m^{(i)})$ and sends it to the adversary. The adversary may submit any polynomial number of message queries.*

   Let $\mathcal{Q} = \{(m^{(1)}, t^{(1)}), \ldots, (m^{(q)}, t^{(q)})\}$ be the set of messages $m^{(i)}$ submitted in the query phase along with the tags $t^{(i)}$ computed by MAC.
3. ***Forgery:*** *The adversary outputs a message-tag pair (m^*, t^*). The output of the game is 1 if $(m^*, t^*) \notin \mathcal{Q}$ and $\mathsf{Verify}(k, m^*, t^*) = 1$. The output is 0 otherwise.*

4.3 *Fixed-length MACs*

Previously, we defined what a MAC is, and specified correctness and security definitions for MACs. In this section, we'll define a fixed-length MAC for length $\ell(n)$.

Theorem 4.3. *If $F : \{0,1\}^n \to \{0,1\}^n$ is a secure PRF, then the MAC scheme $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ constructed below has EUF-CMA security.*

- $\mathsf{Gen}(1^n)$:

  Output $k \xleftarrow{\$} \{0,1\}^n$

- $\mathsf{MAC}(k, m)$:

  Output $t = F_k(m)$

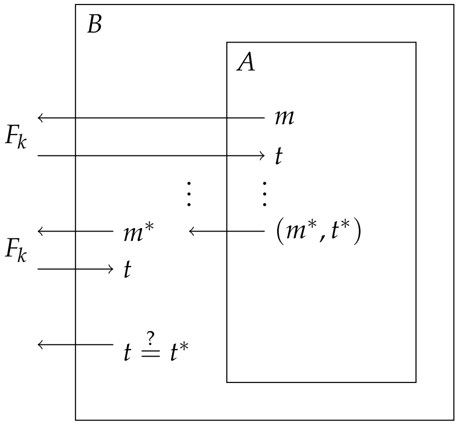
- $\mathsf{Verify}(k, m, t)$

  If $t = F_k(m)$, then return 1.
  Otherwise return 0.

That is, we just compute the PRF on our message as the MAC.

Proof. To prove security, suppose for contradiction that there exists an adversary A that breaks the security for Π. We'd like to construct an adversary B that breaks the security of the PRF.

Here, the adversary A expects queries for tags, given messages as input. B can simply forward these requests on to F, and return the response back to A. Further, A outputs a pair (m^*, t^*), which B can send m^* to F, and output whether $t = t^*$.



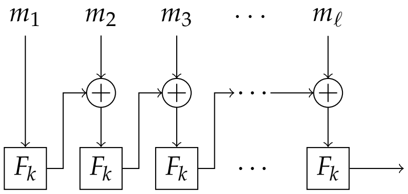
Analyzing the probability for B, we have

$$\left|\Pr(B^{F_k(\cdot)}(1^n) = 1) - \Pr(B^{R_n(\cdot)}(1^n) = 1)\right| = \left|\varepsilon_A(n) - \frac{1}{2^n}\right| = \mathsf{nonnegl}(n).$$

Here, the first term is because the correctness follows immediately from the correctness of A, and the second term is due to the fact that the output of R_n is random. □

4.4 *Variable-length MACs*

Now, let us look at messages with lengths that are a multiple of n. In particular, we have a few blocks $m_1, \ldots, m_\ell$, each of size n. There are a few ways to do this, but we'll look at a method similar to the counter mode we looked at last time.



This construction avoids having to store a tag equal in length to the message, but this is not secure, due to length extension attacks. In particular, suppose we query for the tag t associated with 0^n. We can then query another tag t' for $0^n \oplus t$. Observe here that t' is also the tag for 0^{2n}.

A solution is to use different keys for each PRF, but this isn't too efficient, since we're still calling the PRF once per block of length n. We'll instead improve this to use only one block cipher call—we do some preprocessing and only call F_k once on the output of the preprocessing.

In particular, we'll claim that applying a universal hash function to the input and then applying the block cipher is a secure MAC.

Definition 4.10 (Universal Hash Function). *A function $h : \mathcal{F} \times \mathcal{F}^* \to \mathcal{F}$ (where $\mathcal{F}$ is a field of size 2^m) is a universal hash function if for all $m, m' \in \mathcal{F}^{\leq \ell}$ (i.e. m and m' have length at most ℓ),*

$$\Pr_s(h(s,m) = h(s,m')) \leq \frac{\ell}{|F|}.$$

That is, the probability of collision is small.

Crucially here, we fix m and m', and we sample s. (If we fix an s, we can almost surely find an m and m' that collide.)

Today, we'll look at the following function:

$$h(s, m_0, \ldots, m_{\ell-1}) = m_0 + m_1 s + m_2 s^2 + \cdots + m_{\ell-1} s^{\ell-1} + s^\ell.$$

Claim 4.1. *The function defined by*

$$h(s, m_0, \ldots, m_{\ell-1}) = m_0 + m_1 s + m_2 s^2 + \cdots + m_{\ell-1} s^{\ell-1} + s^\ell$$

is a universal hash function.

Proof. We'd like to argue that for a fixed m and m', and a random s, the probability that there is a collision is at most $\frac{\ell}{|\mathcal{F}|}$.

We'll look at

$$h(x, m_0, \dots, m_t) - h(x, m'_0, \dots, m'_t) = (m_0 - m'_0) + \cdots + (m_{t-1} - m'_{t-1})x^{\ell-1}.$$

If there is a collision, this difference is 0. The probability that this polynomial of degree at most ℓ has a zero at x is at most $\frac{\ell}{|\mathcal{F}|}$, since it has at most ℓ zeroes. This means that h is indeed a universal hash function. □

Claim 4.2. *The MAC given by $F_k(h(s, m_1, \dots, m_\ell))$, for the universal hash function h given prior, is secure. (This is a slight variation on the Carter–Wegman MAC.)*

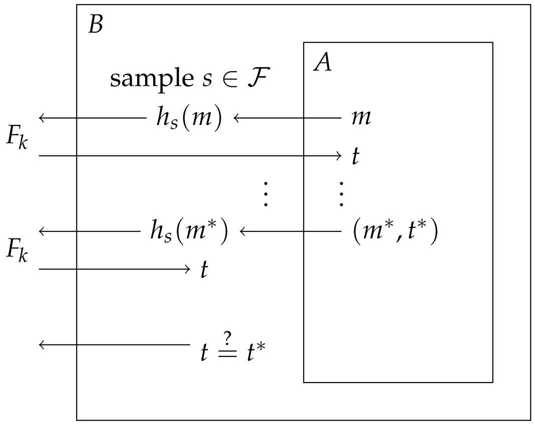
Proof. Suppose for contradiction that there exists a nu-PPT A that breaks the security of this scheme.

Here, for appropriately generated k and s, A makes queries $m \mapsto F_k(h_s(m))$, and outputs (m^*, t^*).

We'd like to create an adversary B that either breaks the security of the PRF, or breaks the security of the universal hash function.

B will start by sampling $s \in \mathcal{F}$. When given the query for m_1, it computes $h_s(m_1)$ and queries for $F_k(h_s(m_1))$, which it sends back to A. If F_k was actually pseudorandom, then A is given a pseudorandom input, and if F_k was random R_n, then A is given a random input.

A must still be able to generate pairs (m^*, t^*) even when given a random input, due to the security of the PRF.

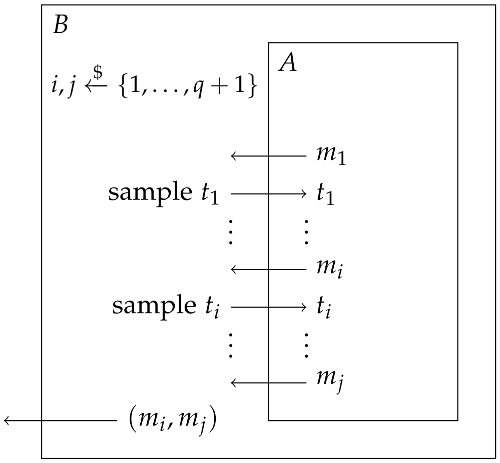


Let E be the event that there exists an $m, m' \in L \cup \{m^*\}$, such that $h_s(m) = h(m')$. If E does not happen, then the hash function never collides. This means that the attacker only sees random values depending on distinct inputs, so this reduces to the case from earlier (when the MAC is just F_k).

As such, we'd like to show that collisions in $h_s(\cdot)$ occur with negligible probability.

To show this, suppose for contradiction that collisions actually do occur with non-negligible probability. We then want to construct an adversary B utilizing A that just outputs m and m' such that when s is sampled, $h_s(m) = h_s(m')$ with high probability.

B will pick a random $i, j \in \{1, \ldots, q+1\}$ (here suppose $i < j$), where q is the number of MAC queries. We then run A until the jth query. Taking the ith and jth query, we then output m_i and m_j as our pair of messages. We still need to entertain the queries made by A, so we can just return random values for tags (giving the same value if it requests it for the same message).



By assumption, we know that E occurs with non-negligible probability. That is, among the queries made by A, there is a non-negligible probability that $h_s(m_i) = h_s(m_j)$. Since here the implementation of B just picks out a pair of random queries from those made by A, the pair (m_i, m_j) output by B also has a collision with non-negligible probability. (In particular, with probability $\Pr(E)/q^2$.

This breaks the definition of a universal hash function, which is a contradiction. □

So far, we know how to generate tags of fixed length, and of lengths that are a multiple of n. If we have a message that is not a multiple of n, we could potentially just pad the input with 0's, but this causes an issue, as m and $m\|0$ have the same tag.

Instead, one solution is to put the size of the message in the first block, and we can still put the padding at the end. This way, if the messages differ by length, the first block will be different, and if the messages do not differ by length, then we're essentially just ignoring the padding. This gives us a MAC for arbitrary-length messages.

4.5 *Authenticated Encryption Schemes*

We've talked about confidentiality and integrity separately, but generally we want both properties—when Alice sends a message to Bob, we'd like for any eavesdropper to be unable to recover the message, *and* we'd like Bob to be able to verify that the message actually came from Alice.

A scheme that achieves both of these conditions is called an *authenticated encryption scheme*.

Definition 4.11 (Authenticated Encryption Scheme). *A scheme Π is an* authenticated encryption scheme *if it is CPA-secure, and it has ciphertext integrity (CI).*

Definition 4.12 (Ciphertext Integrity (CI)). *Consider the following game for the scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.*

```
function $\mathrm{CI}_\Pi^A(n)$
    $k \leftarrow \mathsf{Gen}(1^n)$
    $c^* \leftarrow A^{\mathsf{Enc}(k,\cdot)}(1^n)$
    $L \leftarrow$ the list of queries made by $A$
    return $(\mathsf{Dec}(k, c^*) \neq \perp) \wedge (c^* \notin L)$
end function
```

A scheme has ciphertext integrity if for all nu-PPT A, $\Pr(\mathrm{CI}_\Pi^A)$ is negligible.

Observe that an authenticated encryption scheme is also CCA-secure, since the CI property says that the adversary can never generate a valid ciphertext. This means that whenever an adversary requests the decryption of a ciphertext, we can always return $\perp$ (unless they previously requested a ciphertext for a message, and wants to decode that ciphertext). This means that the decryption oracle is essentially useless, and this reduces to the CPA case.

Next, we'll construct an authenticated encryption scheme, called "Encrypt-then-MAC", utilizing a CPA-secure encryption scheme and an EUF-CMA MAC scheme.

Claim 4.3. *Let $\Pi_e = (\mathsf{Gen}_e, \mathsf{Enc}_e, \mathsf{Dec}_e)$ be a CPA-secure encryption scheme, and let $\Pi_m = (\mathsf{Gen}_m, \mathsf{Mac}_m, \mathsf{Verify}_m)$ be an EUF-CMA-secure MAC scheme.*

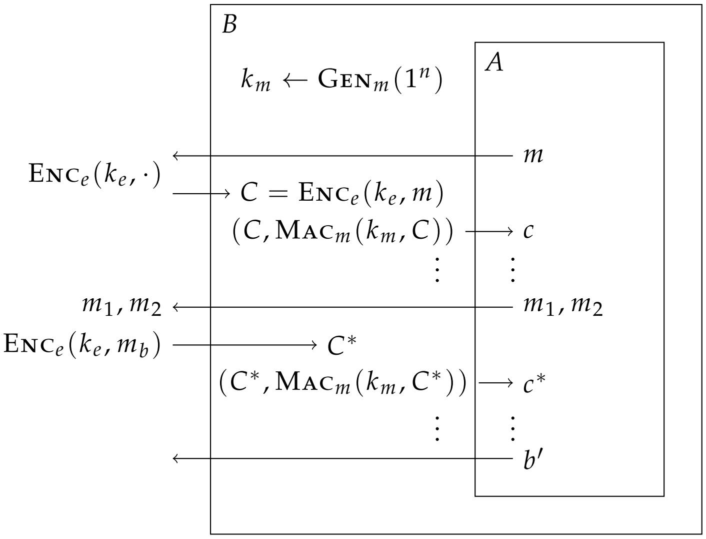
The following scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is an authenticated encryption scheme.

```
function GEN($1^n$)
    $k_e \leftarrow \mathsf{Gen}_e(1^n)$
    $k_m \leftarrow \mathsf{Gen}_m(1^n)$
    return $(k_e, k_m)$
end function

function ENC($(k_e, k_m), m$)
    $c \leftarrow \mathsf{Enc}_e(k_e, m)$
    $t \leftarrow \mathsf{Mac}_m(k_m, c)$
    return $(c, t)$
end function

function DEC($(k_e, k_m), (c, t), m$)
    if $\mathsf{Verify}_m(k_m, c, t)$ then
        return $\mathsf{Dec}_e(k_e, c)$
    else
        return $\perp$
    end if
end function
```

Proof. Suppose for contradiction that we have an adversary A that breaks the CPA security of Π. The CPA game allows for queries of the ciphertext for messages m, produces a pair m_0, m_1, and then gets $c^* = \mathsf{Enc}(k, m_B)$, and A eventually outputs b' to identify which message was encrypted.

We'd like to construct another adversary B, which breaks the CPA-security of Π_e. The only difference here is the MACs, so B can sample a $k_m \leftarrow \mathsf{Gen}_m(1^n)$, and perform all of the MACs itself.

In particular, when A asks for the ciphertext of M, we pass it to the oracle for Π_e, and attach $t \leftarrow \mathsf{Mac}_m(k_m, c)$. If A is able to distinguish between ciphertexts of M_0 and M_1, then we can use the same bit to distinguish between ciphertexts for Π_e.

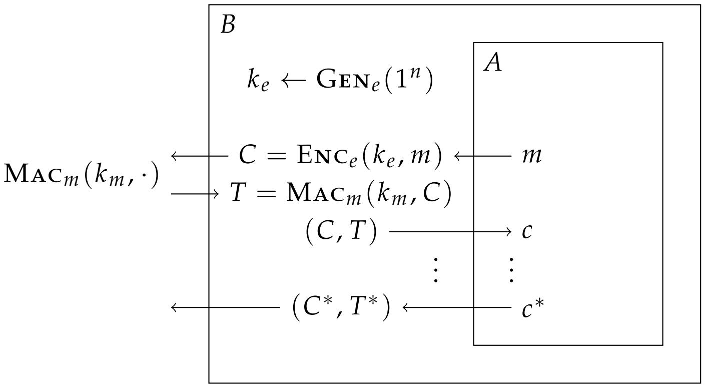

B
A
$k_m \leftarrow \textsc{Gen}_m(1^n)$
$\textsc{Enc}_e(k_e, \cdot)$
m
$C = \textsc{Enc}_e(k_e, m)$
$(C, \textsc{Mac}_m(k_m, C))$
c
m_1, m_2
m_1, m_2
$\textsc{Enc}_e(k_e, m_b)$
C^*
$(C^*, \textsc{Mac}_m(k_m, C^*))$
c^*
b'

To prove ciphertext integrity, suppose we have an adversary A that breaks the ciphertext integrity of Π. Here, A asks for ciphertext queries, and eventually returns a new ciphertext that is valid.

We'd like to construct an adversary B that is able to generate a new message and a tag, given oracle access to the MAC scheme. The construction will follow similarly to the prior proof on CPA security.

Here, our adversary B can sample $k_e \leftarrow \mathsf{Gen}_e(1^n)$. When A asks for the encryption of M, B can send $m = \mathsf{Enc}_e(k_e, M)$ to the MAC oracle, and it returns $c = (m, t)$ to A.

When A returns $C^* = (c^*, t^*)$, B can also just return the same, since the tag t^* is being computed on c^*.


B
A
$k_e \leftarrow \textsc{Gen}_e(1^n)$
$\textsc{Mac}_m(k_m, \cdot)$
$C = \textsc{Enc}_e(k_e, m)$
m
$T = \textsc{Mac}_m(k_m, C)$
(C, T)
c
(C^*, T^*)
c^*

□

As an example, AES-GCM is the most popular authenticated encryption scheme that is used, and also has the ability to authenticate additional data. (AES-GCM basically just appends the associated data to the ciphertext, so that the encryption is only on the message, but the MAC is on both the ciphertext and the associated data.) This scheme uses a counter-mode encryption scheme, and the MAC that we saw, but makes this more efficient.

5
Digital Signatures

In this chapter, we will introduce the notion of a digital signature. At an intuitive level, a digital signature scheme helps providing authenticity of messages and ensuring non-repudiation. We will first define this primitive and then construct what is called as one-time secure digital signature scheme. An one-time digital signature satisfies a weaker security property when compared to digital signatures. We then introduce the concept of collision-resistant hash functions and then use this along with a one-time secure digital signature to give a construction of digital signature scheme.

5.1 Definition

A digital signature scheme is a tuple of three algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ with the following syntax:

1. $\mathsf{Gen}(1^n) \rightarrow (vk, sk)$: On input the message length (in unary) 1^n, Gen outputs a secret signing key sk and a public verification key vk.

2. $\mathsf{Sign}(sk, m) \rightarrow \sigma$: On input a secret key sk and a message m of length n, the Sign algorithm outputs a signature σ.

3. $\mathsf{Verify}(vk, m, \sigma) \rightarrow \{0, 1\}$: On input the verification key vk, a message m and a signature σ, the Verify algorithm outputs either 0 or 1.

We require that the digital signature to satisfy the following correctness and security properties.

Correctness. For the correctness of the scheme, we have that $\forall m \in \{0, 1\}^n$,

$$\Pr\left[(vk, sk) \leftarrow \mathsf{Gen}(1^n), \sigma \leftarrow \mathsf{Sign}(sk, m) : \mathsf{Verify}(vk, m, \sigma) = 1\right] = 1.$$

Security. Consider the following game between an adversary and a challenger .

1. The challenger first samples $(vk, sk) \leftarrow \mathsf{Gen}(1^n)$. The challenger gives vk to the adversary.

2. **Signing Oracle.** The adversary is now given access to a signing oracle. When the adversary gives a query m to the oracle, it gets back $\sigma \leftarrow \mathsf{Sign}(sk, m)$.

3. **Forgery.** The adversary outputs a message, signature pair (m^*, σ^*) where m^* is different from the queries that adversary has made to the signing oracle.

4. The adversary wins the game if $\mathsf{Verify}(vk, m^*, \sigma^*) = 1$.

We say that the digital signature scheme is secure if the probability that the adversary wins the game is $\mathsf{negl}(n)(n)$.

5.2 *One-time Digital Signature*

An one-time digital signature has the same syntax and correctness requirement as that of a digital signature scheme except that in the security game the adversary is allowed to call the signing oracle only once (hence the name one-time). We will now give a construction of one-time signature scheme from the assumption that one-way functions exists.

Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way function.

- $\mathsf{Gen}(1^n)$: On input the message length (in unary) 1^n, Gen does the following:

  1. Chooses $x_{i,b} \leftarrow \{0,1\}^n$ for each $i \in [n]$ and $b \in \{0,1\}$.
  2. Output $vk = \begin{bmatrix} f(x_{1,0}) & \dots & f(x_{n,0}) \\ f(x_{1,1}) & \dots & f(x_{n,1}) \end{bmatrix}$ and $sk = \begin{bmatrix} x_{1,0} & \dots & x_{n,0} \\ x_{1,1} & \dots & x_{n,1} \end{bmatrix}$

- $\mathsf{Sign}(sk, m)$: On input a secret key sk and a message $m \in \{0,1\}^n$, the Sign algorithm outputs a signature $\sigma = x_{1,m_1} \| x_{2,m_2} \| \dots \| x_{n,m_n}$.

- $\mathsf{Verify}(vk, m, \sigma)$: On input the verification key vk, a message m and a signature σ, the Verify algorithm does the following:

  1. Parse $\sigma = x_{1,m_1} \| x_{2,m_2} \| \dots \| x_{n,m_n}$.
  2. Compute $vk'_{i,m_i} = f(x_{i,m_i})$ for each $i \in [n]$.
  3. Check if for each $i \in [n]$, $vk'_{i,m_i} = vk_{i,m_i}$. If all the checks pass, output 1. Else, output 0.

Before we prove any security property, we first observe that this scheme is completely broken if we allow the adversary to ask for two signatures. This is because the adversary can query for the signatures on 0^n and 1^n respectively and the adversary gets the entire secret key. The adversary can then use this secret key to sign on any message and break the security.

We will now argue the one-time security of this construction. Let $\mathcal{A}$ be an adversary who breaks the security of our one-time digital signature scheme with non-negligible probability $\mu(n)$. We will now construct an adversary $\mathcal{B}$ that breaks the one-wayness of f. $\mathcal{B}$ receives a one-way function challenge y and does the following:

1. $\mathcal{B}$ chooses i^* uniformly at random from $[n]$ and b^* uniformly at random from $\{0,1\}$.
2. It sets $vk_{i^*,b^*} = y$
3. For all $i \in [n]$ and $b \in \{0,1\}$ such that $(i,b) \neq (i^*,b^*)$, $\mathcal{B}$ samples $x_{i,b} \leftarrow \{0,1\}^n$. It computes $vk_{i,b} = f(x_{i,b})$.
4. It sets $vk = \begin{bmatrix} vk_{1,0} & \dots & vk_{n,0} \\ vk_{1,1} & \dots & vk_{n,1} \end{bmatrix}$ and sends vk to $\mathcal{A}$.
5. $\mathcal{A}$ now asks for a signing query on a message m. If $m_{i^*} = b^*$ then $\mathcal{B}$ aborts and outputs a special symbol $\texttt{abort}_1$. Otherwise, it uses it knowledge of $x_{i,b}$ for $(i,b) \neq (i^*,b^*)$ to output a signature on m.
6. $\mathcal{A}$ outputs a valid forgery (m^*,σ^*). If $m^*_{i^*} = m_{i^*}$ then $\mathcal{B}$ aborts and outputs a special symbol $\texttt{abort}_2$. If it does not abort, then it parses σ^* as $1, m_1 \| x_{2,m_2} \| \dots \| x_{n,m_n}$ and outputs x_{i^*,b^*} as the inverse of y.

We first note that conditioned on $\mathcal{B}$ not outputting $\texttt{abort}_1$ or $\texttt{abort}_2$, the probability that $\mathcal{B}$ outputs a valid preimage of y is $\mu(n)$. Now, probability $\mathcal{B}$ does not output $\texttt{abort}_1$ or $\texttt{abort}_2$ is $1/2n$ (this is because $\texttt{abort}_1$ is not output with probability $1/2$ and conditioned on not outputting $\texttt{abort}_1$, $\texttt{abort}_2$ is not output with probability $1/n$). Thus, $\mathcal{B}$ outputs a valid preimage with probability $\mu(n)/2n$. This completes the proof of security.

We now try to extend this one-time signature scheme to digital signatures. For this purpose, we will rely on a primitive called as collision-resistant hash functions.

5.3 *Universal One-way Hash Function – UOWHF*

We now introduce a class of hash function called universal one-way hash function. This form of hash function is stronger than the universal hash function in 4.10 in the sense that the attacker is allowed to select the input by itself but must do so before being presented with the functions description.

5.3.1 Definition

We now give a formal definition of universal one-way hash function(UOWHF, also called "WOOF").

Definition 5.1 (Universal One-way Hash Function(UOWHF)). *Let ℓ: $\mathbb{N} \to \mathbb{N}$. A collection of functions $\{h_s : \{0,1\}^* \to \{0,1\}^{\ell(|s|)}\}_{s\in\{0,1\}^*}$ is called universal one-way hash functions given $\exists$ PPT machine I such that:*

1. *(Easy to compute): $\exists$ a deterministic machine M such that $M(s,x) = h_s(x)$.*
2. *(Hard to form designated collisions): Given a PPT attacker $\mathcal{A}$, the probability of $\mathcal{A}$ to win WOOF game is negligible.*

$$\Pr[WOOF_{\mathcal{A}}^{I,h}(n) = 1] = \mathsf{negl}(n)$$

Where $WOOF_{\mathcal{A}}^{I,h}$ is defined in the following:

1. $(state, x) \leftarrow \mathcal{A}(1^n)$.
2. $s \leftarrow I(1^n)$.
3. $y \leftarrow \mathcal{A}(state, s)$
4. Output $h_s(x) = h_s(y)$

5.3.2 Construction

We construct UOWHF collections in several steps, starting with a related but restricted notion and relaxing the restriction gradually until we reach the unrestricted notion of UOWHF collections.

Definition 5.2 ((d,r)-UOWHFs). *Let d, r: $\mathbb{N} \to \mathbb{N}$. A collection of functions $\{h_s : \{0,1\}^{(d(|s|))} \to \{0,1\}^{r(|s|)}\}_{s\in\{0,1\}^*}$ is called (d,r)-UOWHFs given $\exists$ PPT machine I such that:*

1. *$\exists$ a deterministic machine M such that given s and $x \in \{0,1\}^{(d(|s|))}$, $M(s,x) = h_s(x)$.*
2. *(Hard to form designated collisions): Given a PPT attacker $\mathcal{A}$, the probability of $\mathcal{A}$ to win WOOF game is negligible.*

$$\Pr[WOOF_{\mathcal{A}}^{I,h}(n) = 1] = \mathsf{negl}(n)$$

And d, r satisfying $d(n) > r(n)$. The other case is trivial to discuss.

Now we discuss the construction of 4 types of UOWHFs. We start from the simple cases and eventually lead to the full-fledged UOWHFs.

Step I: $(d, d-1)$**-UOWHFs**. We construct UOWHFs that given arbitrary length, truncate its input length by 1.

Construction 5.1. *Let $f\colon \{0,1\}^* \to \{0,1\}^*$ be a one-way permutation, and $a, b \in GF(2^n)$. We construct $h_s : \{0,1\}^n \to \{0,1\}^{n-1}$:*

$$h_{s=(a,b)}(x) = chop(a \cdot f(x) + b)$$

where $chop(\cdot)$ is the function that chops the first bit of a given bit string.

We then argue that h_s is a $(d, d-1)$-UOWHF.

Proof. We prove this by contradiction. If an adversary $\mathcal{A}$ can break the security of UOWHF, we can construct an adversary $\mathcal{B}$ that breaks the OWF f.

The algorithm of $\mathcal{B}$ is as follows upon receive an input y:

1. $\mathcal{A}$ first emits the preimage x_0 of its chosen.
2. Attacker $\mathcal{B}$ compute $s = (a, b)$ such that $h_s(y) = h_s(x_0)$.
3. Input $s = (a, b)$ to the oracle $\mathcal{A}$ and output what $\mathcal{A}$ outputs.

Since h_s works by chopping the first bit of $af(x) + b$, there are two elements in the preimage of $h_s(x)$, namely x and $x' = f^{-1}(y)$. Because it is easy to compute a and b that $af(x) + b = ay + b$, the hardness relies on finding x' given y, which is inverting the one-way permutation. This creates a contradiction of one-way permutation.

□

Step II: $(2n, n)$**-UOWHFs**. Now we construct length-restricted UOWHFs that shrink their input by a factor of 2.

Construction 5.2. *Let $\{h_s : \{0,1\}^* \to \{0,1\}^{*-1}\}_{s\in\{0,1\}^*}$. We construct $H_{s_1\cdots s_n} : \{0,1\}^{2n} \to \{0,1\}^n$:*

$$H_{s_1\cdots s_n} = h_{s_1}^{n+1,n}(h_{s_2}^{n+2,n+1}(\cdots h_{s_n}^{2n,2n-1}(x))\cdots)$$

We now prove that H is a $(2n, n)$-UOWHF given h is a $(d, d-1)$-UOWHF. That is, if we have an attacker $\mathcal{A}$ that can break $(2n, n)$-UOWHF, we can construct an attacker $\mathcal{B}$ that can break the $(d, d-1)$-UOWHF. Our intuition is that the attacker $\mathcal{B}$ can randomly inject the challenge s_i into a random location of S to $\mathcal{A}$ and $\mathcal{A}$ will have a non-negligible probability to create a collision.

Proof. Formally we define the procedure of $\mathcal{B}$:

1. Select $i \in [n]$.
2. $\forall i' \neq i$, sample $s_{i'}$.

3. Receive x_0 from $\mathcal{A}$.

4. Compute $x_i^0 = H_{partial}(x_0) = h_{s_{n-i+1}}^{2n-i+1,2n-i}(\ldots(h_{s_n}^{2n,2n-1}(x))\ldots)$, which is the input of i-th hash round select by $\mathcal{B}$

5. Output x_0 to the challenger.

6. Receive S_i from the challenger. And send $\{S_0, \ldots, S_n\}$ to $\mathcal{A}$.

7. $\mathcal{A}$ output x_1.

8. $\mathcal{B}$ compute $H_{partial}(x_1)$ and output.

The core idea is that if $\mathcal{A}$ can output a collision with different x, the output will be the same at some point in the hash chain of applying h. Therefore $\mathcal{B}$ can randomly insert the challenge into $\mathcal{A}$. And with $\mu(\mathcal{A})/n$ probability $\mathcal{B}$ will succeed. $\mu(\mathcal{A})$ is the probability that $\mathcal{A}$ succeed. □

Step III: UOWHFs that shrinks any input by a factor of two.

Construction 5.3 ((a $(2n^*, n^*)$-UOWHF for any length of n^*)). *Let $\{h_s : \{0,1\}^{2n} \to \{0,1\}^n\}_{s\in\{0,1\}^*}$. Then for every $x \in \{0,1\}^*$ we have*

$$H_s(x) = h_s(x_1)\cdots h_s(x_t 10^{n^*-|x_t|-1})$$

where $x = x_1 \cdots x_t$, $0 \le |x_t| \le n^*$ and $|x_i| = n^*$ for $i = 1, \ldots, t-1$.

We now give the proof sketch here. We want to prove that H is a $(2n^*, n^*)$-UOWHF given h is a $(2n, n)$-UOWHF. Because $H_s(x)$ is just parallelize $(2n, n)$-UOWHF multiple times.

We assume that $\mathcal{A}$ is able to create a collision x' for some input x of its given and s. Some block x_i' will be different than the block x_i. Therefore we have $h_s(x_i') = h_s(x_i)$ which create contradiction of h is a $(2n, n)$-UOWHF.

Step IV: Full-Fledged UOWHFs. The last step on our way consists of using UOWHFs as constructed in Step III above to obtain full-fledged UOWHFs.

Construction 5.4 ((a UOWHF)). *Let $\{h_s : \{0,1\}^* \to \{0,1\}^*\}_{s\in\{0,1\}^*}$, such that $|h_s(s)| = |x|/2$, for every $x \in \{0,1\}^{2i\cdot|s|}$ where $i \in \mathbb{N}$. Then for every $s_a, \cdots, s_n \in \{0,1\}^n$, every $t \in \mathbb{N}$ and $x \in \{0,1\}^{2^t\cdot n}$, we define*

$$H_{s_1,\ldots,s_n}(x) = (t, h_{s_t}(\cdots(h_{s_2}(h_{s_1}(x)))\cdots))$$

The proof is similar in Step II. We show how to use the arbitrary-length WOOF we constructed to boost this one-time, fixed-length digital signature scheme into a one-time, arbitrary-length digital signature scheme.

5.3.3 Removing Length-Restriction from One-Time Digital Signatures

Let $(\mathsf{Gen}_\ell, \mathsf{Sign}_\ell, \mathsf{Verify}_\ell)$ be a length-restricted one-time digital signature for messages of length $\ell(n)$. Let $h_s : \{0,1\}^* \to \{0,1\}^n$ be a WOOF. First we will review an *insecure* first attempt at a construction for $\ell(n) = n$:

- $\mathsf{Gen}^{\mathrm{BAD}}(1^n)$: Run $(pk_\ell, sk_\ell) \leftarrow \mathsf{Gen}_\ell(1^n)$, $s \leftarrow I(1^n)$. Output $((pk_\ell, s), sk_\ell)$.
- $\mathsf{Sign}^{\mathrm{BAD}}(sk = sk_\ell, m)$: Output $\mathsf{Sign}_\ell(sk_\ell, h_s(m))$.
- $\mathsf{Verify}^{\mathrm{BAD}}(pk = (pk_\ell, s), m, \sigma)$: Output $\mathsf{Verify}_\ell(pk_\ell, h_s(m), \sigma)$.

Notice that this construction does not work because the seed for the WOOF is revealed before the message is chosen, in which case WOOF security does not apply.

To avoid this, we can use the following construction for $\ell(n) = k(n) + n$ where $k(n)$ is the length of the seed produced by $I(1^n)$:

- $\mathsf{Gen}(1^n)$: Run $(pk_\ell, sk_\ell) \leftarrow \mathsf{Gen}_\ell(1^n)$. Output (pk_ℓ, sk_ℓ).
- $\mathsf{Sign}(sk = sk_\ell, m)$: Run $s \leftarrow I(1^n)$ and $\sigma_\ell \leftarrow \mathsf{Sign}_\ell(sk_\ell, s||h_s(m))$. Output (σ_ℓ, s).
- $\mathsf{Verify}(pk = pk_\ell, m, \sigma = (\sigma_\ell, s))$: Output $\mathsf{Verify}_\ell(pk_\ell, s||h_s(m), \sigma_\ell)$.

Note that now the seed is chosen after the message. We will give a proof sketch of the security for this construction. Assume for contradiction that we have a nu-PPT adversary $\mathcal{A}$ which succeeds with non-negligible probability at the digital signature security game for $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$. We will now construct a nu-PPT adversary $\mathcal{B}$ who succeeds with non-negligible probability at the digital signature security game for $(\mathsf{Gen}_\ell, \mathsf{Sign}_\ell, \mathsf{Verify}_\ell)$, a contradiction. $\mathcal{B}$ receives pk_ℓ from its challenger and passes this along to $\mathcal{A}$. When $\mathcal{A}$ queries the signing oracle with message m, $\mathcal{B}$ runs $s \leftarrow I(1^n)$, computes $m_\ell = s||h_s(m)$, queries its signing oracle with message m_ℓ to receive σ_ℓ and returns $\sigma = (s, \sigma_\ell)$ to $\mathcal{A}$. Finally, when $\mathcal{A}$ returns $m^*, \sigma^* = (s^*, \sigma^*_\ell)$, $\mathcal{B}$ outputs $m^*_\ell = s^*||h_{s^*}(m^*), \sigma^*_\ell$.

Since we have replicated the expected input distribution for $\mathcal{A}$, it will succeed with non-negligible probability. Notice that $\mathcal{B}$ will succeed when $\mathcal{A}$ does as long as $m^*_\ell \neq m_\ell$. In analyzing the success probability of $\mathcal{B}$ we have two cases to consider based on whether $s^* = s$. Notice that $\mathcal{A}$ must have non-negligible success either when $s^* = s$ or when $s^* \neq s$ or both. If $\Pr[s^* \neq s \wedge \mathcal{A}$ succeeds$]$ is non-negligible, then $\mathcal{B}$ also succeeds with non-negligible probability since $m^*_\ell = s^*||h_{s^*}(m^*) \neq s||h_s(m) = m_\ell$ in this case. Now assume that $\Pr[s^* = s \wedge \mathcal{A}$ succeeds$]$ is non-negligible. When $\mathcal{A}$ succeeds in this case, it must have found $m^* \neq m$, and so $h_{s^*}(m^*) = h_s(m)$ with only negligible probability because otherwise WOOF security is broken. Thus with non-negligible probability $m^*_\ell \neq m_\ell$ and $\mathcal{B}$ succeeds as well. Therefore either way $\mathcal{B}$ succeeds in the digital signature game against $(\mathsf{Gen}_\ell, \mathsf{Sign}_\ell, \mathsf{Verify}_\ell)$ with non-negligible probability.

5.4 *Multiple-Message Digital Signatures*

Now we will show how to covert this one-time, no-length restriction digital signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ into a ef-ema no-length restriction digital signature scheme by utilizing a pseudorandom function PRF. For each $\alpha \in \{\epsilon\} \cup \{0,1\}^{\leq n}$, let $pk_\alpha, sk_\alpha = \mathsf{Gen}(1^n; PRF_s(\alpha 10 \ldots 0))$ such that $|\alpha 10 \ldots 0| = n + 1$ (i.e. Gen is run with randomness determined by the PRF on an input specified by α). We will use these to make a tree of keys so that the keys used for each message will be distinct with high probability, so WOOF security will continue to apply each time the scheme is used. Note that whenever Sign is called, we require the WOOF to be run with a deterministic seed $s^W_\alpha = PRF_{s'}(\alpha 10 \ldots 0)$. This way paths through the tree will deterministically map to the corresponding signatures. The construction is as follows:

- $\mathsf{GEN}(1^n)$: Output $(pk_\epsilon, s \leftarrow \{0,1\}^n)$, namely the root public key and the seed for the PRF so the rest of the keys can be generated.
- $\mathsf{SIGN}(sk, m)$:
  1. Draw a random path through the key tree $r \leftarrow \{0,1\}^n$.
  2. Now use the secret key at each level to sign its children's public keys and continue to do this along the random path until a leaf is hit, i.e. iteratively sign the random path and its co-path. Namely, for each $i = 0, 1, \ldots, n-1$, let $\alpha_i = r_1 r_2 \cdots r_i$, $m_i = pk_{\alpha_i||0}||pk_{\alpha_i||1}$, and $\sigma_i = \mathsf{Sign}(sk_{\alpha_i}, m_i)$.
  3. Let $\sigma_n = \mathsf{Sign}(sk_r, m)$.
  4. Output $\Sigma = (r, m_0, \sigma_0, \ldots, m_{n-1}, \sigma_{n-1}, \sigma_n)$.
- $\mathsf{VERIFY}(pk, m, \Sigma = (r, \sigma_0, \ldots, \sigma_{n-1}, \sigma_n))$: Use r and the m_i to determine pk_{α_i} and for each $i \in [n]$ run $\mathsf{Verify}(pk_{\alpha_i}, m_i, \sigma_i)$, accepting if all of those do.

The idea is that because the root pk_ϵ is trusted and the corresponding secret keys of one level are used to validate the pk of the level below, trust is maintained down the path and the ultimate pk_r can be trusted to be used to check the signature on m itself.

We will now give a proof sketch for the security of this construction. Assume for contradiction that we have a nu-PPT adversary $\mathcal{A}$

which succeeds with non-negligible probability at the digital signature security game for $(\mathsf{GEN}, \mathsf{SIGN}, \mathsf{VERIFY})$. Let $\mathcal{A}$'s interaction with this security game be called Hybrid H_0. First we will consider the hybrid H_1 where the PRF is replaced by a truly random function. $\mathcal{A}$'s success probability in H_1 is still some non-negligible $\epsilon(n)$ due to PRF security. Now we will consider the hybrid H_2 where if the randomness returned by any two of the signing oracle queries is equal, i.e. $r_j = r_{j'}$ for some distinct $j, j' \in [q]$, then $\mathcal{A}$ aborts. Notice that the probability of this happening is only $q^2/2^n$, a negligible amount, so $\mathcal{A}$'s success probability $\epsilon(n) - q^2/2^n$ remains non-negligible. Thus going forward we can assume that the randomness r_j used by the oracle to sign each query m_j is distinct.

Now notice that either $\mathcal{A}$ outputs a message-signature pair (M^*, Σ^*) which uses $r^* = r_j$ for some $j \in [q]$ or $r^* \neq r_j$ for all $j \in [q]$. We will give some intuition for what happens in each of these cases. In the first case, since signatures in this scheme are deterministic, to succeed $\mathcal{A}$'s signatures must be the same as Σ_j's along the path $r^* = r_j$ until doing a forgery at the leaf, breaking the $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ scheme for M^* which is distinct from all of the m_j queries. In the second case, $\mathcal{A}$ goes along a path $r^* \neq r_j$, so at the first node which diverges from all r_j it must forge a signature that verifies with an honest public key from the level above, breaking the $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ scheme for the corresponding α_i^* which is distinct from all of the other α_i^j that were used to answer queries.

This can be formalized by constructing a nu-PPT adversary $\mathcal{B}$ for the one-time digital signature security game of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ who takes the pk it's been given and guesses which query $\mathcal{A}$ forges for, using its pk and one-time oracle to provide a signature for that query and otherwise answering honestly using self-generated keys. This degrades $\mathcal{B}$'s probability of success by $\epsilon(n)/q$ which is still non-negligible.

Next, after introducing collision resistant hash functions, we will see a different, though closely related, alternate construction for multiple-message digital signatures.

5.5 *Collision Resistant Hash Functions*

As the name suggests, collision resistant hash function family is a set of hash functions H such that for a function h chosen randomly from the family, it is computationally hard to find two different inputs x, x' such that $h(x) = h(x')$. We now give a formal definition.

5.5.1 Definition of a family of CRHF

A set of function ensembles

$$\{H_n = \{h_i : D_n \to R_n\}_{i \in I_n}\}_n$$

where $|D_n| < |R_n|$ is a family of collision resistant hash function ensemble if there exists efficient algorithms $(\mathsf{Sampler}, \mathsf{Eval})$ with the following syntax:

1. $\mathsf{Sampler}(1^n) \to i$: On input 1^n, $\mathsf{Sampler}$ outputs an index $i \in I_n$.
2. $\mathsf{Eval}(i, x) = h_i(x)$: On input i and $x \in D_n$, Eval algorithm outputs $h_i(x)$.
3. $\forall$ PPT $\mathcal{A}$ we have

   $\Pr[i \leftarrow \mathsf{Sampler}(1^n), (x, x') \leftarrow \mathcal{A}(1^n, i) : h_i(x) = h_i(x') \wedge x \neq x'] \leq \mathsf{negl}(n)(n)$

5.5.2 Collision Resistant Hash functions from Discrete Log

We will now give a construction of collision resistant hash functions from the discrete log assumption. We first recall the discrete log assumption:

Definition 5.3 (Discrete-Log Assumption). *We say that the discrete-log assumption holds for the group ensemble $\mathcal{G} = \{\mathbb{G}_n\}_{n \in \mathbb{N}}$, if for every non-uniform PPT algorithm $\mathcal{A}$ we have that*

$$\mu_{\mathcal{A}}(n) := \Pr_{x \leftarrow |\mathbb{G}_n|}[\mathcal{A}(g, g^x) = x]$$

is a negligible function.

We now give a construction of collision resistant hash functions.

- $\mathsf{Sampler}(1^n)$: On input 1^n, the sampler does the following:
  1. It chooses $x \leftarrow |\mathbb{G}_n|$.
  2. It computes $h = g^x$.
  3. It outputs (g, h).
- $\mathsf{Eval}((g, h), (r, s))$: On input (g, h) and two elements $(r, s) \in |\mathbb{G}_n|$, Eval outputs $g^r h^s$.

We now argue that this construction is collision resistant. Assume for the sake of contradiction that an adversary gives a collision $(r_1, s_1) \neq (r_2, s_2)$. We will now use this to compute the discrete logarithm of h. We first observe that:

$$\begin{aligned} r_1 + xs_1 &= r_2 + xs_2 \\ (r_1 - r_2) &= x(s_2 - s_1) \end{aligned}$$

We infer that $s_2 \neq s_1$. Otherwise, we get that $r_1 = r_2$ and hence, $(r_1, s_1) = (r_2, s_2)$. Thus, we can compute $x = \frac{r_1 - r_2}{s_1 - s_2}$ and hence the discrete logarithm of h is computable.

5.6 CRHF-Based Multiple-Message Digital Signature

We now explain how to combine collision-resistant hash functions and one-time signatures to get a signature scheme for multiple messages. We first construct an intermediate primitive wherein we will still have the same security property as that of one-time signature but we would be able to sign messages longer than the length of the public-key.[1]

5.6.1 One-time Signature Scheme for Long Messages

We first observe that the CRHF family H that we constructed earlier compresses $2n$ bits to n bits (also called as 2-1 CRHF). We will now give an extension that compresses an arbitrary long string to n bits using a 2-1 CRHF.

Merkle-Damgard CRHF. The sampler for this CRHF is same as that of 2-1 CRHF. Let h be the sampled hash function. To hash a string x, we do the following. Let x be a string of length m where m is an arbitrary polynomial in n. We will assume that $m = kn$ (for some k) or otherwise, we can pad x to this length. We will partition the string x into k blocks of length n each. For simplicity, we will assume that k is a perfect power of 2 or we will again pad x appropriately. We will view these k-blocks as the leaves of a complete binary tree of depth $\ell = \log_2 k$. Each intermediate node is associated with a bit string y of length at most ℓ and the root is associated with the empty string. We will assign a $\mathsf{tag} \in \{0,1\}^n$ to each node in the tree. The i-th leaf is assigned tag_i equal to the i-block of the string x. Each intermediate node y is assigned a $\mathsf{tag}_y = h(\mathsf{tag}_{y\|0} \| \mathsf{tag}_{y\|1})$. The output of the hash function is set to be the tag value of the root. Notice that if there is a collision for this CRHF then there are exists one intermediate node y such that for two different values $\mathsf{tag}_{y\|0}, \mathsf{tag}_{y\|1}$ and $\mathsf{tag}'_{y\|0}, \mathsf{tag}'_{y\|1}$ we have, $h(\mathsf{tag}_{y\|0}, \mathsf{tag}_{y\|1}) = \mathsf{tag}'_{y\|0}, \mathsf{tag}'_{y\|1}$. This implies that there is a collision for h.

Construction. We will now use the Merkle-Damgard CRHF and the one-time signature scheme that we constructed earlier to get a one-time signature scheme for signing longer messages. The main idea is simple: we will sample a (sk, vk) for signing n-bit messages and to sign a longer message, we will first hash it using the Merkle-

[1] Note that in the one-time signature scheme that we constructed earlier, the length of message that can be signed is same as the length of the public-key.

Damgard hash function to n-bits and then sign on the hash value. The security of the construction follows directly from the security of the one-time signature scheme since the CRHF is collision-resistant.

5.6.2 *Signature Scheme for Multiple Messages*

We will now describe the construction of signature scheme for multiple messages. Let $(\mathsf{Gen}', \mathsf{Sign}', \mathsf{Verify}')$ be a one-time signature scheme for signing longer messages.

1. $\mathsf{Gen}(1^n)$: Run $\mathsf{Gen}'(1^n)$ using to obtain sk, vk. Sample a PRF key K. The signing key is (sk, K) and the verification key is vk.

2. $\mathsf{Sign}((sk, K), m)$: To sign a message m, do the following:

   (a) Parse m as $m_1 m_2 \dots m_\ell$ where each $m_i \in \{0, 1\}$.

   (b) Set $sk_0 = sk$ and $m_0 = \epsilon$ (where ϵ is the empty string).

   (c) For each $i \in [\ell]$ do:

      i. Evaluate $\mathsf{PRF}(m_1 \| \dots \| m_{i-1} \| 0)$ and $\mathsf{PRF}(m_1 \| \dots \| m_{i-1} \| 1)$ to obtain r_0 and r_1 respectively. Run $\mathsf{Gen}'(1^n)$ using r_0 and r_1 as the randomness to obtain $(sk_{i,0}, vk_{i,1})$ and $(sk_{i,1}, vk_{i,1})$.

      ii. Set $\sigma_i = \mathsf{Sign}(sk_{i-1, m_{i-1}}, vk_{i,0} \| vk_{i,1})$

      iii. If $i = \ell$, then set $\sigma_{\ell+1} = \mathsf{Sign}(sk_{i, m_i}, m)$.

   (d) Output $\sigma = (\sigma_1, \dots, \sigma_{\ell+1})$ along with all the verification keys as the signature.

3. $\mathsf{Verify}(vk, \sigma, m)$: Check if all the signatures in σ are valid.

To prove security, we will first use the security of the PRF to replace the outputs with random strings. We will then use the security of the one-time signature scheme to argue that the adversary cannot mount an existential forgery.

5.7 *Trapdoor Permutations and RSA*

Definition 5.4 (Trapdoor Permutation). *A function family $\{f_s : D_s \to D_s\}_{s \in \{0,1\}^*}$ is a* one-way trapdoor permutation *if there exists PPT I, D, F, F^{-1} such that*

- *$(s, \tau) \leftarrow I(1^n)$ produces the seed and trapdoor,*
- *$D(s)$ outputs a uniformly random element of D_s,*
- *$\forall s \in I(1^n), x \in D_s$, $F(s, x) = f_s(x)$,*
- *$\forall (s, \tau) \in I(1^n), y \in D_s$, $F^{-1}(\tau, y) = f_s^{-1}(x)$, and*
- *f_s is one-way.*

The RSA trapdoor permutation construction is as follows:

- $I_{RSA}(1^n) \to (s = (N, e), \tau = (N, d)$ for $N = PQ$ for $2^{n-1} \leq P < Q \leq 2^n$ such that $d = e^{-1} \mod \phi(n)$ for $e < N$ which is coprime to $\phi(n) = (P-1)(Q-1)$. Let $D_s = \{1, \ldots, N\}$.
- $F_{RSA}(s, x) = x^e \mod N$.
- $F_{RSA}^{-1}(\tau, x) = y^d \mod N$.

Unfortunately, under the assumption that factoring is hard, we still don't have a security proof for the RSA trapdoor permutation in the plain model. However, we will see a proof in what's called the "random oracle model" next.

5.8 Random Oracle Model

We looked at RSA-FDH in the last section, and in this section we'll continue on and provide some semblance of a security analysis of the scheme.

As a note, collision resistance of the hash function isn't quite enough for the security of the RSA-FDH scheme. In particular, if we can find three messages m_1, m_2, m_3 such that $H(m_1) \cdot H(m_2) = H(m_3) \pmod N$ (this isn't protected against with collision resistance), then we can break the scheme, assuming that we use the RSA trapdoor function. Here, we'd have

$$\begin{aligned}\sigma_1\sigma_2 &= f^{-1}(H(m_1)) \cdot f^{-1}(H(m_2)) \\ &= H(m_1)^d H(m_2)^d \pmod N \\ &= (H(m_1)H(m_2))^d \pmod N \\ &= H(m_3)^d \pmod N\end{aligned}$$

Ideally, we'd like to have a proof of the security of this scheme, but nobody has been able to come up with one yet. Instead, we can only hope to find some kind of *evidence* for the security of the scheme.

This evidence comes from the *random oracle model* (ROM), otherwise known as the *random oracle methodology*.

Suppose we're given a scheme $\Pi^H = (A^H, B^H, C^H, \ldots)$, where calls to the hash function H is explicit. (Some functions may not call the hash function, but that's okay.)

We'd like to perform some analysis on these schemes, even though we may not fully understand the properties of the hash function—we'd like to abstract it out. To do this, we instead prove the security of $\Pi^O = (A^O, B^O, C^O, \ldots)$, where the hash function is replaced with an oracle O for a truly random function.

This oracle assumption is a very strong one, and is perhaps not the most indicative of the security of the original scheme—there are cases where the scheme Π^O under an oracle O is secure, but replacing the oracle with *any* instantiation breaks the security of the scheme.

When we're trying to prove security of Π^O, we'll look at an adversary $\mathcal{A}^O$, which has access to O. Here, observe that we can provide the answers to the oracle queries—we just need to find a contradiction to the existence of the function $\mathcal{A}$, regardless of what the oracle O does.

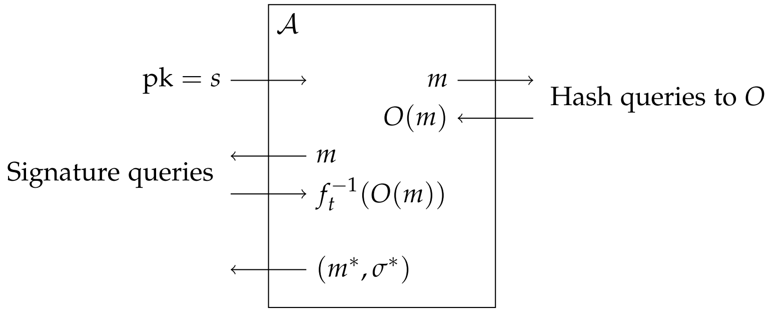
Note here that the adversary $\mathcal{A}$ in this case is forced to explicitly call the oracle for its hash function queries—the fact that we can see these calls is called *observability*. In the standard model, we can't actually see the queries that the adversary makes, since it just runs the predefined hash function itself.

Another property is called *programmability*: since we're working with a random oracle, the only thing that matters is that the output of the oracle looks uniformly random. This means that we can replace a uniform output x of the oracle with $f(x)$, for some one-way permutation f. This allows us to control some secret parameter that affects the output distribution of the oracle O. In the standard model, we don't have programmability—we again just have a fixed hash function that we can't change after the fact.

Theorem 5.1. *RSA-FDH is EUF-CMA secure in the ROM, assuming $\{f_s\}_s$ is a secure family of trapdoor permutations.*

Proof. Suppose we have an adversary $\mathcal{A}$ in this model.

The first thing it is given is a public key $\mathrm{pk} = s$. The adversary then gets to make signature queries: $m \mapsto f_t^{-1}(O(m))$. At the very end, it must output a forged signature (m^*, σ^*). This adversary is also allowed to make separate hash queries to the random oracle: $m \mapsto O(m)$.



WLOG, suppose that for every message m that $\mathcal{A}^O$ queries for a signature, it has already made a query for the same message to the hashing oracle. (Otherwise, we can simply make a wrapper around

$\mathcal{A}^O$ that does this.) We can also assume WLOG that when the adversary outputs (m^*, σ^*), it has also made the hashing query $O(m^*)$. Let's call this hybrid H_0.

For the hybrid H_1, we'll abort the machine if for any m, m' in the hash queries, we have $O(m) = O(m')$, essentially removing all collisions from the oracle. This happens with negligible probability $(q^2/2^n)$, so this hybrid is still indistinguishable from H_0.

Next, we'll construct an adversary $\mathcal{B}$ using $\mathcal{A}$, and inverts the trapdoor permutation. In particular, given (s, y^*), where $y^* = f^{-1}(x^*)$, the goal is to output x^*.

Suppose $\mathcal{A}$ makes q_s signing queries and q_h hashing queries.

We pass in s as pk. $\mathcal{B}$ first samples an $i^* \leftarrow \{1, \ldots, q_h\}$. We then set the output of the ith hash query to y^*. In particular, we have $O(m_{q_{i^*}}) = y^*$. If the adversary happens to call a signing query on i^*, we'll abort.

We still need to specify what happens on all other queries, and we want to make sure that we can respond with a signature query on all of these other queries. For $i \neq i^*$, we sample $x \in D_s$, and compute $y = f_s(x)$. On the ith hashing query, we then set $O(m_i) = y$. If the adversary later requests a signature on the same m_i, then we output x for the signing query. (This is because $f^{-1}(O(m_i)) = f^{-1}(y) = x$.)

In particular, the adversary must have called the hashing query for its output m^*, and with some probability, this is the i^*th query, in which case the message m^* is our inverse x^*.

Analyzing the probabilities, we have that

$$\begin{aligned}\Pr(\mathcal{B} \text{ outputs } f^{-1}(x^*)) &= \Pr(\mathcal{A} \text{ successful} \wedge \text{no sign query on } m_{i^*} \wedge m^* = m_{q_i^*}) \\ &= \varepsilon \times \left(1 - \frac{1}{q_h}\right)^{q_s} \times \frac{1}{q_h} \\ &\approx \frac{\varepsilon}{q_h}\end{aligned}$$

which is non-negligible, assuming $\mathcal{A}$ is successful with non-negligible probability. □

We'll now talk about a different scheme and analyze its security under the random oracle model.

This scheme is called the *Schnorr signature scheme*. Given a group G of prime order q and a hash function $H : \{0,1\}^* \to \mathbb{Z}_q$, we define

- $\textsc{Gen}(1^n) = (\text{pk} = g^x, \text{sk} = x \leftarrow \mathbb{Z}_q)$

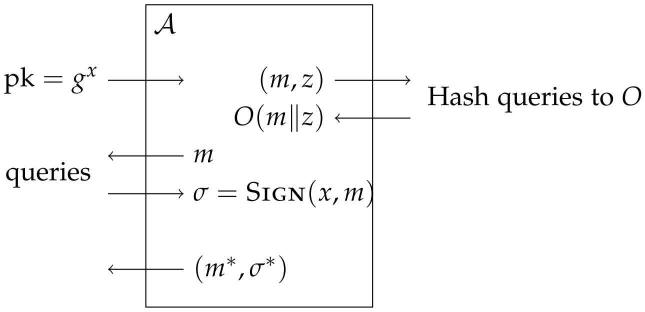
- $\textsc{Sign}(\text{sk}, m)$:
  $k \leftarrow \mathbb{Z}_q$
  $r = g^k$
  $h = H(m \| r)$

$s = k + hx$
$\sigma = (h, s)$

- $\textsc{Verify}(\mathrm{pk}, m, \sigma)$:
  output $h \overset{?}{=} H(m \| \frac{g^s}{\mathrm{pk}^h})$

Theorem 5.2. *The Schnorr signature scheme is EUF-CMA secure in the ROM, assuming the discrete log problem is hard.*

Proof. The adversary $\mathcal{A}^O$ gets a public key $\mathrm{pk} = g^x$, can make signing queries $m \mapsto \textsc{Sign}(x, m)$ and hashing queries $(m, z) \mapsto O(m \| z)$, for $z \in G$. $\mathcal{A}$ then returns a forgery (m^*, σ^*).



WLOG, we can assume that $m^* \| r^*$ is in the list of hash queries (where r^* was the value computed in the output signature σ^*).

We'll define a modified signing algorithm as follows:

```
function $\textsc{Sign}'(\mathrm{sk}, m)$
    $h, s \in \mathbb{Z}_q$ uniformly
    $g^k \leftarrow \frac{g^s}{g^{hx}} = \frac{g^s}{\mathrm{pk}^h}$
    $h = H(m \| g^k)$
    output $(h, s)$
end function
```

The main idea here is to provide a random signature, consistent with the definition of $\textsc{Sign}$, so that $\textsc{Verify}$ will still succeed.

We'll then define a wrapper $\mathcal{A}'^O$, which performs these modified signing queries by itself, since it no longer requires the secret key x. As such, $\mathcal{A}'^O$ only makes hashing queries, and produces (m^*, σ^*). In particular, $\mathcal{A}'$ depends on $\mathrm{pk}, q_1, h_1, \ldots, q_H, h_H$, but all of the queries $q_1, \ldots, q_H$ are deterministic depending on the previous hash output (or dependent on pk in the case of q_1). This means that $\mathcal{A}'$ can actually be thought of as a function of

$$\mathcal{A}'(\mathrm{pk}, h_1, h_2, \ldots, h_H).$$

The main insight that we'll use is that we can run $\mathcal{A}'$ until the $(i^* - 1)$th query, and on the i^*th query, we run the adversary twice, on two different possible responses: h_{i^*} and h'_{i^*}. These two executions share the first $i^* - 1$ hashing queries, and both are perfectly valid executions of the adversary. We'll use these two executions to break the discrete log problem.

Let us define $\mathcal{B}$ that breaks the discrete log problem, given as input (g, g^x). Here, we'll let g^x be the public key.

In response to hashing queries, if $\mathcal{A}'$ asks for the hash of $m \| z$, we respond with a random value (or the same value as before if queried multiple times) as $O(m \| z)$.

Now, we'd like to be able to find x, utilizing the behavior of $\mathcal{A}'$. At the i^*th query, we run the adversary twice, with h_{i^*} as the hash in the first execution, and h'_{i^*} as the hash in the second execution. In the first execution, we would have gotten queries $q_{i^*}, q_{i^*+1}, \ldots, q_h$, and outputted (m^*, σ^*). In the second execution we would have gotten queries $q'_{i^*}, q'_{i^*+1}, \ldots, q'_{h}$, and outputted (m'^*, σ'^*).

Now, in the i^*th query, note that $s = k + hx$ in the first execution, and $s' = k + h'x$ in the second execution. Crucially, the value of k is the same here, since the query utilizes the same value of r, and we can solve for $x = \frac{s-s'}{h-h'}$.

The probability that the adversary $\mathcal{A}'$ succeeds in producing a forgery while utilizing i^* is $\mu(n) = \varepsilon(n)/q_h$. For ease, let us also define the two halves of the input to $\mathcal{A}'$ as $\alpha = (\text{pk}, h_1, \ldots, h_{i^*-1})$ and $\beta = (h_{i^*}, \ldots, h_{q_h})$. We then define the "good set" as

$$S = \left\{ \alpha \mid \Pr_{\beta}(\mathcal{A}'(\alpha, \beta) \text{ outputs a forgery}) \geq \frac{\mu(n)}{2} \right\}.$$

We can also see that $\Pr(\alpha \in S) \geq \frac{\mu(n)}{2}$; to see why, suppose by contradiction $\Pr(\alpha \in S) < \frac{\mu(n)}{2}$. Here, we have

$$\begin{aligned} \Pr(\mathcal{A}' \text{ succeeds}) &= \Pr(\mathcal{A}' \text{ succeeds} \mid \alpha \in S) \Pr(\alpha \in S) + \Pr(\mathcal{A}' \text{ succeeds} \mid x \notin S) \Pr(\alpha \notin S) \\ &< 1 \cdot \frac{\mu(n)}{2} + \frac{\mu(n)}{2} \cdot 1 < \mu(n) \end{aligned}$$

which is a contradiction.

The probability that $\mathcal{B}$ succeeds is thus

$$\Pr(\alpha \in S) \Pr(\mathcal{A}'(\alpha, \beta) \text{ succeeds} \mid \alpha \in S) \Pr(\mathcal{A}'(\alpha, \beta') \text{ succeeds} \mid \alpha \in S) \geq \left(\frac{\mu(n)}{2} \right)^3,$$

due to the definition of S from earlier.

This means that in total, our probability of success is

$$\Pr(\mathcal{B} \text{ succeeds}) \geq \frac{\varepsilon^3(n)}{8q_h^3},$$

which is non-negligible if $\mathcal{A}'$ succeeds with non-negligible probability, giving us our contradiction. □

5.9 BLS Signatures

Bilinear Pairings. Recall that we used prime order groups $\mathbb{G}$ to build the Schnorr signature scheme. Today we will introduce another mathematical object, pairing friendly groups, which support a new "bilinear pairing" operation defined as follows. Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be groups of prime order p. We define a bilinear pairing $e : G_1 \times G_2 \to \mathbb{G}_T$ to be an *efficiently* computable map that satisfies the following properties:

- Bilinearity: For all $a, b \in \mathbb{Z}_p$ and $g_1 \in G_1, g_2 \in G_2$, we have $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$.
- Non-degenerate: $e(g_1, g_2) \neq 1_{\mathbb{G}_T}$.

Observe that in any group, by using the group operation, one can compute "additions" of the scalar exponents, but in pairing friendly groups one can additionally compute *one* "multiplication" of the scalar exponents.

Note that while a pairing provides us with additional *functionality*, it also means that we need to reevaluate any hardness assumptions that we have made. Consider for instance the decisional Diffie-Hellman (DDH) assumption which states that it is hard to distinguish between (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^c) for random $a, b, c \in \mathbb{Z}_p$. In the presence of a bilinear pairing, natural variants of the DDH assumption are no longer hard such as distinguishing between $(g_1, g_2, g_1^a, g_2^b, g_1^{ab})$ and $(g_1, g_2, g_1^a, g_2^b, g_1^c)$. Instead, we will define new assumptions that are conjectured to be hard even given a bilinear pairing.

co-CDH assumption. Let $\mathbb{G}_1$, $\mathbb{G}_2$ be groups of prime order with generators g_1 and g_2 and let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be an efficiently computable bilinear pairing. The co-CDH assumption states that for all non-uniform PPT adversaries $\mathcal{A}$:

$$\Pr\left[\mathcal{A}(g_1, g_2, g_1^a, g_1^b, g_2^b) \to g_1^{ab} \mid a, b \leftarrow\$ \mathbb{Z}_p\right] \leq \mathsf{negl}(n)$$

BLS Signature scheme. We are now ready to describe the BLS signature scheme [2].

- $\mathsf{Gen}(1^\lambda)$: Sample $\mathsf{sk} \leftarrow\$ \mathbb{Z}_p$ and set $\mathsf{vk} \leftarrow g_2^{\mathsf{sk}}$. Output $(\mathsf{vk}, \mathsf{sk})$.
- $\mathsf{Sign}(\mathsf{sk}, m)$: Given a message $m \in \{0,1\}^*$, output $\sigma \leftarrow H(m)^{\mathsf{sk}}$, where $H : \{0,1\}^* \to \mathbb{G}_1$ is a hash function that maps arbitrary strings to elements in $\mathbb{G}_1$.
- $\mathsf{Verify}(\mathsf{vk}, m, \sigma)$: Output $e(\sigma, g_2) \overset{?}{=} e(H(m), \mathsf{vk})$.

[2] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532, Gold Coast, Australia, December 9–13, 2001. Springer, Berlin, Heidelberg, Germany

Correctness. Correctness is easy to see by plugging in explicit expressions for the signature and verification key – $e(\sigma = H(m)^{\mathsf{sk}}, g_2) \overset{?}{=} e(H(m), \mathsf{vk} = g_2^{\mathsf{sk}})$.

Security. If H is modelled as a random oracle, we will show that if there exists an adversary $\mathcal{A}$ that can forge a signature with non-negligible probability, then we can use $\mathcal{A}$ to build $\mathcal{B}$ that can solve the co-CDH problem with non-negligible probability.

Theorem 5.3. *The BLS signature scheme is is existentially unforgeable under chosen message attacks assuming the co-CDH problem is hard in the random oracle model.*

Proof. We will use a similar strategy as in the proof of RSA full domain hash. Let $\mathcal{A}$ be a non-uniform PPT adversary that can forge a signature with non-negligible probability. We will use $\mathcal{A}$ to build a non-uniform PPT adversary $\mathcal{B}$ that can solve the co-CDH problem with non-negligible probability. $\mathcal{B}$ works as follows:

- $\mathcal{B}$ receives $(g_1, g_2, g_1^a, g_1^b, g_2^b)$ as input and is tasked with computing g_1^{ab}.
- $\mathcal{B}$ now runs the EUF-CMA game with $\mathcal{A}$ where $\mathcal{A}$ makes signing queries and in the end outputs a forgery (m^*, σ^*), where m^* has not been previously queried.
- At the start of the protocol, $\mathcal{B}$ sets $\mathsf{vk} = g_1^b$ and sends it to $\mathcal{A}$. For all random oracle queries $H(m)$, $\mathcal{A}$ samples $r \leftarrow\$ \mathbb{Z}_p$ and sets $H(m) = g_1^r$. When $\mathcal{A}$ asks for a signature on m_i, $\mathcal{B}$ samples $r_i \leftarrow\$ \mathbb{Z}_p$ and sets $H(m_i) = g_1^{r_i}$ (if the query has not been previously made). It then responds with $\sigma_i = H(m_i)^b = (g_1^b)^{r_i}$. However, for a randomly chosen index i^* (out of the maximum number of queries $\mathcal{A}$ can make), $\mathcal{B}$ sets $H(m_{i^*}) = g_1^a$. Now, if $\mathcal{A}$ asks for a signature on the chosen m_{i^*}, $\mathcal{B}$ aborts and restarts the EUF-CMA game. In the ends, $\mathcal{A}$ outputs a forgery with non-negligible probability. And since there are only a polynomial number of queries, $\mathcal{A}$ outputs a forgery on m_{i^*} with non-negligible probability. In which case $\mathcal{B}$ outputs σ_{i^*} as its output in the co-CDH game.

We now analyze the probability that $\mathcal{B}$ solves the co-CDH problem.

$$\begin{aligned}\Pr[\mathcal{B} \to g_1^{ab}] &= \Pr[\mathcal{A} \text{ outputs forgery} \wedge m_{i^*} \text{ was not queried} \wedge \text{forgery on } m_{i^*}] \\ &\geq \epsilon \times \left(1 - \frac{1}{q_h}\right)^{q_s} \times \frac{1}{q_h}\end{aligned}$$

which is non-negligible. Thus, by contradiction, the BLS signature scheme is existentially unforgeable under chosen message attacks assuming the co-CDH problem is hard in the random oracle model.

□

Exercises

Exercise 5.1. ***Digital signature schemes can be made deterministic.*** *Given a digital signature scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *for which* Sign *is probabilistic, provide a construction of a digital signature scheme* $(\mathsf{Gen}', \mathsf{Sign}', \mathsf{Verify}')$ *where* Sign' *is deterministic.*

6
Public Key Encryption

Public key encryption allows two parties to communicate with each other with the guarantees of privacy for their messages against an eavesdropper who monitors all communication between the two parties. Recall that we were able to build Digital Signatures and Symmetric Key Encryption, from the weakest building block in cryptography – one-way functions. However, there are lower bounds [1] suggesting that public key encryption cannot be built from one-way functions alone. Instead, we will build public key encryption from more structured assumptions (which are a *stronger* assumption in the sense that they imply one-way functions).

6.1 *Definitions*

Definition 6.1 (Public Key Encryption). *A public key encryption scheme is a tuple of three algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *defined as follows:*

- $\mathsf{Gen}(1^n) \rightarrow (\mathsf{pk}, \mathsf{sk})$*: outputs a public key and secret key* $(\mathsf{pk}, \mathsf{sk})$*.*
- $\mathsf{Enc}(\mathsf{pk}, m) \rightarrow c$*: Takes as input the public key and a message* m *and outputs a ciphertext* c*.*
- $\mathsf{Dec}(\mathsf{sk}, c) \rightarrow m$*: Takes as input a secret key* sk *and ciphertext* c*, and output a message* m*.*

Definition 6.2 (Perfect Correctness). *A public key encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is said to be correct if for all* $n \in \mathbb{N}$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^n)$, $m \in \{0,1\}^*$*, it holds that*

$$\Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m] = 1$$

The above definition can be relaxed to allow for a negligible probability of error during decryption and still remain meaningful.

We now define two different notions of security for public key encryption schemes – IND-CPA and IND-CCA security.

[1] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *21st Annual ACM Symposium on Theory of Computing*, pages 44–61, Seattle, WA, USA, May 15–17, 1989. ACM Press

Definition 6.3 (IND-CPA Security)**.** *A public key encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is said to be IND-CPA-secure if for all non-uniform PPT* $\mathcal{A}$*,*

$$\left|\Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{CPA}(n)} = 1] - \frac{1}{2}\right| \leq \mathsf{negl}(n),$$

where $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{CPA}(n)}$ *is defined as follows:*

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^n)$.
- $b \leftarrow\$ \{0, 1\}$.
- $\mathcal{A}(\mathsf{pk}) \rightarrow (m_0, m_1, \mathsf{st})$.
- $c^* \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$.
- $\mathcal{A}(c^*, \mathsf{st}) \rightarrow b'$.
- *Output* 1 *if* $b = b'$ *and* 0 *otherwise.*

Definition 6.4 (IND-CCA Security)**.** *A public key encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is said to be IND-CCA-secure if for all non-uniform PPT* $\mathcal{A}$*,*

$$\left|\Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{CCA}(n)} = 1] - \frac{1}{2}\right| \leq \mathsf{negl}(n),$$

where $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{CCA}(n)}$ *is defined as follows:*

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^n)$.
- $b \leftarrow\$ \{0, 1\}$.
- $\mathcal{A}^{\mathsf{Dec}(\mathsf{sk})}(\mathsf{pk}) \rightarrow (m_0, m_1, \mathsf{st})$.
- $c^* \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$.
- $\mathcal{A}^{\mathsf{Dec}(\mathsf{sk})}(c^*, \mathsf{st}) \rightarrow b'$*, where the decryption oracle now returns* $\perp$ *when queried on* c^**.*
- *Output* 1 *if* $b = b'$ *and* 0 *otherwise.*

6.2 Trapdoor Functions

Definition 6.5 (Trapdoor Function)**.** *A trapdoor function is a tuple of four algorithms* $(\mathsf{Gen}, f, f^{-1}, D)$ *defined as follows:*

- $\mathsf{Gen}(1^n) \to (s,t)$*: outputs an index* s *and corresponding trapdoor* t*.*
- $D(s)$*: Outputs a description of the domain of the trapdoor function.*
- $f(s,x) \to y$*: Takes as input an index* s *and a element* x *from the domain of the trapdoor function and outputs an element* y*.*
- $f^{-1}(t,y) \to x$*: Takes as input a trapdoor* t *and an element* y *from the range of the trapdoor function and outputs an element* x*.*

We require the following properties:

- ***Correctness:*** *For all* $n \in \mathbb{N}$, $(s,t) \leftarrow \mathsf{Gen}(1^n)$, $x \in D(s)$*, it holds that* $\Pr[f^{-1}(t, f(s,x)) \neq x] = \mathsf{negl}(n)$*. Note that if we demand that* f *is injective, then this property is automatically satisfied.*
- ***One-wayness:*** *For all non-uniform PPT* $\mathcal{A}$*, we have* $\Pr[\mathcal{A}(s,y) \to x \mid (s,t) \leftarrow \mathsf{Gen}(1^n), x \leftarrow\!\!\$\ D(s), y \leftarrow f(s,x)] = \mathsf{negl}(n)$.

6.3 Public Key Encryption from Trapdoor Functions

Given a trapdoor function, we can build a public key encryption scheme as follows:

- $\mathsf{Gen}(1^n)$: Run $\mathsf{TDF.Gen}(1^n)$ to obtain (s,t). Set $\mathsf{pk} = s$ and $\mathsf{sk} = t$.
- $\mathsf{Enc}(\mathsf{pk}, m)$: Sample $x \leftarrow\!\!\$\ D(s)$ and compute $y \leftarrow f(s,x)$. Sample $r \leftarrow\!\!\$\ \{0,1\}^{|x|}$ and output $c = (y, r, m \oplus \mathsf{HC}(x,r))$, where HC is the hardness concentration bit, as defined in Section 2.4.
- $\mathsf{Dec}(\mathsf{sk}, c)$: Parse $c = (y,r,z)$ and output $m = z \oplus \mathsf{HC}(f^{-1}(t,y), r)$.

Proof Sketch. To show that the above scheme is CPA secure, observe that the message space $\{0,1\}$. So the only two messages the distinguisher can pick are $m_0 = 0$ and $m_1 = 1$. Now if the distinguisher can indeed distinguish between encryptions of m_0 and m_1, then the same distinguisher can be used to identify the hardness concentration bit. By using a similar strategy as in the proof of Theorem 2.3, we can recover the pre-image x of the trapdoor function. Thus, violating the one-wayness of the trapdoor function.

6.4 Public Key Encryption from Computational Diffie-Hellman

Although we have a construction of public key encryption from trapdoor functions, we did not know how to build trapdoor functions from some very natural assumptions such as CDH for quite some time. It was only recently that Garg and Hajiabadi [2] showed that this was indeed possible. Given the non-trivial nature of the construction, we will not cover it in this course and instead provide a *direct* construction of public key encryption from the Computational Diffie-Hellman (CDH) assumption.

Let $\mathbb{G}$ be a prime order group of order p with generator g, where the CDH problem is assumed to be hard.

- $\mathsf{Gen}(1^n)$: Sample $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_p$ and set $\mathsf{pk} \leftarrow g^{\mathsf{sk}}$. Output $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Enc}(\mathsf{pk}, m)$: Sample $\alpha \leftarrow\!\!\$\ \mathbb{Z}_p$, and set $y \leftarrow g^{\alpha}$, $k \leftarrow \mathsf{pk}^{\alpha}$. Now sample $r \leftarrow\!\!\$\ \{0,1\}^{|B|}$ and output $c = (y, r, m \oplus \mathsf{HC}(k, r))$.
- $\mathsf{Dec}(\mathsf{sk}, c)$: Parse $c = (y, r, z)$ and compute $k \leftarrow y^{\mathsf{sk}}$. Output $m = z \oplus \mathsf{HC}(k, r)$.

Proof Sketch. The proof of security is similar to the proof of security for the trapdoor function based public key encryption scheme. The only difference is that we now we reduce the hardness of the CDH problem.

6.5 Improving Efficiency.

The above construction can only be used to encrypt a single bit and results in a ciphertext of size $O(\mathsf{poly}(n))$. Encrypting a message with M bits, results in a ciphertext of size $O(M \cdot \mathsf{poly}(n))$. Encryption and decryption also requires $O(M)$ public-key operations (evaluating the TDF/group operations) and we would like to bring this down to $O(1)$. The idea is to extract a symmetric key instead of a single bit by replacing the hardness concentration function with a random oracle. We can then use the symmetric key to encrypt long message with constant rate.

The construction is almost identical except that we replace the hardness concentration bit with a random oracle to extract a symmetric key that can be used to encrypt a long message. Given a trapdoor function the construction works as follows:

- $\mathsf{Gen}(1^n)$: Run $\mathsf{TDF.Gen}(1^n)$ to obtain (s, t). Set $\mathsf{pk} = s$ and $\mathsf{sk} = t$.
- $\mathsf{Enc}(\mathsf{pk}, m)$: Sample $x \leftarrow\!\!\$\ D(s)$ and compute $y \leftarrow f(s, x)$. Encrypt m using a symmetric encryption scheme with key $H(x)$ as $z \leftarrow \mathsf{Sym.Enc}(H(x), m)$. Output $c = (y, z)$.

[2] Sanjam Garg and Mohammad Hajiabadi. Trapdoor functions from the computational Diffie-Hellman assumption. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 362–391, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland

- $\mathsf{Dec}(\mathsf{sk}, c)$: Parse $c = (y, z)$ and compute $x \leftarrow f^{-1}(t, y)$. Decrypt z using the symmetric encryption scheme with key $H(x)$ as $m \leftarrow z \oplus \mathsf{Sym.Dec}(H(x), z)$. Output m.

The same idea can be extended to the CDH based public key encryption scheme.

Theorem 6.1. *The construction above is IND-CPA secure in the random oracle model assuming f is a trapdoor function.*

Proof. Given an adversary $\mathcal{A}$ that can win the IND-CPA game, with non-negligible advantage, we construct an adversary $\mathcal{B}$ that can break either the one-wayness of the trapdoor function or the security of the with non-negligible advantage.

Let s, y^* be the challenge given to $\mathcal{B}()$. $\mathcal{B}$ provides $\mathcal{A}$ with $\mathsf{pk} :- s$. $\mathcal{A}$ outputs (m_0, m_1) and in response $\mathcal{B}$ computes a challenge ciphertext as follows: Sample $b \leftarrow\$ \{0, 1\}$, a random key $k^* \leftarrow\$ \{0, 1\}^{|B|}$ and set $z^* \leftarrow \mathsf{Sym.Enc}(k^*, m_b)$. $\mathcal{B}$ also lazily samples responses to random oracle queries except if a query q satisfies $f(s, q) = y^*$, then $\mathcal{B}$ sets $H(q) = k^*$, and outputs q as the inverse image of y^* under f.

We now argue that $\mathcal{A}$ queries the random oracle on an element q such that $f(s, q) = y^*$, with non-negligible probability. If this never occurs, then $\mathcal{A}$ never receives receive k^* but is able to distinguish symmetric-key encryptions of m_0 from encryptions of m_1 with non-negligible probability. This violates the security of the symmetric encryption scheme. Thus, $\mathcal{A}$ must query the random oracle on an element q such that $f(s, q) = y^*$ with non-negligible probability and hence $\mathcal{B}$ succeeds with non-negligible probability. □

6.6 *Fujisaki-Okamoto Transformation*

6.7 *Kyber Encryption Scheme*

6.8 *Cramer-Shoup Construction*

7
Advanced Encryption Schemes

7.1 Identity-Based Encryption

We introduce Bilinear Maps and two of its applications: NIKE, Non-Interactive Key Exchange; and IBE, Identity Based Encryption.

7.2 Diffie-Hellman Key Exchange

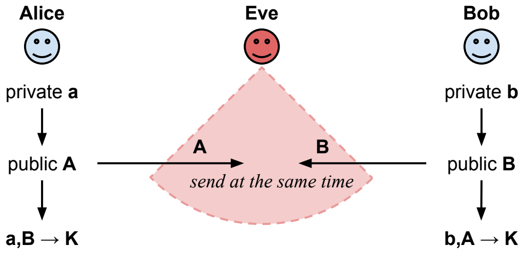


Figure 7.1: Diffie-Hellman Key Exchange

Fig 7.1 illustrates Diffie-Hellman key exchange. Alice and Bob each has a private key (a and b respectively), and they want to build a shared key for symmetric encryption communication. They can only communicate over a insecure link, which is eavesdropped by Eve. So Alice generates a public key A and Bob generates a public key B, and they send their public key to each other at the same time. Then Alice generates the shared key K from a and B, and likewise, Bob generates the shared key K from b and A. And we have $\forall$ PPT Eve, $Pr[k = Eve(A, B)] = neg(k)$, where k is the length of a.

7.2.1 Discussion 1

Assume that $\forall(g, p)$, and $a_1, b_1 \overset{\$}{\leftarrow} Z_p^*$, and $a_2, b_2, r \overset{\$}{\leftarrow} Z_p^*$, we have $(g^{a_1}, g^{b_1}, g^{a_1 b_1}) \overset{c}{\simeq} (g^{a_2}, g^{b_2}, g^{r})$. How to apply this to Diffie-Hellman Key Exchange?

Make $A = g^a$, $B = g^b$, $K = A^b = g^{ab}$, and $K = B^a = g^{ab}$.

7.2.2 Discussion 2

How does Diffie-Hellman Key Exchange imply Public Key Encryption?

Alice $pk = A$, $sk = a$, $Enc(pk, m \in \{0,1\})$.
Bob $b, r \leftarrow Z_p^*$ $(g^b, mA^b + (1-m)g^r)$
Alice $Dec(sk, (c_1, c_2))$
$c_1^a \overset{?}{=} c_2$

7.3 Bilinear Maps

Definition 7.1. *Bilinear Maps*

Bilinear Maps is (G, P, G_T, g, e), where e is an efficient function $G \times G \to G_T$ such that

- *if g is generator of G, then $e(g, g)$ is the generator of G_T.*
- *$\forall a, b \in Z_p$, we have $e(g^a, g^b) = e(g, g)^{ab} = e(g^b, g^a)$.*

7.3.1 Discussion 1

How does Bilinear Maps apply to Diffie-Hellman?

Make $A = g^a$, $B = g^b$, and $T = g^{ab}$, then Diffie-Hellman has $e(A, B) = e(g, T)$.

7.4 Tripartite Diffie-Hellman

Fig 7.2 illustrates Tripartite Diffie-Hellman key exchange. a, b, and c are private key of Alice, Bob, and Carol, respectively. They use g^a, g^b, g^c as public key, and the shared key $K = e(g, g)^{abc}$. Formally, we have

$$a, b, c \overset{\$}{\leftarrow} Z_p^*, r \overset{\$}{\leftarrow} Z_p^*$$

$$A = g^a, B = g^b, C = g^c$$

$$K = e(g, g)^{abc}$$

7.5 IBE: Identity-Based Encryption

IBE contains four steps: *Setup*, *KeyGen*, *Enc*, and *Dec*. We illustrate it in Figure 7.3. In first step, Key authority get a Master Public Key (MPK) and Master Signing Key (MSK) from $Setup(1^k)$. Then a user with an ID (in this example, "Mike"), sends his ID to the key authority. The key authority generates the Signing Key of Mike with

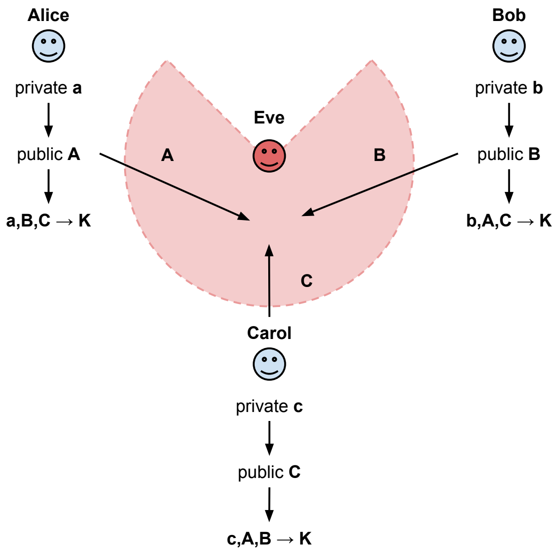


Figure 7.2: Tripartite Diffie-Hellman Key Exchange

$KeyGen(MSK, ID)$ ans sends it back. Another use, Alice, wants to send an encrypted message to Mike. She only has MPK and Mike's ID. So she encrypts the message with $c = Enc(MPK, ID = Mike, m)$, and sends the encrypted message c to Mike. Mike decodes c with $m = Dec(c, SK_{Mike})$. Notice that Alice never need to know Mike's public key. She only needs to remember MPK and other people's IDs.

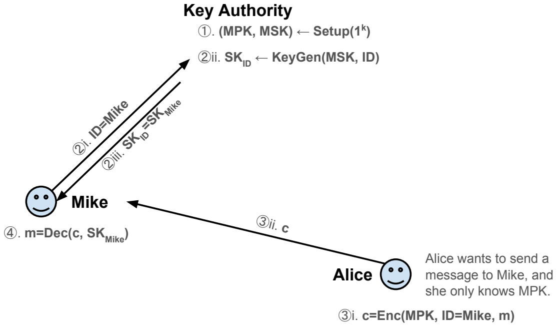

Key Authority
①. (MPK, MSK) ← Setup(1^k)
②ii. SK_ID ← KeyGen(MSK, ID)
②i. ID=Mike
②iii. SK_ID=SK_Mike
Mike
④. m=Dec(c, SK_Mike)
③ii. c
Alice
Alice wants to send a message to Mike, and she only knows MPK.
③i. c=Enc(MPK, ID=Mike, m)

Figure 7.3: Identity-Based Encryption

Formally, we have

$$Pr\begin{bmatrix} (MPK, MSK) \leftarrow Setup(1^k), \\ SK_{ID} \leftarrow KeyGen(MSK, ID), \\ c \leftarrow Enc(MPK, ID, m), \\ m \leftarrow Dec(SK_{ID}, c) \end{bmatrix} = 1$$

7.5.1 *Security Descriptions*

We have different security descriptions for IBE, as discussed in this section.

7.5.2 *CCA1*

$$\begin{array}{rcl}
\textbf{Challenger} & & \textbf{Adversary} \\
(MPK, MSK) \leftarrow Setup(1^k) & \xrightarrow{MPK} & \\
 & \xleftarrow{ID_1} & \\
SK_{ID_1} \leftarrow KeyGen(MSK, ID_1) & \xrightarrow{SK_{ID_1}} & \\
\vdots & \vdots & \vdots \\
 & \xleftarrow{ID_i} & \\
SK_{ID_i} \leftarrow KeyGen(MSK, ID_i) & \xrightarrow{SK_{ID_i}} & \\
 & \xleftarrow{ID^*, m_0, m_1} & \forall i \in [q], ID^* \neq ID_i \\
b \xleftarrow{\$} \{0,1\},\ c^* = Enc(MPK, ID^*, m_b) & \xrightarrow{c^*} & \\
\text{Output 1 if } b' = b, \text{ otherwise 0} & \xleftarrow{b'} &
\end{array}$$

7.5.3 *CCA2*

In CCA2, we allow adversary to send further queries after getting c^*.

$$\begin{array}{rcl}
\textbf{Challenger} & & \textbf{Adversary} \\
(MPK, MSK) \leftarrow Setup(1^k) & \xrightarrow{MPK} & \\
 & \xleftarrow{ID_1} & \\
SK_{ID_1} \leftarrow KeyGen(MSK, ID_1) & \xrightarrow{SK_{ID_1}} & \\
 & \vdots & \\
 & \xleftarrow{ID_i} & \\
SK_{ID_q} \leftarrow KeyGen(MSK, ID_q) & \xrightarrow{SK_{ID_q}} & \\
 & \xleftarrow{ID^*, m_0, m_1} & \forall i \in [q'], ID^* \neq ID_i \\
b \xleftarrow{\$} \{0,1\},\ c^* = Enc(MPK, ID^*, m_b) & \xrightarrow{c^*} & \\
 & \xleftarrow{ID_{q+1}} & \\
SK_{ID_{q+1}} \leftarrow KeyGen(MSK, ID_{q+1}) & \xrightarrow{SK_{ID_{q+1}}} & \\
\vdots & \vdots & \vdots \\
 & \xleftarrow{ID_{q'}} & \\
SK_{ID_{q'}} \leftarrow KeyGen(MSK, ID_{q'}) & \xrightarrow{SK_{ID_{q'}}} & \\
\text{Output 1 if } b' = b, \text{ otherwise 0} & \xleftarrow{b'} &
\end{array}$$

7.5.4 Selective Security

In selective security, the adversary sends ID^* before everything.

$$\begin{array}{rcl}
\textbf{Challenger} & & \textbf{Adversary} \\
& \xleftarrow{ID^*} & \forall i \in [q], ID^* \neq ID_i \\
(MPK, MSK) \leftarrow Setup(1^k) & \xrightarrow{MPK} & \\
& \xleftarrow{ID_1} & \\
SK_{ID_1} \leftarrow KeyGen(MSK, ID_1) & \xrightarrow{SK_{ID_1}} & \\
\vdots & \vdots & \vdots \\
& \xleftarrow{ID_q} & \\
SK_{ID_q} \leftarrow KeyGen(MSK, ID_q) & \xrightarrow{SK_{ID_q}} & \\
& \xleftarrow{m_0, m_1} & \\
b \xleftarrow{\$} \{0,1\}, c^* = Enc(MPK, ID^*, m_b) & \xrightarrow{c^*} & \\
\text{Output 1 if } b' = b, \text{ otherwise } 0 & \xleftarrow{b'} &
\end{array}$$

7.5.5 Discussion 1

How does Bilinear Maps apply to IBE?

Given Bilinear Maps: (G, P, G_T, g, e), we have

1. $(G, P, G_T, g, e) \leftarrow Setup(1^k)$
2. $s \leftarrow Z_p^*$, and $H_1 : \{0,1\}^* \to G$, $H_2 : G_T \to \{0,1\}^n$
3. $MPK = (G, g^s, H_1, H_2)$, and $MSK = (s)$

Let's look at how we construct each function in IBE.

$KeyGen(s, ID)$:

1. Output $SK_{ID} = (H_1(ID))^s$

$Enc(MPK, ID, m)$:

1. $r \leftarrow Z_p^*$
2. $c_1 = g^r$
3. $c_2 = m \oplus H_2(e(A, H_1(ID)^r))$, where $A = g^s$
4. Output (c_1, c_2)

$Dec(SK_{ID}, (c_1, c_2))$:

1. Get $e(A, H_1(ID)^r) = e(H_1(ID)^s, c_1) = e(SK_{ID}, c_1)$
2. Get $m = c_2 \oplus H_2(e(A, H_1(ID))^r)$

Proof. To prove this, we use a hybrid argument. Assume we have two oracles with exact random functions, denoted as O_{H_1} and O_{H_2}. One can request a random string from them with a query ID. The random strings are denoted as $H_1(ID)$ and $H_2(ID)$, respectively. These two oracles keep track of query IDs and corresponding responses. If a query ID was seen before, they return the exact same response corresponding to it. If not, they generate a random string, correspond the string to the ID, and return the string.

We first define $\mathcal{H}_0$, in which $H_1(ID)$ and $H_2(ID)$ are generated by the oracles. We use the construction described above.

$$\begin{array}{rcl}
\textbf{Challenger} & & \textbf{Adversary} \\
& \xleftarrow{\mathcal{G},ID^*} & \forall i \in [q], ID^* \neq ID_i \\
& \xleftarrow{g^s} & \\
& \xleftarrow{O_{H_1}} & \\
& \xleftarrow{O_{H_2}} & \\
& \xleftarrow{ID_1} & \\
SK_{ID_1} \leftarrow KeyGen(s, ID_1) & \xrightarrow{SK_{ID_1}} & \\
\vdots & \vdots & \vdots \\
& \xleftarrow{ID_q} & \\
SK_{ID_q} \leftarrow KeyGen(s, ID_q) & \xrightarrow{SK_{ID_q}} & \\
& \xleftarrow{m_0,m_1} & \\
b \overset{\$}{\leftarrow} \{0,1\}, c^* = Enc(MPK, ID^*, m_b) & \xrightarrow{c^*} & \\
\text{Output 1 if } b' = b \text{, otherwise 0} & \xleftarrow{b'} &
\end{array}$$

Then we discard oracle's H_1, and use $H_1(ID) = g^{\alpha_{ID}}$, where $\alpha_{ID} \leftarrow Z_p^*$. We denote this as $\mathcal{H}_1$.

Then we change SK_{ID} to $SK_{ID} = (H_1(ID))^s = (g^{\alpha_{ID}})^s$. We denote this as $\mathcal{H}_2$.

We have Bilinear Decision Diffie-Hellman (DDH). If $\mathcal{H}_2$ breaks DDH, then $\mathcal{H}_0$ can as well.

In DDH, we have $(g^a, g^b, g^c, e(g,g)^{abc}) \overset{c}{\simeq} (g^a, g^b, g^c, e(g,g)^r)$. We denote $A = g^a$, $B = g^b$, $C = g^c$. And in $\mathcal{H}_2$, we have $A = g^s$, $B = H_1(ID^*)$, $C = c_1 = g^r$. And in $c_2 = m \oplus H_2(e(g^s, H_1(ID^*))^r)$, we have $T = H_2(e(g^s, H_1(ID^*))^r) = e(g,g)^{abc}$.

□

8
Proving Computation Integrity

8.1 Commitment Schemes

8.2 Zero-Knowledge Proofs

Traditional Euclidean style proofs allow us to prove veracity of statements to others. However, such proof systems have two shortcomings: (1) the running time of the verifier needs to grow with the length of the proof, and (2) the proof itself needs to be disclosed to the verifier. In this chapter, we will provide methods enabling provers to prove veracity of statements of their choice to verifiers while avoiding the aforementioned limitations. In realizing such methods we will allow the prover and verifier to be probabilistic and also allow them to interact with each other.[1]

8.3 Interactive Proofs

Definition 8.1. **(Interactive Proof System)** *For a language L we have an interactive proof system if $\exists$ a pair of algorithms (or better, interacting machines) $(\mathcal{P}, \mathcal{V})$, where $\mathcal{V}$ is polynomial in $|x|$, and both can flip coins, such that:*

- *Completeness:* $\forall x \in L$

$$\Pr_{\mathcal{P},\mathcal{V}}[Output_{\mathcal{V}}(\mathcal{P}(x) \leftrightarrow \mathcal{V}(x)) = 1] = 1,$$

- *Soundness:* $\forall x \notin L$, $\forall \mathcal{P}^*$

$$\Pr_{\mathcal{V}}[Output_{\mathcal{V}}(\mathcal{P}^*(x) \leftrightarrow \mathcal{V}(x)) = 1] < neg(|x|),$$

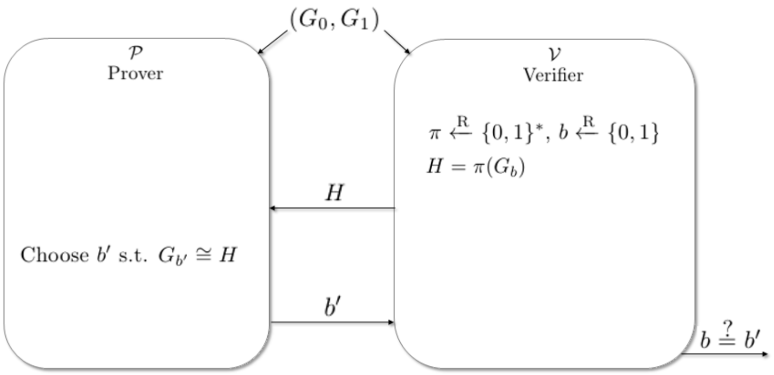
where $Output_{\mathcal{V}}(\mathcal{P}(x) \ \leftrightarrow \ \mathcal{V}(x))$ denotes the output of $\mathcal{V}$ in the interaction between $\mathcal{P}$ and $\mathcal{V}$ where both parties get x as input. We stress that $\mathcal{P}$ and $\mathcal{P}^*$ can be computationally unbounded.

[1] Formally, they can be modeled as interactive PPT Turing Machines.

Interactive Proof for Graph Non-Isomorphism (GNI). We say that two graphs G_0 and G_1 are isomorphic, denoted $G_0 \cong G_1$, if $\exists$ an isomorphism $f : V(G_0) \rightarrow V(G_1)$ s.t. $(u, v) \in E(G_0)$ iff $(f(u), f(v)) \in E(G_1)$, where $V(G)$ and $E(G)$ are the vertex and edge sets of some graph G. On the other hand, G_0 and G_1 are said to be non-isomorphic, $G_0 \ncong G_1$, if $\nexists$ any such f, and $GNI = \{(G_0, G_1) | \; G_0 \ncong G_1\}$ be the language that consists of pairs of graphs that are not isomorphic.

GNI is not believed to have short proofs so an interactive proof could offer a prover a mechanism to prove to a polynomially bounded verifier that two graphs are non-isomorphic.

The intuition behind a protocol to accomplish the above task is simple. Consider a verifier that randomly rename the vertices of one of the graphs and give it to the prover. Can the prover given the relabeled graph figure out which graph did the verifier start with? If G_0 and G_1 were not isomorphic then an unbounded prover can figure this out. However, in case G_0 and G_1 are isomorphic then the distribution resulting form random relabelings of G_0 and G_1 are actually identical. Therefore, even an unbounded prover has no way of distinguishing which graph the verifier started with. So the prover has only a $\frac{1}{2}$ probability of guessing which graph the verifier started with. Note that by repeating this process we can reduce the success probability of a cheating prover to negligible. More formally:



- Completeness: If $(G_0, G_1) \; \in$ GNI, then the unbounded $\mathcal{P}$ can distinguish isomorphism of G_0 against those of G_1 and can always return the correct b'. Thus, $\mathcal{V}$ will always output 1 for this case.

- Soundness: If $(G_0, G_1) \; \notin$ GNI, then it is equiprobable that H is a random isomorphism of G_0 as it is G_1 and so $\mathcal{P}$'s guess for b' can be correct only with a probability $\frac{1}{2}$. Repeating this protocol k times means the probability of guessing the correct b' for all k interactions is $\frac{1}{2^k}$. And so the probability of $\mathcal{V}$ outputting 0 (e.g. rejecting $\mathcal{P}$'s proof at the first sign of falter) is $1 - \frac{1}{2^k}$.

8.4 Zero Knowledge Proofs

Definition 8.2. **(NP-Verifier)** *A language L has an NP-verifier if ∃ a verifier $\mathcal{V}$ that is polynomial time in $|x|$ such that:*

- *Completeness:* $\forall x \in L,\ \exists$ *a* $proof\ \pi$ *s.t.* $\mathcal{V}(x, \pi) = 1$
- *Soundness:* $\forall x \notin L\ \forall$ *purported proof* π *we have* $\mathcal{V}(x, \pi) = 0$

That is, the conventional idea of a proof is formalized in terms of what a computer can efficiently verify. So a set of statements considered true (e.g. in a language L) is complete and sound if a proof can be written down that can be “easily" and rigorously verified if and only if a statement is in the language.

Efficient Provers. Unfortunately (fortunately?), there aren’t real-life instances of all-powerful provers that we know of. And for cryptography we must make more reasonable assumptions about the provers. In this case we will assume provers are also bounded to be *efficient*.

Previously, if a prover wanted to prove that two graphs, G_0 and G_1 were isomorphic, it would use its all-powerfulness to find the isomorphic mapping between the two graphs and give it to the verifier to complete the proof. But now, being computationally bounded, the prover is in the same boat as the verifier and can find a proof no better than the verifier can. In order for the prover to be able to prove something that the verifier cannot find out on their own, the prover must have some extra information. If, for example, the prover simply knew the isomorphism between the graphs, this would be the sufficient extra information it needs to enact the proof. That’s a rather boring proof though. We have interaction now! Can’t we do something fancier?

What if the prover wanted to prove that two graphs were isomorphic but didn’t want to fully reveal the isomorphism that they know. If they’re lying and don’t know an isomorphism is their a way we can exploit them again?

When G_0 and G_1 are isomorphic, the isomorphism between them would be a *witness*, w, to that fact, that can be used in the proof. Unfortunately, the prover is being stubborn and won’t just tell us that isomorphism, $w : V(G_0) \to V(G_1)$, that they claim to have. The prover is comfortable however giving us a “scrambled" version, ϕ, of w as long as it doesn’t leak any information about their precious w. For example, the prover is willing to divulge $\phi = \pi \circ w$ where π is a privately chosen random permutation of $|V| = |V(G_0)| = |V(G_1)|$ vertices. Since π renames vertices completely randomly, it scrambles what w is doing entirely and ϕ is just a random permutation of $|V|$

elements. At this point, we might be a little annoyed at the prover since we could have just created a random permutation on our own. This might give us an idea on how to gain a little more information however, even though we gained none here:

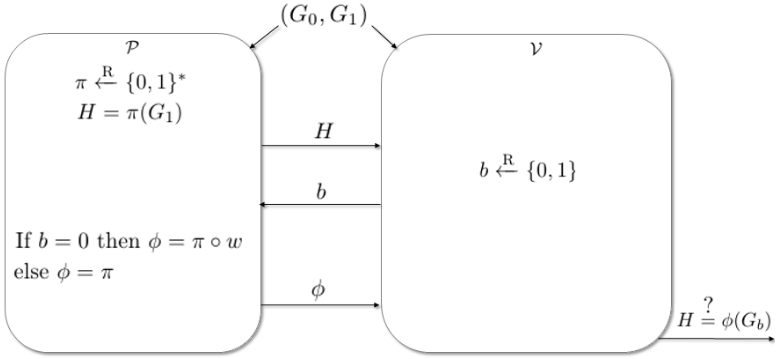
If we want to be convinced that ϕ really is of the form $\pi \circ w$, thus containing w in its definition, and isn't just a completely random permuation, we can note that if it is of that form then $\phi(G_0) = \pi(w(G_0)) = \pi(G_1)$ (since w being an isomorphism implies that $w(G_0) = G_1$). Note that we started with a mapping on input G_0 and ended with a mapping on input G_1. With an isormphism, one could get from one graph to the other seamlessly; if the prover *really* has the isomorphism it claims to have, then it should have no problem displaying this ability. So, what if we force the prover to give us $H = \pi(G_1)$ just after randomly choosing its π and then let it show us its ability to go from G_1 to G_0 with ease: give us a ϕ so that $\phi(G_0) = \pi(G_1) = H$. The only way the prover can give a mapping that jumps from G_0 to G_1 in such a way is if they know an isomorphism; if the prover could find a ϕ efficiently but did *not* know an isomorphism then they would have been able to see that $\pi^{-1}(\phi(G_0)) = G_1$ and thus have $\pi^{-1} \circ \phi$ as an isomorphism from G_0 to G_1, which would contradict the assumed hardness of finding isomorphisms in the GI problem. So by forcing the prover to give us H as we've defined and to produce a ϕ so that $\phi(G_0) = H$, we've found a way to expose provers that don't really have an isomorphism and we can then be convinced that they really do know w when they pass our test. And the prover didn't directly tell us w, so they may be able to salvage some secrecy!

But not everything is airtight about this interaction. Why, for instance, would the prover be willing to provide $H = \pi(G_1)$ when they're trying to divulge as little information as possible? The prover was comfortable giving us ϕ since we could have just simulated the process of getting a completely random permutation of vertices ourselves, but couldn't the additional information of H reveal information about w? At this point, the annoyed feeling may return as we realize that, $H = \pi(G_1) = \pi'(G_0)$, for some π', is just a random isomorphic copy of G_0 *and* G_1 as long as $G_0 \cong G_1$; we could have just chosen a random π', set $H = \pi'(G_0)$, and let $\phi = \pi'$ and would have created our very own random isomorphic copy, H, of G_1 that satisfies our test condition $H = \phi(G_0)$ just like what we got from our interaction with the prover. We couldn't have gained any new information from the prover because we could have run the whole test on our own!

Well, something must be wrong; we couldn't have been convinced of something without gaining *any* new information. Indeed, the test has a hole in it: how can we force the prover to give us $H = \pi(G_1)$ like we asked? If the prover is lying and it knows our test condition is to verify that $H = \phi(G_0)$, the prover might just cheat and give us $H = \pi(G_0)$ so it doesn't have to use knowledge of w to switch from G_1 to G_0. And, in fact, by doing this and sending $\phi = \pi$, the prover would fool us!

To keep the prover on their toes, though, we can randomly switch whether or not we want H to equal $\phi(G_0)$ or $\phi(G_1)$. If, in our interaction, the prover must first provide their $H = \pi(G_1)$ before we let them know which we want, they then lock themselves into a commitment to either G_0 or G_1 depending on whether they're trying to cheat or not, respectively. They only have a 50% chance of committing to the same case we want on a given round and so, if they don't have w to deftly switch between G_0 and G_1 to always answer correctly, they again have to be an extremely lucky guesser if they're trying to lie.

Again, we've created an interactive scheme that can catch dishonest provers with probability 1-$\frac{1}{2^k}$ and where we always believe honest provers!



(G_0, G_1)
$\mathcal{P}$
$\mathcal{V}$
$\pi \xleftarrow{R} \{0,1\}^*$
$H = \pi(G_1)$
H
$b \xleftarrow{R} \{0,1\}$
b
If $b = 0$ then $\phi = \pi \circ w$
else $\phi = \pi$
ϕ
$H \overset{?}{=} \phi(G_b)$

- Completeness: If $(G_0, G_1) \in$ GI and $\mathcal{P}$ knows w, then whether $\mathcal{V}$ chooses $b = 0$ or 1, $\mathcal{P}$ can always give the correct ϕ which, by definition, will always result in $H = \phi(G_b)$ and so $\mathcal{V}$ will always output 1.

- Soundness: If $(G_0, G_1) \notin$ GI, then $\mathcal{P}$ can only cheat, as discussed earlier, if the original H it commits to ends up being $\pi(G_b)$ for the b that is randomly chosen at the next step. Since b isn't even chosen yet, this can only happen by chance with probability $\frac{1}{2}$. And so the probability $\mathcal{V}$ outputs 0 is $1 - \frac{1}{2^k}$ for k rounds.

And so, again, we've correctly captured the idea of a proof by having this interaction. But there's a strange feeling that may be lingering around us...

As a verifier, we've seen some things in interacting with the prover. Surely, clever folks like ourselves must be able to glean *some* information about w after seeing enough to thoroughly convince us that the prover knows w. We've first seen H, and we've also seen the random b that we chose, along with ϕ at the end; this is our whole view of information during the interaction. But we're more bewildered than annoyed this time when we realize we could have always just chosen b and ϕ randomly and set $H = \phi(G_b)$ on our own. Again, everything checks out when $G_0 \cong G_1$ and we could have produced everything that we saw during the interaction before it even began. That is, the distribution of the random variable triple (H, b, ϕ) is identical whether it is what we saw from the prover during the interaction or it is yielded from the solitary process we just described. We've just constructed a complete interactive proof system that entirely convinces us of the prover's knowledge of w, yet we could have simulated the whole experience on our own! We couldn't have gain any knowledge about w since we didn't see anything we couldn't have manufactured on own, yet we are entirely convinced that $(G_0, G_1) \in$ GI and that $\mathcal{P}$ knows w! And so the prover has proven something to us yet has given us absolutely zero additional knowledge!

This may feel very surprising or as if you've been swindled by a fast talker, and it very much should feel this way; it was certainly an amazing research discovery! But this is true, and it can be made rigorous:

We should first be sure what we want out of this new proof system. We of course want it to be complete and sound so that we accept proofs iff they're true. But we also want the verifier to gain zero knowledge from the interaction; that is, the verifier should have been able to simulate the whole experience on its own without the verifier. Finally, we would also like all witnesses to a true statement to each be sufficient to prove the veracity of that statement and so we let R be the relation s.t. $x \in L$ iff $\exists$ a witness w s.t. $(x, w) \in R$. We can then gather all witness by defining $R(x)$ to be the set of all such witnesses.

Definition 8.3. (Honest Verifier Zero Knowledge Proof [HVZK]) *For a language L we have a (perfect) HVZK proof system w.r.t. witness relation R if $\exists$ an interactive proof system, $(\mathcal{P}, \mathcal{V})$ s.t. $\exists$ a PPT machine $\mathcal{S}$ (called the simulator) s.t. $\forall x \in L$, $\forall w \in R(x)$ the following distributions are identical:*

$$View_{\mathcal{V}}(\mathcal{P}(x, w) \leftrightarrow \mathcal{V}(x))$$

$$\mathcal{S}(x)$$

where $View_{\mathcal{V}}(\mathcal{P}(x, w) \leftrightarrow \mathcal{V}(x))$ is the random coins of $\mathcal{V}$ and all the messages $\mathcal{V}$ saw.

Remark 8.1. *In the above definition, $\text{View}_{\mathcal{V}}(\mathcal{P}(x,w) \leftrightarrow \mathcal{V}(x))$ contains both the random coins of $\mathcal{V}$ and all the messages that $\mathcal{V}$ saw, because they together constitute the view of $\mathcal{V}$, and they are correlated. If the random coins of $\mathcal{V}$ are not included in the definition of $\text{View}_{\mathcal{V}}(\mathcal{P}(x,w) \leftrightarrow \mathcal{V}(x))$, then even if $\mathcal{S}$ can generate all messages that $\mathcal{V}$ saw with the same distribution as in the real execution, the verifier may still be able to distinguish the two views using its random coins.*

There is an interesting progression of the requirements of a proof system: Completeness, Soundness, and the Zero Knowledge property. Completeness first cares that a prover-verifier pair exist and can capture all true things as a team that works together; they both honestly obey the protocol trying prove true statements. Soundness, however, assumes that the prover is a liar and cares about having a strong enough verifier that can stand up to any type of prover and not be misled. Finally, Zero Knowledge assumes that the verifier is hoping to glean information from the proof to learn the prover's secrets and this requirement makes sure the prover is clever enough that it gives no information away in its proof.

Unlike the soundness' requirment for a verifier to combat *all* malicious provers, HVZK is only concerned with the verifier in the original prover-verifier pair that follows the set protocol. Verifiers that stray from the protocol or cheat, however, are captured in the natural generalization to Zero Knowledge proofs.

Definition 8.4 (Efficient Prover Zero-Knowledge Proof)**.** *We say (P, V) is an efficient prover zero-knowledge proof system for a language L and relation R_L if*

1. *The prover P runs in polynomial time.*

2. *The protocol is* complete. *That is, for every $x \in L$ there exists a witness $w \in R_L(x)$ such that*

$$\Pr[P(x,w) \leftrightarrow V(x) \text{ accepts}] = 1.$$

3. *The protocol is* sound *against unbounded provers. That is, for $\forall x \notin L$, we have*

$$\Pr[P^*(x,w) \leftrightarrow V(x) \text{ rejects}] \geq 1/2$$

for any prover P^* of arbitrary computation power and any witness w.

4. *There exists an expected polynomial time probabilistic machine S (a simulator) such that for all PPT V^*, for all $x \in L, w \in R_L(x), z \in \{0,1\}^*$ we have*

$$\{\text{View}_{V^*}(P(x,w) \leftrightarrow V^*(x,z))\} \simeq_c \{S^{V^*}(x,z)\}$$

The soundness probability can be amplified to be greater than any

$1 - 1/2^k$, for arbitrary $k > 0$, by repeating the proof k times. More precisely, we construct an efficient prover zero-knowledge proof system $(\tilde{P}, \tilde{V})$ which repeats (P, V) independently for k times, and $\tilde{V}$ accepts if and only if V accepts in all the executions.

It is easy to see that $\tilde{P}$ runs in polynomial time and that the protocol is complete. Moreover, it has the following soundness guarantee: for $\forall x \notin L$,

$$\begin{aligned}&\Pr\left[\tilde{P}^*(x,w) \leftrightarrow \tilde{V}(x) \text{ rejects}\right]\\ =&1 - \Pr\left[\forall 1 \leq i \leq k, P_i^*(x,w) \leftrightarrow V(x) \text{ accepts}\right]\\ =&1 - \prod_{i=1}^{k} \Pr\left[P_i^*(x,w) \leftrightarrow V(x) \text{ accepts}\right]\\ \geq&1 - \frac{1}{2^k}\end{aligned}$$

for any prover $\tilde{P}^* = (P_1^*, \cdots, P_k^*)$ of arbitrary computation power and any witness w.

Finally, it is zero-knowledge, namely, there exists an expected PPT $\tilde{S}$ such that for all PPT $\tilde{V}^*$, and for all $x \in L, w \in R_L(x), z \in \{0,1\}^*$,

$$\left\{View_{\tilde{V}^*}(\tilde{P}(x,w) \leftrightarrow \tilde{V}^*(x,z))\right\} \simeq_c \left\{\tilde{S}^{\tilde{V}^*}(x,z)\right\}.$$

The construction of $\tilde{S}$ is repeating S for k times. We prove by hybrid argument that the above two distributions are indistinguishable. H_i is defined to be the output of repeating S for the first i executions with $\tilde{V}^*$ and repeating P for the rest $k - i$ executions. Then H_0 is the left distribution and H_k is the right one. Any attacher that can distinguish the above two distributions leads to an attacker that can distinguish H_{i-1} and H_i for some $1 \leq i \leq k$, which violates the zero-knowledge property of the original proof system (P, V).

The order of the quantifiers in item 4 matters. If we quantify over x and w before quantifying over the simulator, then we could hardcode x and w into our simulator. That is, for all $x \in L, w \in R_L(x)$, there exists an expected polynomial time probabilistic machine $S_{x,w}$ such that for all PPT V^* and $z \in \{0,1\}^*$,

$$\{View_{V^*}(P(x,w) \leftrightarrow V^*(x,z))\} \simeq_c \{S_{x,w}^{V^*}(x,z)\}$$

Since we would like our simulator to be universal, this is not acceptable.

If we quantify first over the verifier V^* and then over simulators S, then this variant is considered as *non-black-box zero-knowledge*. Our standard definition is considered as *black-box zero-knowledge*. There also exist variants that use statistical indistinguishability rather than computational indistinguishability.

The z in item 4 is considered as *auxiliary input*. The auxiliary input is crucial for the above argument of soundness amplification.

We will discuss the importance of requiring expected polynomial time in the next section.

8.5 Graph Isomorphism

Recall our protocol for graph isomorphism: the interaction is $P(x, w) \leftrightarrow V(x)$ where x represents graphs $G_0 = (V, E_0)$ and $G_1 = (V, E_1)$ and w represents a permutation π on V such that $\pi(G_0) = G_1$.

1. P samples a random permutation $\sigma : V \to V$ and sends the graph $H = \sigma(G_1)$ to V.

2. V samples a random bit b and sends it to P.

3. If $b = 1$, then P defines a permutation τ to be σ. If $b = 0$, then instead $\tau = \sigma \circ \pi$. P then sends τ to V.

4. V verifies that $\tau(G_b) = H$ and accepts if so.

We will show that this is an efficient prover zero-knowledge proof system. It is clear that if G_0 and G_1 are isomorphic, then this protocol will succeed with probability 1.

For soundness, observe that if G_0 is not isomorphic to G_1, then the graph H that P sends to V in step 1 of the protocol can be isomorphic to at most one of G_0 or G_1. Since V samples a bit b uniformly at random in step 2, then there is a probability of at most 1/2 that P can produce a valid isomorphism in step 3.

For zero knowledge, consider the following simulator S with input G_0 and G_1 (with vertex set V) and verifier V^*:

1. Guess a bit b uniformly at random.

2. Sample a permutation $\pi : V \to V$ uniformly at random and send $\pi(G_b)$ to V^*.

3. Receive b' form V^*.

4. If $b = b'$, then output $(\pi(G_b), b, \pi)$ and terminate. Otherwise, restart at step 1.

Note that if $G_0 \simeq G_1$, then $\pi(G_b)$ is statistically independent of b because b and π are sampled uniformly. Thus, with probability 1/2, V^* will output b so on average, two attempts will be needed before S terminates. It follows that S will terminate in *expected* polynomial time.

Since b is sampled uniformly at random, $\pi(G_b)$ is uniformly distributed with all graphs of the form $\sigma(G_1)$ where σ is sampled uniformly at random from permutations on V. Thus, the output $\pi(G_b)$ in our simulator will be identically distributed with the output H in our graph isomorphism protocol.

In step 3 of our graph isomorphism protocol, note that τ is distributed uniformly at random. This is because composing a uniformly random permutation with a fixed permutation will not change its distribution. Thus τ will be identically distributed with π in our simulator. It follows that the transcripts outputted by our simulator will be identically distributed with the transcripts produced by the graph isomorphism protocol.

8.6 Zero-Knowledge for NP

An n-coloring of a graph $G = (A, E)$ is a function $c : A \to \{1, \dots, n\}$ such that if $(i, j) \in E$, then $c(i) \neq c(j)$. So we want to paint each vertex of a graph a certain color so that the endpoints of any edge are colored differently.

In the graph 3-coloring problem (3COL), we are given a graph and asked if there exists a 3-coloring. In this section, we will provide a computational zero knowledge proof for 3COL. It is a fact that 3COL is NP-complete, so any problem in NP has a polynomial time reduction to 3COL. Thus, by giving a zero knowledge proof for 3COL, we will show that there are zero knowledge proofs for all of NP.

We will first give a high-level description of a zero-knowledge protocol for 3COL. Suppose a prover P wants to convince a verifier V that his graph G is 3-colorable without revealing what the coloring c actually is. If the three colors we use are red, green, and blue, then note that if we colored all the red vertices blue, all the green vertices red, and all the blue vertices green, we would still have a valid 3-coloring. In fact, if ϕ was any permutation on the color set of red, green, and blue, then $\phi \circ c$ would be a valid 3-coloring of G.

P asks V to leave the room and then samples a random permutation ϕ of the three colors. He colors the vertices of G according to $\phi \circ c$, then covers all the vertices with cups. At this point, P invites V back into the room. V is allowed to pick one edge and then uncover the two endpoints of the edge. If the colors on the two endpoints are the same, then V rejects P's claim that the graph is 3-colorable.

If the colors on the two endpoints are different, then V leaves the room again, P samples ϕ randomly, and the process repeats itself. Certainly if G is actually 3-colorable, then V will never reject the claim. If G is not 3-colorable, then there will always be an edge with endpoints that are colored identically and V will eventually uncover such an edge.

Note that V does not gain any information on the coloring because it is masked by a (possibly) different random permutation every time V uncovers an edge. Of course this protocol depends on P not being able to quickly recolor the endpoints of an edge after removing the cups. This is why we need commitment schemes.

8.6.1 Commitment Schemes

We want to construct a protocol between a sender and a receiver where the sender sends a bit to the receiver, but the receiver will not know the value of this bit until the sender chooses to "open" the data that he sent. Of course, this protocol is no good unless the receiver can be sure that the sender was not able to change the value of his bit in between when the receiver first obtained the data and when the sender chose to open it.

Definition 8.5. *A* commitment scheme *is a PPT machine C taking input (b, r) that satisfies two properties:*

- *(perfect binding) For all r, s, we have $C(0, r) \neq C(1, s)$.*
- *(computational hiding) $\{C(0, U_n)\} \simeq_c \{C(1, U_n)\}$*

So for the sender to "open" the data, he just has to send his value of r to the receiver. We say that r is a *decommitment* for $C(x, r)$. Why do we require perfect binding instead of just statistical binding? If there existed even a single pair r, s where $C(0, r) = C(1, s)$, then the sender could cheat. If he wished to reveal a bit value of 0 then he could just offer r and if he wished to reveal a bit value of 1 then he could just offer s.

We can use injective one-way functions to construct commitment schemes.

Theorem 8.1. *If injective one-way functions exist, then so do commitment schemes.*

Proof. We can let f be an injective one-way function. Recall from Lecture 3 that $f'(x, r) := (f(x), r)$ will also be an injective one-way function with hard-core bit $B(x, r) := \langle x, r \rangle$. We claim that $C(b, x, r) := (f'(x, r), b \oplus B(x, r))$ is a commitment scheme.

If $(x, r) \neq (y, s)$ then $C(0, x, r) \neq C(0, y, s)$ because f' is injective. Since $C(0, x, r) = (f'(x, r), B(x, r)) \neq (f'(x, r), \overline{B(x, r)}) = C(1, x, r)$, then C satisfies perfect binding.

Suppose D can distinguish $C(0, U_n)$ from $C(1, U_n)$. Then we can distinguish $B(x, r)$ from $\overline{B(x, r)}$ given $f'(x, r)$ which contradicts the fact that $B(x, r)$ is a hard-core bit for $f'(x, r)$. Thus, C has the computational hiding property. □

We can extend the definition of commitment schemes to hold for messages longer than a single bit. These commitment schemes will work by taking our commitment schemes for bits and concatenating them together. For the extended definition, we require that for any two messages m_0 and m_1 of the same length, the ensembles $\{C(m_0, U_n)\}$ and $\{C(m_1, U_n)\}$ are computationally indistinguishable.

8.6.2 3COL Protocol

Below we describe the protocol $P(x, z) \leftrightarrow V(x)$, where x describes a graph $G = (\{1, \ldots, n\}, E)$ and z describes a 3-coloring c:

1. P picks a random permutation $\pi : \{1, 2, 3\} \to \{1, 2, 3\}$ and defines the 3-coloring $\beta := \pi \circ c$ of G. Using a commitment scheme C for the messages $\{1, 2, 3\}$, P defines $\alpha_i = C(\beta(i), U_n)$ for each $i \in V$. P sends $\alpha_1, \alpha_2, \ldots, \alpha_n$ to V.

2. V uniformly samples an edge $e = (i, j) \in E$ and sends it to P.

3. P opens α_i and α_j.

4. V will accept only if it received valid decommitments for α_i and α_j, and if $\beta(i)$ and $\beta(j)$ are distinct and valid colors.

It is clear that this protocol is PPT. If G is not 3-colorable, then there will be at least a $1/|E|$ probability that V will reject P's claim in step 4. Since $|E| \leq n^2$ we can repeat the protocol polynomially many times to increase the rejection probability to at least 1/2.

We will now show that this protocol is zero-knowledge. We describe a simulator S below, given a verifier V^*:

1. Sample an edge $e = (i, j) \in E$ uniformly at random.

2. Assign c_i and c_j to have distinct values from $\{1, 2, 3\}$ and do so uniformly at random. Set $c_k := 1$ for all $k \neq i, j$.

3. Compute n random keys $r_1, \ldots, r_n$ and set $\alpha_i = C(c_i, r_i)$ for all i.

4. Let $e' \in E$ be the response of V^* upon receiving $\alpha_1, \ldots, \alpha_n$.

5. If $e' \neq e$, then terminate and go back to step 1. Otherwise, proceed. If S returns to step 1 more than $2n|E|$ times, then output fail and halt the program.

6. Print $\alpha_1, \ldots, \alpha_n, e$, send r_i and r_j to V^* and then print whatever V^* responds with.

By construction, S will run in polynomial time. However, sometimes it may output a fail message. We will show that this occurs with negligible probability.

Suppose that for infinitely many graphs G, V^* outputs $e' = e$ in step 4 with probability less than $1/2|E|$. If this is true, then it is possible for us to break the commitment scheme C that we use in S. Consider a modified version of S called $\tilde{S}$, where in step 2 we set $c_i = 1$ for all i. Note that in this case, V^* cannot distinguish between any of the edges so the probability that it returns $e' = e$ is $1/|E|$.

If we gave V^* a set of commitments $\alpha_k = C(1, r_k)$ for random keys r_k, then we would be in the setting of $\tilde{S}$. If we gave V^* the commitments α_k but with two of the values set to $C(c, r)$ and $C(c', r')$ where c, c' are distinct random values from $\{1, 2, 3\}$ and r, r' are random keys, then we are in the setting of S. This implies that it possible to distinguish between these two commitment settings with a probability of at least $1/2|E|$ which is non-negligible. It follows that V^* outputs $e' = e$ with probability less than $1/2|E|$ for only finitely many graphs G.

Thus, the probability that S outputs fail in the end is less than $(1 - 1/2|E|)^{2n|E|} < 1/e^n$ which is negligible.

Now we need to argue that the transcripts generated by S are computationally indistinguishable from the transcripts generated by $P \leftrightarrow V^*$. Again, we consider a modified version of S, called S', given a 3-coloring of its input G as auxiliary input. In step 2 of the simulation, S' will choose a random permutation of the colors in its valid 3-coloring for the values of c_i rather than setting all but two values c_i and c_j equal to 1. Note that this is how our protocol between P and V behaves.

Observe that $P \leftrightarrow V^*$ is computationally indistinguishable from S' because S' outputs fail with negligible probability. Thus, it suffices to show that S and S' are computationally indistinguishable. Again, we will suppose otherwise and argue that as a result we can distinguish commitments.

We consider two messages m_0 and m_1 of the same length where m_0 consists of $n - 2$ instances of the message 1 and two committed colors c_i and c_j (for a random edge $(i, j) \in E$) and m_1 consists of a committed random 3-coloring of G (with a random edge $(i, j) \in E$) chosen. Observe that by feeding the former message to V^* we are in the setting of S' and by feeding the latter message to V^* we are in the setting of S. If we could distinguish those two settings, then we could distinguish the commitments for m_0 and m_1. This contradiction completes our argument that our 3-coloring protocol is zero-knowledge.

8.7 NIZK Proof Systems

We now consider a different class of Zero-Knowledge proof systems, where no interaction is required: The Prover simply sends one message to the Verifier, and the Verifier either accepts or rejects. Clearly for this class to be interesting, we must have some additional structure: both the Prover and Verifier additionally have access to a common random public string σ (trusted to be random by both). For example, they could derive σ by looking at sunspot patterns.

8.8 Definitions

Definition 8.6 (NIZK Proof System). *A* NIZK proof system *for input x in language L, with witness ω, is a set of efficient (PPT) algorithms (K, P, V) such that:*

1. *Key Generation: $\sigma \leftarrow K(1^k)$ generates the random public string.*
2. *Prover: $\pi \leftarrow P(\sigma, x, \omega)$ produces the proof.*
3. *Verifier: $V(\sigma, x, \pi)$ outputs $\{0, 1\}$ to accept/reject the proof.*

Which satisfies the completeness, soundness, and zero-knowledge properties below.

Note: We will assume throughout that x is of polynomially-bounded length, i.e., we are considering the language $L \cap \{0, 1\}^{P(k)}$.

Completeness. $\forall x \in L, \forall \omega \in R_L(x)$:

$$\Pr[\sigma \leftarrow K(1^k), \pi \leftarrow P(\sigma, x, \omega) : V(\sigma, x, \pi) = 1] = 1.$$

Non-Adaptive Soundness. $\forall x \notin L$:

$$\Pr[\sigma \leftarrow K(1^k) : \exists\ \pi \mathsf{st} V(\sigma, x, \pi) = 1] = \mathsf{negl}(k).$$

The above definition is "non-adaptive", because it does not allow a cheating prover to decide which statement to prove after seeing the randomness σ. We may also consider the stronger notion of "adaptive soundness", where the prover is allowed to decide x after seeing σ:

Adaptive Soundness.

$$\Pr[\sigma \leftarrow K(1^k) : \exists\ (x, \pi) \mathsf{st} x \notin L, V(\sigma, x, \pi) = 1] = \mathsf{negl}(k).$$

(Non-Adaptive) Zero-Knowledge. The exists a PPT simulator S such that $\forall x \in L, \omega \in R_L(x)$, the two distributions are computationally indistinguishable:

1. $\sigma \leftarrow K(1^k)$
2. $\pi \leftarrow P(\sigma, x, \omega)$
3. Output (σ, π)

1. $(\sigma, \pi) \leftarrow S(1^k, x)$
2. Output (σ, π)

That is, the simulator is allowed to generate the distribution of randomness σ together with π. Note that if we did not allow S to produce σ, this definition would be trivial (a verifier could convince himself by running the simulator, instead of interacting with P). Allowing S to generate σ still keeps the definition zero-knowledge (since a verifier sees both (σ, π) together), but puts P and S on unequal footing.

We could also consider the adaptive counterpart, where a cheating verifier can choose x after seeing σ:

(Adaptive) Zero-Knowledge. The exists a PPT simulator split into two stages S_1, S_2 such that for all PPT attackers $\mathcal{A}$, the two distributions are computationally indistinguishable:

1. $\sigma \leftarrow K(1^k)$
2. $(x, \omega) \leftarrow \mathcal{A}(\sigma)$, s.t. $(x, \omega) \in R_L$
3. $\pi \leftarrow P(\sigma, x, \omega)$
4. Output (σ, x, π)

1. $(\sigma, \tau) \leftarrow S_1(1^k)$
2. $(x, \omega) \leftarrow \mathcal{A}(\sigma)$
3. $\pi \leftarrow S_2(\sigma, x, \tau)$
4. Output (σ, x, π)

where τ should be thought of as local state stored by the simulator (passed between stages).

Now we show that adaptive soundness is not harder to achieve than non-adaptive soundness.

Theorem 8.2. *Given a NIZK (K, P, V) that is* non-adaptively sound*, we can construct a NIZK that is* adaptively sound*.*

Proof. For $x_0 \notin L$, let us call a particular σ "bad for x_0" if there exists a false proof for x_0 using randomness σ: $\exists\ \pi \text{st} V(\sigma, x_0, \pi) = 1$. By non-adaptive soundness of (K, P, V), we have $\Pr_\sigma[\sigma \text{ bad for } x_0] = \mathsf{negl}(k)$.

Now we construct a new NIZK (K', P', V') by repeating (K, P, V) polynomially-many times (using fresh randomness, and V' accepts if and only if V accepts in each iteration). We can ensure that $\mathsf{negl}(k) \leq 2^{-2P(k)}$. Now by union bound:

$$\Pr[\sigma \leftarrow K'(1^k) : \exists\ (x, \pi) \text{st} V'(\sigma, x, \pi) = 1] \leq 2^{P(k)} \cdot \Pr_\sigma[\sigma \text{ bad for } x_0] \leq 2^{-P(k)}.$$

So this new NIZK is adaptively-sound. □

8.9 Trapdoor One-Way Permutation

Definition 8.7 (Trapdoor One-Way Permutation). *A trapdoor one-way permutation is a collection of one-way permutations $\{f_i : D_i \to D_i\}_{i \in I}$ where $D_i \subset \{0,1\}^{|i|}$ with five properties.*

1. *$\exists$ PPT G such that $G(1^k)$ outputs (i, t_i) where $i \in I \cap \{0,1\}^k$*
2. *It is easy to sample from D_i given i*
3. *f_i is easy to compute but hard to invert*
4. *f_i is a permutation*
5. *$\exists$ PPT A such that $A(i, y, t_i) \in f_i^{-1}(y)$*

When f_i is a one-way trapdoor permutation, it is a one-way permutation with the property that it is easy to compute f_i^{-1} only if given access to trapdoor information t_i. The function G is PPT and computes this trapdoor information. The function A is PPT and inverts f_i using this trapdoor information.

8.9.1 RSA

RSA is the only known example of a trapdoor one-way permutation. It relies on the assumption that factoring numbers is hard, but testing primality is easy. (It is known that testing primality can be done deterministically in polynomial time. It is believed that factoring can not be done in polynomial time, however this has not been proven. The best factoring algorithms are sub-exponential though.)

Definition 8.8. *RSA defines the functions (G, F, A) as follows.*

$$\begin{aligned} G(1^k) &= ((N,e),d) \text{ where } N = pq, \text{ for primes } p,q, \\ &\qquad\qquad \gcd(e,\phi(N)) = 1 \\ &\qquad\qquad d = e^{-1} \pmod{\phi(N)} \\ F_{N,e}(x) &= x^e \pmod{N} \\ A((N,e),y,d) &= y^d \pmod{N} \end{aligned}$$

The function G randomly selects the values of (p, q, e) to satisfy the desired properties. We note that if e were not coprime to $\phi(N)$, then the function would not be a permutation.

The function ϕ is Euler's Totient, and when p, q are primes, $\phi(pq) = (p-1)(q-1)$. (That is, ϕ is the order of the multiplicative group $\mathbb{Z}_N$.)

The trapdoor piece of information is the multiplicative inverse of e modulo the order of the group. It is believed hard to compute this information given only the integer N.

It is easy to show correctness of this scheme:

$$A(i, F_i(x), t_i) = (x^e)^d = x \pmod N$$

We leave it as an exercise that RSA is semantically secure with no additional assumptions.

8.10 NIZK in the Hidden-Bit Model

The hidden-bit model is a variant of the common-reference-string NIZK, where the prover can selectively reveal only parts of the random string to the verifier. (Imagine clouds obscuring the random string in the sky from the verifier, and the prover can choose which clouds to clear.)

Definition 8.9 (NIZK in the Hidden-Bit Model). *A NIZK in the hidden-bit model for statement x (with witness ω) is efficient algorithms (K_H, P_H, V_H) such that:*

1. *$r \leftarrow K_H(1^k)$ generates the hidden random string (ℓ-bits).*
2. *$(I, \phi) \leftarrow P_H(r, x, \omega)$ generates the indices $I \subseteq [\ell]$ to reveal, and the proof ϕ.*
3. *$V_H(I, \{r_i\}_{i \in I}, x, \phi)$ accepts or rejects, given the indices I, the random string r at indices I, statement x, and proof ϕ.*

Which satisfies the completeness, soundness, and zero-knowledge properties as previously defined.

Theorem 8.3. *Given a NIZK (P_H, V_H) in the hidden-bit model, we can construct a NIZK (P, V) in the normal model using trapdoor one-way permutations.*

Proof. Let the common-reference-string σ in the normal model be of length $k\ell$ and partition it into ℓ blocks of k-bits each: $\sigma = \sigma_1 \ldots \sigma_\ell$. Let $\mathcal{F}$ be a family of 2^k trapdoor OWPs, and let $B(\cdot)$ be the corresponding hard-core bit. We may assume the soundness error of (P_H, V_H) (that is, the probability of r allowing a fake proof) is at most 2^{-2k}, by the same repetition argument as in Theorem 8.2. The protocol for the normal (P, V) is:

Prover $P(\sigma, x, \omega)$**:**

1. Sample trapdoor OWP: $(f, f^{-1}) \leftarrow \mathcal{F}(1^k)$.
2. Let $\alpha_i = f^{-1}(\sigma_i)$ for $\forall i \in [\ell]$.
3. Compute hidden-bit $r_i = B(\alpha_i)$ for $\forall i \in [\ell]$. Let $r := r_1 \cdots r_\ell$.

4. Run the HBM prover: $(I, \phi) \leftarrow P_H(r, x, \omega)$.

5. Send $(f, I, \{\alpha_i\}_{i \in I}, \phi)$ to verifier.

Verifier $V(\sigma, x, f, I, \{\alpha_i\}_{i \in I}, \phi)$**:**

1. Confirm $f \in \mathcal{F}$, and $f(\alpha_i) = \sigma_i \ \forall i \in I$.

2. Compute the revealed bits $r_i = B(\alpha_i) \ \forall i \in I$.

3. Output $V_H(I, \{r_i\}_{i \in I}, x, \phi)$.

Intuitively, σ_i hides r_i because $\sigma_i \xleftarrow{f} \alpha_i \xrightarrow{B} r_i$, so by security of the hard-core bit, the verifier cannot find $r_i = B(\alpha_i)$ from $\sigma_i = f(\alpha_i)$.

Notice that if the prover is honest, then α_i will be distributed uniformly random as well (since f^{-1} is a permutation), and r_i will be unbiased as well (since $B(\cdot)$ is a hard-core bit). So this reduces exactly to the HBM distributions, and completeness of this protocol is clear (from completeness of (P_H, V_H)).

For soundness: for a fixed $f = f_0$, the distribution of r_i is uniformly random, so by the soundness of (P_H, V_H) we have

$$\Pr_{\sigma}[P^* \text{ can cheat using } f_0] \leq 2^{-2k}$$

However, a cheating P^* may be able to cleverly pick f to influence r_i, allowing him to cheat. Since we know there are only 2^k possible choices of f (the verifier confirms f is properly sampled), we can use the union bound to prove soundness:

$$\Pr_{\sigma}[\ \exists \text{ some } f \in \mathcal{F} \text{ s.t. } P^* \text{ can cheat}] \leq 2^{-k}.$$

Note that more serious problems can occur if V does not confirm $f \in \mathcal{F}$. For example, if f is not a permutation, then $f^{-1}(\sigma_i)$ can be multi-valued, and the prover can choose to “explain" σ_i using either α_i or α_i' – which is a problem if $B(\alpha_i) \neq B(\alpha_i')$.

To prove zero-knowledge, we construct a sequence of prover-hybrids. Differences from the previous hybrid are in red:

H_0 (normal model)

1. $\sigma_1 \ldots \sigma_\ell = \sigma \xleftarrow{\$} \{0,1\}^{k\ell}$

2. $(f, f^{-1}) \leftarrow \mathcal{F}$

3. $\alpha_i = f^{-1}(\sigma_i) \ \forall i \in [\ell]$

4. $r_i = B(\alpha_i) \ \forall i \in [\ell]$

5. $(I, \phi) \leftarrow P_H(r, x, \omega)$

6. Output $(\sigma, f, I, \{\alpha_i\}_{i \in I}, \phi)$

H_1

1. $(f, f^{-1}) \leftarrow \mathcal{F}$
2. $\alpha_i \overset{\$}{\leftarrow} \{0,1\}^k \ \forall i \in [\ell]$
3. $\sigma_i = f(\alpha_i) \ \forall i \in [\ell]$
4. $r_i = B(\alpha_i) \ \forall i \in [\ell]$
5. $(I, \phi) \leftarrow P_H(r, x, \omega)$
6. Output $(\sigma, f, I, \{\alpha_i\}_{i \in I}, \phi)$

In H_1, we sample α_i uniformly at random and then generate σ_i (instead of sampling σ_i and then generating α_i). This induces an exactly identical distribution, since f is a permutation.

H_2

1. $(f, f^{-1}) \leftarrow \mathcal{F}$
2. $r_i \overset{\$}{\leftarrow} \{0,1\} \ \forall i \in [\ell]$
3. $\alpha_i \overset{\$}{\leftarrow} B^{-1}(r_i) \ \forall i \in [\ell]$
4. $\sigma_i = f(\alpha_i) \ \forall i \in [\ell]$
5. $(I, \phi) \leftarrow P_H(r, x, \omega)$
6. Output $(\sigma, f, I, \{\alpha_i\}_{i \in I}, \phi)$

In H_2, we again switch the sampling order: first sample the (unbiased) bit r_i, then sample α_i from the pre-image of r_i (which can be done efficiently by simply trying random α_i's until $B(\alpha_i) = r_i$). This distribution is exactly identical to H_1. (The sampling order can be thought of as factoring the joint distribution: $\Pr(\alpha_i, r_i) = \Pr(r_i)\Pr(\alpha_i | r_i)$)

H_3

1. $(f, f^{-1}) \leftarrow \mathcal{F}$
2. $r_i \overset{\$}{\leftarrow} \{0,1\} \ \forall i \in [\ell]$
3. $\alpha_i \overset{\$}{\leftarrow} B^{-1}(r_i) \ \forall i \in [\ell]$
4. $\sigma_i = f(\alpha_i) \ \forall i \in I$
5. $\sigma_i \overset{\$}{\leftarrow} \{0,1\}^k \ \forall i \notin I$
6. $(I, \phi) \leftarrow P_H(r, x, \omega)$

7. Output $(\sigma, f, I, \{\alpha_i\}_{i\in I}, \phi)$

In H_3, we only generate σ_i honestly for $i \in I$, and output random σ_i for $i \notin I$. To argue that this is computational indistinguishable from H_2, first notice that for a fixed (known) bit r,

$$\{f(B^{-1}(r)\}\text{IND}\{f(B^{-1}(\bar{r})\} \tag{8.1}$$

where the randomness is over sampling the pre-image B^{-1}. Distinguishing the above distributions is by definition equivalent to guessing the hard-core bit, so they are indistinguishable. Given the above, we can further argue that

$$\{f(B^{-1}(r)\}\text{IND}\mathcal{U}_k \tag{8.2}$$

where $\mathcal{U}_k$ is uniform over $\{0,1\}^k$. To see this, notice that $\mathcal{U}_k$ can be equivalently generated by first sampling a random bit b, then outputting $f(B^{-1}(b))$, since f is a permutation. Therefore, any distinguisher for (8.2) can also be used to distinguish (8.1) with at least as much distinguishing-advantage (in fact, twice as much). Finally, (8.2) justifies swapping $\sigma_i = f(\alpha_i) = f(B^{-1}(r_i))$ with random for $i \notin I$ in hybrid H_3.

H_4

1. $(f, f^{-1}) \leftarrow \mathcal{F}$
2. $(I, \{r_i\}_{i\in I}, \phi) \leftarrow S_H(1^k, x)$
3. $\alpha_i \overset{\$}{\leftarrow} B^{-1}(r_i)\ \forall i \in I$
4. $\sigma_i = f(\alpha_i)\ \forall i \in I$
5. $\sigma_i \overset{\$}{\leftarrow} \{0,1\}^k\ \forall i \notin I$
6. Output $(\sigma, f, I, \{\alpha_i\}_{i\in I}, \phi)$

Finally, H_4 simply swaps the hidden-bit prover P_H for the hidden-bit simulator S_H, which is indistinguishable by the zero-knowledge property of (P_H, V_H). So (P, V) is a NIZK system in the normal model. □

□

8.11 NIZK in the Hidden-Bit Model for Graph Hamiltonian

Definition 8.10. *A Hamiltonian cycle in a graph is a cycle that visits each vertex exactly once. A Hamiltonian graph is a graph that contains a Hamiltonian cycle. More precisely, given a graph $G = (V, E)$ with $|V| = n$, we say that G is a Hamiltonian graph if there are $x_1, \ldots, x_n \in V$ such that they are all distinct vertices, and $\forall i \in \{1, \ldots, n-1\} : (x_i, x_{i+1}) \in E$, $(x_n, x_1) \in E$.*

It is well known that the problem of determining if a graph is Hamiltonian is NP-complete. Here we will construct a NIZK proof in the hidden-bit model (HBM) that is able to prove that a graph is Hamiltonian.

First we define how graphs are represented as matrices.

Definition 8.11. *A graph $G = (V, E)$ with $|V| = n$, can be represented as an $n \times n$ adjacency matrix M_G of boolean values such that* $M_G[i, j] = \begin{cases} 1 & \text{if } (i, j) \in E, \\ 0 & \text{otherwise.} \end{cases}$

A cycle matrix *is a matrix which corresponds to a graph that contains a Hamiltonian cycle and contains no edges other than this cycle.*

A permutation matrix *is a boolean matrix such that each row and each column has exactly one entry equal to 1.*

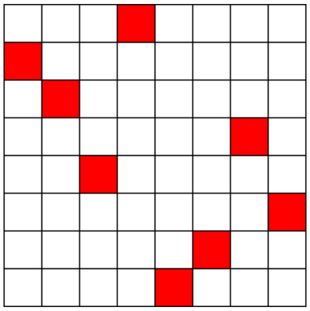


Figure 8.1: Cycle matrix.

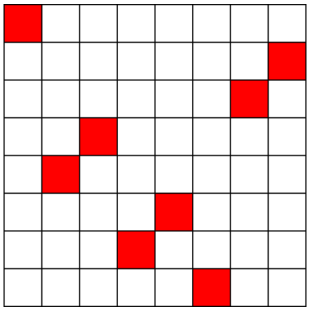


Figure 8.2: Permutation matrix.

Every cycle matrix is a permutation matrix, but the converse is not true. For each size n, there are $n!$ different permutation matrices but only $(n-1)!$ cycle matrices.

In Figure 8.1, one can see the cycle matrix as a cycle $(1, 4, 7, 6, 8, 5, 3, 2)$ on the set $\{1, 2, 3, 4, 5, 6, 7, 8\}$. In Figure 8.2, it is possible to interpret the matrix as a permutation $(1)(2, 8, 6, 5)(3, 7, 4)$ on the same set.

Theorem 8.4. *There is a non-interactive zero-knowledge (NIZK) proof in the hidden-bit model (HBM) for the problem of proving that a graph is Hamiltonian.*

Proof. In the hidden-bit model (HBM), there is a random string r with ℓ bits that the prover can read. The prover should be able to produce a proof ϕ and choose a set $I \subseteq \{1, 2, \ldots, \ell\}$ such that the proof and the bits of the string corresponding to the set I will be revealed to the verifier.

Let the graph be $G = (V, E)$ with $|V| = n$. The objective is to convince the verifier that the assertion is correct (the graph G is Hamiltonian).

Suppose that the random string r comes from a distribution such that this string represents the entries from an $n \times n$ cycle matrix M_c. Then a proof can be produced as follows.

Since the prover P knows the Hamiltonian cycle $x_1, \ldots, x_n$ in G, he can find a function $\phi : V \to \{1, 2, \ldots, n\}$ that puts the Hamiltonian cycle exactly over the cycle of M_c. More precisely, for this function we have $M_c[\phi(x_i), \phi(x_{i+1})] = 1$ for each edge (x_i, x_{i+1}) in the Hamiltonian cycle of G (we view indices modulo n). This means that all the edges of M_c will be covered by edges of G. Conversely, all the non-edges of G must be taken to non-edges of M_c.

So the strategy for the prover is to reveal the mapping ϕ and also reveal entries of M_c corresponding to $\phi(e)$ where $e \notin E$. More precisely, for the set $I = \{(\phi(u), \phi(v)) \mid (u, v) \notin E\}$, P reveals $M_c[\phi(u), \phi(v)] = 0$, which proves that $(\phi(u), \phi(v))$ is a non-edge of M_c.

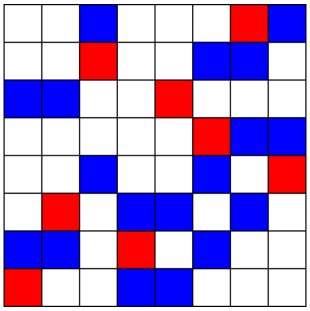


Figure 8.3: Graph matrix that includes a Hamiltonian cycle. Edges are blue/red and the cycle is red. White cells are non-edges.

A visual example is shown in Figure 8.3. The cycle graph M_c given by the random string corresponds to the red cells. These cells have value 1 in the matrix M_c and all other cells have value 0. The prover P provides a bijection ϕ that maps the edges of G to this matrix in such a way that all red cells are covered and some others may also be covered (blue cells). The important property guaranteed is that all the non-edges of G are mapped to cells that have a value 0 in the matrix (white cells).

This proof satisfies the three properties required for a zero knowledge proof.

Completeness: if P and V are both honest, then P will be able to convince V that the statement is true. That's because P knows the Hamiltonian cycle of G, hence he is always able to produce the mapping ϕ.

Soundness: if P is lying and trying to prove a false statement, then he will get caught with probability 1. If P does not know any Hamiltonian cycle in G, then any function ϕ he chooses will not cover all the edges in M_c. Hence there will be an entry in the matrix M_c which is one and will be revealed as a non-edge of G.

Zero Knowledge: V cannot get any information besides the fact that P knows a Hamiltonian cycle in G. A simulator S for this proof can be simply a machine that generates a random permutation $\phi : V \to \{1, 2, \ldots, n\}$ and reveals zeros for all the non-edges of $\phi(G)$.

In this proof we assumed that the random string r comes from a very specific distribution that corresponds to cycle matrices. Now we need to show that the general problem (where r comes from a random uniform distribution of ℓ bits) can be reduced into this previous scenario.

We proceed as follows. Let the length of the random string be $\ell = \lceil 3 \cdot \log_2 n \rceil \cdot n^4$. We view the random string r as n^4 blocks of $\lceil 3 \cdot \log_2 n \rceil$ bits and we generate a random string r' of length n^4 such that each bit in r' is 1 if and only if all the bits in the corresponding block of r are equal to 1. This way, the probability that the i-th bit of r' equals 1 is $\Pr[r'_i = 1] \approx \frac{1}{n^3}$ for every i.

Then we create an $n^2 \times n^2$ matrix M whose entries are given by the bits of r'. Let x be the number of one entries in the matrix M. The expected value for x is $\frac{n^4}{n^3} = n$. And the probability that x is exactly n is noticeable. To prove that, we can use Chebyshev's inequality:

$$\Pr[|x - n| \geq n] \leq \frac{\sigma^2}{n^2} = \frac{n^4 \cdot \frac{1}{n^3} \cdot \left(1 - \frac{1}{n^3}\right)}{n^2} < \frac{1}{n}.$$

So we have $\Pr[1 \leq x \leq 2n - 1] > \frac{n-1}{n}$. And the probability $\Pr[x = k]$ is maximal for $k = n$, so we conclude that $\Pr[x = n] > \frac{n-1}{n(2n-1)} > \frac{1}{3n}$.

Now suppose that this event ($x = n$) occurred and we have exactly n entries equal to 1 in matrix M. What is the probability that those n entries are all in different rows and are all in different columns?

We can think about the problem this way: after k one entries have been added to the matrix, the probability that a new entry will be in a different row and different column is given by $\left(1 - \frac{k}{n^2}\right)^2$. Multiplying all these values we get

$$\begin{aligned}\Pr[\text{no collision}] &\geq \left(1-\frac{1}{n^2}\right)^2 \cdot \left(1-\frac{2}{n^2}\right)^2 \cdots \left(1-\frac{n-1}{n^2}\right)^2 \\ &> 1-2\left(\frac{1}{n^2}+\frac{2}{n^2}+\cdots+\frac{n-1}{n^2}\right) = 1-\frac{n-1}{n} = \frac{1}{n}.\end{aligned}$$

Now assume that this event happened: the matrix M has exactly n entries equal to 1 and they are all in different rows and different columns. Then we can define a new $n \times n$ matrix M_c by selecting only those n rows and n columns of M. By construction, M_c is a permutation matrix. The probability that M_c is a cycle matrix is $\frac{(n-1)!}{n!} = \frac{1}{n}$. An example is shown in Figures 8.4 and 8.5.

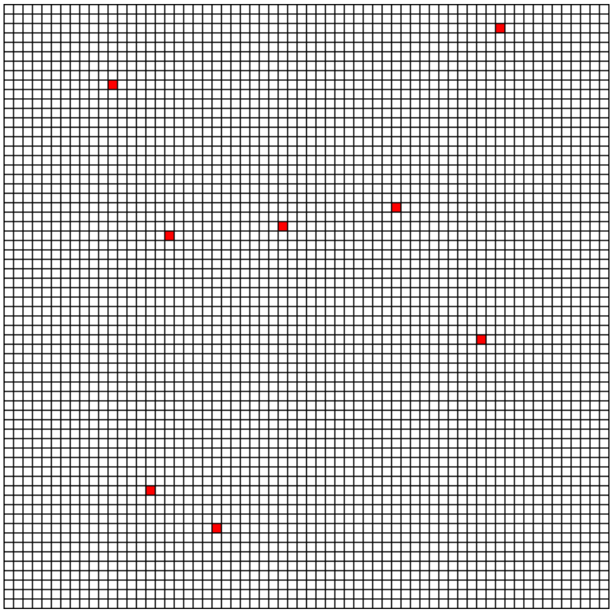


Figure 8.4: Matrix M which is $n^2 \times n^2$ for $n = 8$.

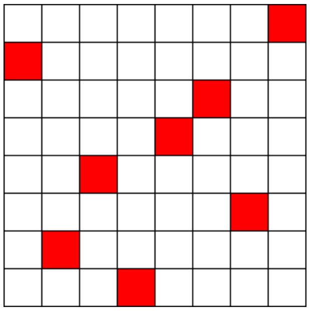


Figure 8.5: Matrix M_c which is $n \times n$ for $n = 8$. The construction worked, because M_c is a cycle matrix.

Now let's join all those probabilities. The probability that M_c is a cycle matrix is at least

$$\frac{1}{3n} \cdot \frac{1}{n} \cdot \frac{1}{n} > \frac{1}{3n^3}.$$

If we repeat this process n^4 times, then the probability that M_c is a cycle matrix in at least one iteration is at least

$$1 - \left(1 - \frac{1}{3n^3}\right)^{n^4} \approx 1 - e^{-\frac{n}{3}} = 1 - \mathsf{negl}(n).$$

The proof system works as follows. Given a random string r, the prover P tries to execute the construction above to obtain a cycle matrix. If the construction fails, the prover simply reveals all the bits in the string r to the verifier, who checks that the constructions indeed fails. If the construction succeeds, the prover reveals all the entries in the random string r that correspond to values in the matrix M which are not used in matrix M_c. The verifier will check that all these values for matrix M are indeed 0.

Then the prover proceeds as in the previous scenario using matrix M_c: he reveals the transformation ϕ and opens all the non-edges.

This process is repeated n^4 times. Or, equivalently, a big string of length $\lceil 3 \cdot \log_2 n \rceil \cdot n^4 \cdot n^4$ is used and they are all executed together. This produces a zero knowledge proof.

Completeness: if P knows the Hamiltonian cycle of G, then he will be able to find a suitable transformation ϕ whenever a cycle graph is generated by the construction.

Soundness: if P is lying and trying to prove a false statement, then he will get caught with very high probability. If any of the n^4 iterations produces a cycle graph, then P will be caught. So the probability that he will be caught is $1 - e^{-\frac{n}{3}} = 1 - \mathsf{negl}(n)$.

Zero Knowledge: again V cannot get any information if the construction succeeds. And if the construction doesn't succeed, all V gets is the random string r, which also doesn't give any information. □

Theorem 8.5. *For any language L in NP, there is a non-interactive zero-knowledge (NIZK) proof in the hidden-bit model (HBM) for the language L.*

Proof. The language L^* of Hamiltonian graphs is NP-complete. So any problem in L can be reduced to a problem in L^*. More precisely, there is a polynomial-time function f such that

$$x \in L \Longleftrightarrow f(x) \in L^*.$$

So given an input x, the prover can simply calculate $f(x)$ and produce a NIZK proof in the hidden-bit model for the fact that $f(x) \in L^*$. Then the verifier just needs to calculate $f(x)$ and check if the proof for the fact $f(x) \in L^*$ is correct. □

Theorem 8.6. *For any language L in NP, there is a non-interactive zero-knowledge (NIZK) proof in the common reference string (CRS) model for the language L.*

Proof. In Theorem 8.3 it was shown that any NIZK proof in the hidden-bit model can be converted into a NIZK proof in the standard (common reference string) model by using a trapdoor permutation. □

8.12 zkSNARKs

Exercises

Exercise 8.1 (Leaky ZK proof). *Formally define:*

1. *What it means for an interactive proof (P, V) to be* ***first-bit leaky*** *zero-knowledge, where we require that the protocol doesn't leak anything more than the first bit of the witness.*

2. *What it means for an interactive proof (P, V) to be* ***one-bit leaky*** *zero-knowledge, where we require that the protocol doesn't leak anything more than one bit that is an arbitrary adversarial chosen function of the witness.*

Exercise 8.2 (Proving OR of two statements). *Give a statistical zero-knowledge proof system $\Pi = (P, V)$ (with efficient prover) for the following language.*

$$L = \left\{ ((G_0, G_1), (G_0', G_1')) \middle| G_0 \simeq G_1 \bigvee G_0' \simeq G_1' \right\}$$

Caution: *Make sure the verifier doesn't learn which of the two pairs of graphs is isomorphic.*

Exercise 8.3 (ZK implies WI). *Let $L \in NP$ and let (P, V) be an interactive proof system for L. We say that (P, V) is* witness indistinguishable (WI) *if for all PPT V^*, for all $x \in L$, distinct witnesses $w_1, w_2 \in R_L(x)$ and auxiliary input $z \in \{0,1\}^*$, the following two views are computationally indistinguishable:*

$$View_{V^*}\left(P(x, w_1) \leftrightarrow V^*(x, z)\right) \simeq_c View_{V^*}\left(P(x, w_2) \leftrightarrow V^*(x, z)\right).$$

1. *Show that if (P, V) is an efficient prover zero-knowledge proof system, then it is also witness indistinguishable.*

2. *Assume (P, V) is an efficient prover zero-knowledge proof system. We have seen in the exercise that (P, V) is also witness indistinguishable. Define $(\tilde{P}, \tilde{V})$ to repeat (P, V) independently for k times* in parallel *(k is a polynomial), and $\tilde{V}$ accepts if and only if V accepts in all the executions. Prove that $(\tilde{P}, \tilde{V})$ is still witness indistinguishable.*

Exercise 8.4. ***Multi-statement NIZK.*** *The NIZK proof system we constructed in class required a fresh common random string (CRS) for each* *statement proved. In various settings we would like to reuse the same random string to prove multiple theorem statements while still preserving the zero-knowledge property.*

A multi-statement NIZK proof system (K, P, V) for a language L with corresponding relation R is a NIZK proof system for L with a stronger zero-knowledge property, defined as follows: $\exists$ a PPT machine $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that $\forall$ PPT machines A_1 and A_2 we have that:

$$\left| \Pr\begin{bmatrix} \sigma \leftarrow K(1^\kappa), \\ (\{x_i, w_i\}_{i\in[q]}, \textsf{state}) \leftarrow A_1(\sigma), \\ \textit{such that } \forall i \in [q], (x_i, w_i) \in R \\ \forall i \in [q], \pi_i \leftarrow P(\sigma, x_i, w_i); \\ A_2(\textsf{state}, \{\pi_i\}_{i\in[q]}) = 1 \end{bmatrix} - \Pr\begin{bmatrix} (\sigma, \tau) \leftarrow \mathcal{S}_1(1^\kappa), \\ (\{x_i, w_i\}_{i\in[q]}, \textsf{state}) \leftarrow A_1(\sigma), \\ \textit{such that } \forall i \in [q], (x_i, w_i) \in R \\ \forall i \in [q], \pi_i \leftarrow \mathcal{S}_2(\sigma, x_i, \tau); \\ A_2(\textsf{state}, \{\pi_i\}_{i\in[q]}) = 1 \end{bmatrix} \right| \leq \textsf{negl}(\kappa).$$

Assuming that a single statement NIZK proof system (K, P, V) for NP exists, construct a multi-statement NIZK proof system (K', P', V') for NP.

Hint: *Let $g : \{0,1\}^\kappa \rightarrow \{0,1\}^{2\kappa}$ be a length doubling PRG. Let K' output the output of K along with y, a random 2κ bit string. To prove $x \in L$ the prover P' proves that $\exists(w, s)$ such that either $(x, w) \in R$ or $y = g(s)$.*

9
Secure Computation

9.1 Introduction

Secure multiparty computation considers the problem of different parties computing a joint function of their separate, private inputs without revealing any extra information about these inputs than that is leaked by just the result of the computation. This setting is well motivated, and captures many different applications. Considering some of these applications will provide intuition about how security should be defined for secure computation:

Voting: Electronic voting can be thought of as a multi party computation between n players: the voters. Their input is their choice $b \in \{0, 1\}$ (we restrict ourselves to the binary choice setting without loss of generality), and the function they wish to compute is the majority function.

Now consider what happens when only one user votes: their input is trivially revealed as the output of the computation. What does privacy of inputs mean in this scenario?

Searchable Encryption: Searchable encryption schemes allow clients to store their data with a server, and subsequently grant servers tokens to conduct specific searches. However, most schemes do not consider access pattern leakage. This leakage tells the server potentially valuable information about the underlying plaintext. How do we model all the different kinds information that is leaked?

From these examples we see that defining security is tricky, with lots of potential edge cases to consider. We want to ensure that no party can learn anything more from the secure computation protocol than it can from just its input and the result of the computation. To formalize this, we adopt the **real/ideal paradigm**.

9.2 Real/Ideal Paradigm

Notation. Suppose there are n parties, and party P_i has access to some data x_i. They are trying to compute some function of their inputs $f(x_1, \ldots, x_n)$. The goal is to do this securely: even if some parties are corrupted, no one should learn more than is strictly necessitated by the computation.

Real World. In the real world, the n parties execute a protocol Π to compute the function f. This protocol can involve multiple rounds of interaction. The real world adversary $\mathcal{A}$ can corrupt arbitrarily many (but not all) parties.

Ideal World. In the ideal world, an angel helps in the computation of f: each party sends their input to the angel and receives the output of the computation $f(x_1, \ldots, x_n)$. Here the ideal world adversary $\mathcal{S}$ can again corrupt arbitrarily many (but not all) parties.

To model malicious adversaries, we need to modify the ideal world model as follows. Some parties are honest, and each honest party P_i simply sends x_i to the angel. The other parties are corrupted and are under control of the adversary $\mathcal{S}$. The adversary chooses an input x_i' for each corrupted party P_i (where possibly $x_i' \neq x_i$) and that party then sends x_i' to the angel. The angel computes a function f of the values she receives (for example, if only party 1 is honest, then the angel computes $f(x_1, x_2', x_3', \ldots, x_n')$) in order to obtain a tuple $(y_1, \ldots, y_n)$. She then sends y_i of corrupted parties to the adversary, who gets to decide whether or not honest parties will receive their response from the angel. The angel obliges. Each honest party P_i then outputs y_i if they receive y_i from the angel and $\perp$ otherwise, and corrupted parties output whatever the adversary tells them to.

Definition of Security. A protocol Π is secure against computationally bounded adversaries if for every PPT adversary $\mathcal{A}$ in the real world, there exists an PPT adversary $\mathcal{S}$ in the ideal world such that for all tuples of bit strings $(x_1, \ldots, x_n)$, we have

$$\text{Real}_{\Pi,\mathcal{A}}(x_1, \ldots x_n) \overset{c}{\simeq} \text{Ideal}_{F,\mathcal{S}}(x_1, \ldots, x_n)$$

where the left-hand side denotes the output distribution induced by Π running with $\mathcal{A}$, and the right-hand side denotes the output distribution induced by running the ideal protocol F with $\mathcal{S}$. The ideal protocol is either the original one described for semi-honest adversaries, or the modified one described for malicious adversaries.

Assumptions. We have brushed over some details of the above setting. Below we state these assumptions explicitly:

1. **Communication channel:** We assume that the communication channel between the involved parties is completely insecure, i.e., it does not preserve the privacy of the messages. However, we assume that it is reliable, which means that the adversary can drop messages, but if a message is delivered, then the receiver knows the origin.

2. **Corruption model:** We have different models of how and when the adversary can corrupt parties involved in the protocol:

    - *Static:* The adversary chooses which parties to corrupt before the protocol execution starts, and during the protocol, the malicious parties remain fixed.
    - *Adaptive:* The adversary can corrupt parties dynamically during the protocol execution, but the simulator can do the same.
    - *Mobile:* Parties corrupted by the adversary can be "uncorrupted" at any time during the protocol execution at the adversary's discretion.

3. **Fairness:** The protocols we consider are not "fair", i.e., the adversary can cause corrupted parties to abort arbitrarily. This can mean that one party does not get its share of the output of the computation.

4. **Bounds on corruption:** In some scenarios, we place upper bounds on the number of parties that the adversary can corrupt.

5. **Power of the adversary:** We consider primarily two types of adversaries:

    - *Semi-honest adversaries:* Corrupted parties follow the protocol execution Π honestly, but attempt to learn as much information as they can from the protocol transcript.
    - *Malicious adversaries:* Corrupted parties can deviate arbitrarily from the protocol Π.

6. **Standalone vs. Multiple execution:** In some settings, protocols can be executed in isolation; only one instance of a particular protocol is ever executed at any given time. In other settings, many different protocols can be executed concurrently. This can compromise security.

9.3 Oblivious transfer

Rabin's oblivious transfer sets out to accomplish the following special task of two-party secure computation. The sender has a bit $s \in \{0,1\}$. She places the bit in a box. Then the box reveals the bit to the receiver with probability 1/2, and reveals $\perp$ to the receiver with probability 1/2. The sender cannot know whether the receiver received s or $\perp$, and the receiver cannot have any information about s if they receive $\perp$.

9.3.1 1-out-of-2 oblivious transfer

1-out-of-2 oblivious transfer sets out to accomplish the following related task. The sender has two bits $s_0, s_1 \in \{0,1\}$ and the receiver has a bit $c \in \{0,1\}$. The sender places the pair (s_0, s_1) into a box, and the receiver places c into the same box. The box then reveals s_c to the receiver, and reveals $\perp$ to the sender (in order to inform the sender that the receiver has placed his bit c into the box and has been shown s_c). The sender cannot know which of her bits the receiver received, and the receiver cannot know anything about s_{1-c}.

Lemma 9.1. *A system implementing 1-out-of-2 oblivious transfer can be used to implement Rabin's oblivious transfer.*

Proof. The sender has a bit s. She randomly samples a bit $b \in \{0,1\}$ and $r \in \{0,1\}$, and the receiver randomly samples a bit $c \in \{0,1\}$. If $b = 0$, the sender defines $s_0 = s$ and $s_1 = r$, and otherwise, if $b = 1$, she defines $s_0 = r$ and $s_1 = s$. She then places the pair (s_0, s_1) into the 1-out-of-2 oblivious transfer box. The receiver places his bit c into the same box, and then the box reveals s_c to him and $\perp$ to the sender. Notice that if $b = c$, then $s_c = s$, and otherwise $s_c = r$. Once $\perp$ is revealed to the sender, she sends b to the receiver. The recieiver checks whether or not $b = c$. If $b = c$, then he knows that the bit revealed to him was s. Otherwise, he knows that the bit revealed to him was the nonsense bit r and he regards it as $\perp$.

It is easy to see that this procedure satisfies the security requirements of Rabin's oblivious transfer protocol. Indeed, as we saw above, $s_c = s$ if and only if $b = c$, and since the sender knows b, we see that knowledge of whether or not the bit s_c received by the receiver is equal to s is equivalent to knowledge of c, and the security requirements of 1-out-of-2 oblivious transfer prevent the sender from knowing c. Also, if the receiver receives r (or, equivalently, $\perp$), then knowledge of s is knowledge of the bit that was not revealed to him by the box, which is again prevented by the security requirements of 1-out-of-2 oblivious transfer. □

Lemma 9.2. *A system implementing Rabin's oblivious transfer can be used to implement 1-out-of-2 oblivious transfer.*

Proof sketch. The sender has two bits $s_0, s_1 \in \{0,1\}$ and the receiver has a single bit c. The sender randomly samples $3n$ random bits $x_1, \ldots, x_{3n} \in \{0,1\}$. Each bit is placed into its own a Rabin oblivious transfer box. The ith box then reveals either x_i or else $\perp$ to the receiver. Let

$$S := \{i \in \{1, \ldots, 3n\} : \text{the receiver knows } x_i\}.$$

The receiver picks two sets $I_0, I_1 \subseteq \{1, \ldots, 3n\}$ such that $\#I_0 = \#I_1 = n$, $I_c \subseteq S$ and $I_{1-c} \subseteq \{1, \ldots, 3n\} \setminus S$. This is possible except with probability negligible in n. He then sends the pair (I_0, I_1) to the sender. The sender then computes $t_j = \left(\bigoplus_{i \in I_j} x_i\right) \oplus s_j$ for both $j \in \{0,1\}$ and sends (t_0, t_1) to the receiver.

Notice that the receiver can uncover s_c from t_c since he knows x_i for all $i \in I_c$, but cannot uncover s_{1-c}. One can show that the security requirement of Rabin's oblivious transfer implies that this system satisfies the security requirement necessary for 1-out-of-2 oblivious transfer. □

We will see below that length-preserving one-way trapdoor permutations can be used to realize 1-out-of-2 oblivious transfer.

Theorem 9.1. *The following protocol realizes 1-out-of-2 oblivious transfer in the presence of computationally bounded and semi-honest adversaries.*

1. *The sender, who has two bits s_0 and s_1, samples a random length-preserving one-way trapdoor permutation (f, f^{-1}) and sends f to the receiver. Let $b(\cdot)$ be a hard-core bit for f.*

2. *The receiver, who has a bit c, randomly samples an n-bit string $x_c \in \{0,1\}^n$ and computes $y_c = f(x_c)$. He then samples another random n-bit string $y_{1-c} \in \{0,1\}^n$, and then sends (y_0, y_1) to the sender.*

3. *The sender computes $x_0 := f^{-1}(y_0)$ and $x_1 := f^{-1}(y_1)$. She computes $b_0 := b(x_0) \oplus s_0$ and $b_1 := b(x_1) \oplus s_1$, and then sends the pair (b_0, b_1) to the receiver.*

4. *The receiver knows c and x_c, and can therefore compute $s_c = b_c \oplus b(x_c)$.*

Proof. Correctness is clear from the protocol. For security, from the sender side, since f is a length-preserving permutation, (y_0, y_1) is statistically indistinguishable from two random strings, hence she can't learn anything about c. From the receiver side, guessing s_{1-c} correctly is equivalent to guessing the hard-core bit for y_{1-c}. □

9.3.2 1-out-of-4 oblivious transfer

We describe how to implement a 1-out-of-4 OT using 1-out-of-2 OT:

1. The sender, P_1 samples 5 random values $S_i \leftarrow \{0,1\}, i \in \{1,\ldots,5\}$.
2. P_1 computes

$$\begin{aligned} \alpha_0 &= S_0 \oplus S_2 \oplus m_0 \\ \alpha_1 &= S_0 \oplus S_3 \oplus m_1 \\ \alpha_2 &= S_1 \oplus S_4 \oplus m_2 \\ \alpha_3 &= S_1 \oplus S_5 \oplus m_3 \end{aligned}$$

   It sends these values to P_2.
3. The parties engage in 3 1-out-of-2 Oblivious Transfer protocols for the following messages: (S_0, S_1), (S_2, S_3), (S_4, S_5). THe receiver's input for the first OT is the first choice bit, and for the second and third ones is the second choice bit.
4. The receiver can only decrypt one ciphertext.

9.4 Yao's Garbled Circuit

9.5 Setup

Yao's Garbled Circuits is presented as a solution to Yao's Millionaires' problem, which asks whether two millionaires can compete for bragging rights of which is richer without revealing their wealth to each other. It started the area of secure computation. We will present a solution for the two party problem; it can be extended to a polynomial number of parties, using the techniques from last lecture.

The solution we saw previously needed an interaction for each AND gate. Yao's solution requires only one message, so it provides a constant size of interaction. We present a solution only for semi honest security. This can be amplified to malicious security, but there are more efficient ways of amplifying this than what we saw last lecture.

9.5.1 Secure Computation

Recall our definition of secure computation. We define ideal and real worlds. Security is defined to hold if anything an attacker can achieve in the real world can also be achieved by an ideal attacker in the ideal world. We define the ideal world to have the properties that we desire. For security to hold these properties must also hold in the real world.

9.5.2 $(\mathsf{Garble}, \mathsf{Eval})$

We will provide a definition, similar to how we define encryption, that allows us avoid dealing with simulators all the time.

Yao's Garbled Circuit is defined as two efficient algorithms $(\mathsf{Garble}, \mathsf{Eval})$. Let the circuit C have n input wires. Garble produces the garbled circuit and two labels for each input wire. The labels are for each of 0 and 1 on that wire and are like encryption keys.

$$(\tilde{C}, \{\ell_{i,b}\}_{i\in[n],b\in\{0,1\}}) \leftarrow \mathsf{Garble}(1^k, C)$$

To evaluate the circuit on a single input we must choose a value for each of the n input wires. Given n of 2n input keys, Eval can evaluate the circuit on those keys and get the circuit result.

$$C(x) \leftarrow \mathsf{Eval}(\tilde{C}, \{\ell_{i,x_i}\}_{i\in[n]})$$

Correctness Correctness is as usual, if you garble honestly, evaluation should produce the correct result.

$$\forall C, x Pr[C(x) = \mathsf{Eval}(\tilde{C}, \{l_{i,x_i}\}), (\tilde{C}, \{\ell_{i,b}\}) = \mathsf{Garble}(1^k, C)] = 1 - neg(k)$$

Security For security we require that a party receiving a garbled circuit and n inputs labels can not computationally distinguish the joint distribution of the circuits and labels from the distribution produced by a simulator with access to the circuit and its evaluation on the input that the labels represent. The simulator does not have access to the actual inputs. If this holds, the party receiving the garbled circuit and n labels can not determine the inputs.

$$\begin{aligned}&\exists \mathsf{Sim} : \forall C, x\\&(\tilde{C}, \{\ell_{i,x_i}\}_{i\in[n]}) \simeq \mathsf{Sim}(1^k, C, C(x)) \text{ where}\\&(\tilde{C}, \{\ell_{i,b}\}_{i\in[n],b\in\{0,1\}}) \leftarrow \mathsf{Garble}(1^k, C)\end{aligned}$$

For simplicity we pass the circuit to the simulator. You could also use universal circuits and pass in with the inputs the specific circuit that the universal circuit should realize.

9.6 *Use for Semi-honest two party secure communication*

Alice, with input x^1, and Bob, with input x^2, have a circuit, C, that they want to evaluate securely. The size of their combined inputs is n, so $|x^1| = n_1, |x^2| = n - n_1, |x^1| + |x^2| = n$. They can do this by Alice garbling a circuit and sending input wire labels to Bob, as in Figure 9.1.

Alice garbles the circuit and passes it to Bob, $\tilde{C}$. Alice passes the labels for her input directly to Bob, $\{\ell_{i,x_i^1}\}_{i\in[n]/[n_2]}$. Alice passes all the labels for Bob's input wires into oblivious transfer, $\{\ell_{i,b_i}\}_{i\in[n]/[n_1],b\in\{0,1\}}$, from which Bob can retrieve the labels for his actual inputs, $\{\ell_{i,x_i^2}\}_{i\in[n]/[n_1]}$. Bob now has the garbled circuit and one label for each input wire. He evaluates the garbled circuit on those garbled inputs and learns $C(x^1||x^2)$. Bob does not learn anything besides the result as he has only the garbled circuit and n garbled inputs. Alice does not learn anything as she uses oblivious transfer to give Bob his input labels and receives nothing in reply.

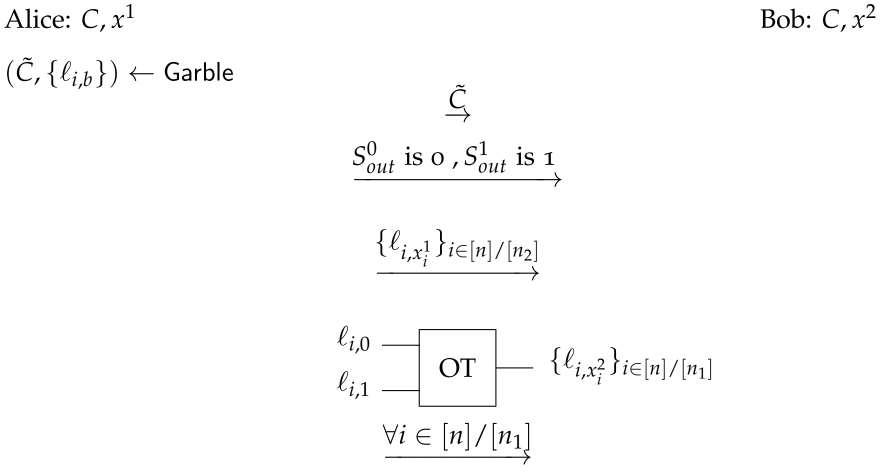

Alice: C, x^1
Bob: C, x^2
$(\tilde{C}, \{\ell_{i,b}\}) \leftarrow$ Garble
$\xrightarrow{\tilde{C}}$
$\xrightarrow{S^0_{out} \text{ is } 0,\ S^1_{out} \text{ is } 1}$
$\xrightarrow{\{\ell_{i,x_i^1}\}_{i\in[n]/[n_2]}}$
$\ell_{i,0}$
$\ell_{i,1}$
OT
$\{\ell_{i,x_i^2}\}_{i\in[n]/[n_1]}$
$\xrightarrow{\forall i \in [n]/[n_1]}$

Figure 9.1: Messages in Yao's Garbeled Circuit

9.6.1 Construction of Garbled Circuits

We would like to garble a circuit such that there are two keys for each input wire. Correctness should be that given one of the two keys for each wire we can compute the output for the inputs those keys correspond to. Security should be that given one key for each wire you can only learn the output, not the actual inputs.

We build the circuit as a bunch of NAND gates that outputs one bit. If more bits are required, this can be done multiple times. NAND gates can create any logic needed. We define the following sets:

$$W = \text{the set of wires in the circuit}$$
$$G = \text{the set of gates in the circuit.}$$

For each wire in the circuit, sample two keys to label the possible inputs 0 and 1 to the wire

$$\forall w \in W \quad S^0_w, S^1_w \leftarrow \{0,1\}^k.$$

We can think of these as the secret keys to an encryption scheme (Gen, Enc, Dec). For such a scheme we can always replace the secret key with the random bits fed into Gen.

Wires For each wire in the circuit we will have an invariant that the evaluator can only get one of the wires two encrypted values. Consider an internal wire fed by the evaluation of a gate. The gate receives two encrypted values as inputs and produces one encrypted output. The output will be one of the two labels for that wire and the evaluator will have no way of obtaining the other label for that wire. For example on wire w_i, the evaluator will only learn the value for 1, $S^1_{w_i}$. We ensure this for the input wires by giving the evaluator only one of the two encrypted values for the wire.

Gates For every gate in the circuit we create four cipher texts. For each choice of inputs we encrypt the output key under each of the input keys. Let gate g have inputs w_1, w_2 and output w_3,

$$\begin{aligned} e_g^{00} &= \mathsf{Enc}_{S^0_{w_1}}(\mathsf{Enc}_{S^0_{w_2}}(S^1_{w_3}, 0^k)) \\ e_g^{01} &= \mathsf{Enc}_{S^0_{w_1}}(\mathsf{Enc}_{S^1_{w_2}}(S^1_{w_3}, 0^k)) \\ e_g^{10} &= \mathsf{Enc}_{S^1_{w_1}}(\mathsf{Enc}_{S^0_{w_2}}(S^1_{w_3}, 0^k)) \\ e_g^{11} &= \mathsf{Enc}_{S^1_{w_1}}(\mathsf{Enc}_{S^1_{w_2}}(S^0_{w_3}, 0^k)). \end{aligned}$$

We add k zeros at the end.

Final Output For the final output wire, S_{out}, we can just give out their values,

$$\begin{aligned} &S^0_{out} \text{ corresponds to 0} \\ &S^1_{out} \text{ corresponds to 1.} \end{aligned}$$

$\tilde{\mathbf{C}}$ For each gate, Alice sends Bob a random permutation of the set of four encrypted output values.

$$\{e_g^{C_1, C_2}\} \quad \forall g \in G \quad C_1, C_2 \in \{0, 1\}.$$

For each gate, Alice sends Bob a random permutation of the set of four encrypted output values

Evaluation With an encrypted gate g, input keys S_{w_1} S_{w_2} for the input wires, and four randomly permuted encryptions of the output keys, $e_g^a, e_g^b, e_g^c, e_g^d$, Bob can evaluate the gate to find the corresponding key S_{w3} for the output wire. Bob can decrypt each of the encrypted output keys until he finds one that decrypts to a string ending in the proper number of 0's, which is very likely to contain the proper output key. We can increase the probability of the correct key by increasing the number of 0's.

$$\exists i \in \{a, b, c, d\} : \mathsf{Dec}_{S_{w_2}}(\mathsf{Dec}_{S_{w_1}}(e^i_g)) = S_{w_3}, 0^k$$

Given input wire labels $\{\ell_{i,x_i}\}_{i \in [n]}$ the complete encrypted circuit $\tilde{C}$ is evaluated by working up from the input gates.

The evaluator should not be able to infer anything except what they could infer in the ideal world. As a simple example, if the evaluator supplies one input to a circuit of just one NAND gate, they would be able to infer the input of the other party. However, this is true is the ideal world as well.

9.7 *Proof Intuition*

What intuition can we offer that the distribution of $\tilde{C}$ with one label per input wire is indistinguishable from what which a simulator could produce with access to the output? For each input wire we are only given one key. As we are doing double encryption, for each input gate we only have the keys needed to decrypt one of the four possible outputs. The other three are protected by semantic security. So from each input gate we learn only one key compounding to its output wire. As the output labels were randomized, we also do not know if that key corresponds to a 0 or a 1. For the next level of gates we again have only one key per input wire, and our argument continues. So for each wire of the circuit we can only know one key corresponding to an output value for the wire. Everything else is random garbage. As we control the mapping from output keys to output values, we can set this to whatever is needed to match the expected output.

Security only holds for evaluation of the circuit with one set of input values and we assume that the circuit is combinatorial and thus acyclic.

9.8 *Malicious attacker intead of semi-honest attacker*

The assumption we had before consisted of a semi-honest attacker instead of a malicious attacker. A malicious attacker does not have to follow the protocol, and may instead alter the original protocol. The main idea here is that we can convert a protocol aimed at semi-honest attackers into one that will work with malicious attackers.

At the beginning of the protocol, we have each party commit to its inputs: Given a commitment protocol *com*, Party 1 produces

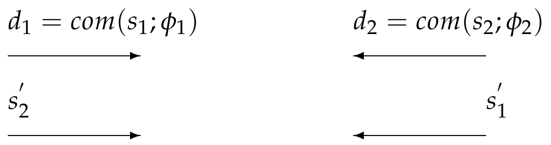
$$c_1 = com(x_1; w_1)$$
$$d_1 = com(r_1; \phi_1)$$

Party 2 produces

$$c_2 = com(x_2; w_2)$$
$$d_2 = com(r_2; \phi_2)$$

We have the following guarantee: $\exists x_i, r_i, w_i, \phi_i$ such that $c_i = com(x_i; w_i) \wedge d_i = com(r_i; \phi_i) \wedge t = \pi(i, \text{transcript}, x_i, r_i)$, where transcript is the set of messages sent in the protocol so far.

Here we have a potential problem. Since both parties are choosing their own random coins, we have to be able to enforce that the coins are *indeed* random. We can solve this by using the following protocol:



We calculate $r_1 = s_1 \oplus s_1'$, and $r_2 = s_2 \oplus s_2'$. As long as one party is picking the random coins honestly, both parties would have truly random coins.

Furthermore, during the first commitment phase, we want to make sure that the committing party actually knows the value that is being committed to. Thus, we also attach along with the commitment a zero-knowledge proof of knowledge (ZK-PoK) to prove that the committing party knows the value that is being committed to.

9.8.1 *Zero-knowledge proof of knowledge (ZK-PoK)*

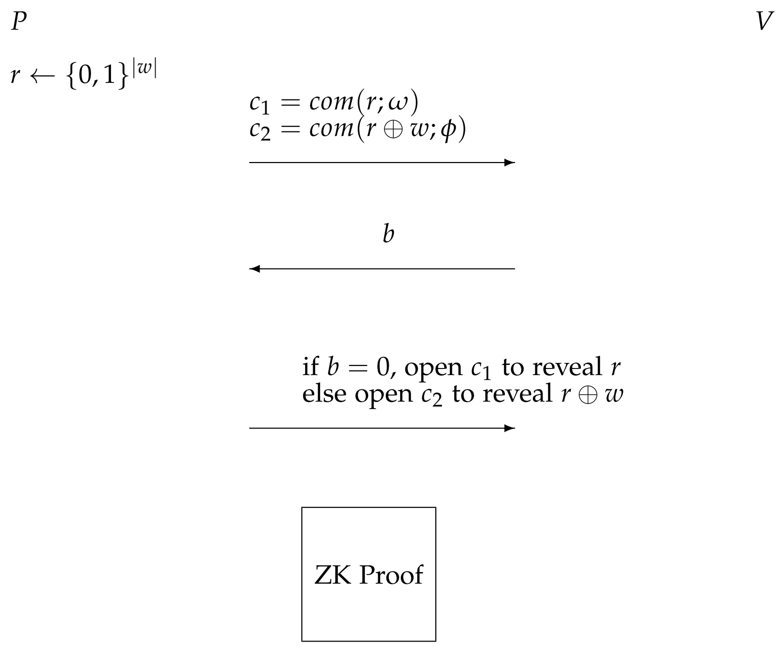
Definition 9.1 (ZK-PoK). *Zero-knowlwedge proof of knowledge (ZK-PoK) is a zero-knowledge proof system (P, V) with the property proof of knowledge with knowledge error κ:*

$\exists$ a PPT E (knowledge extractor) such that $\forall x \in L$ and $\forall P^*$ (possibly unbounded), it holds that if $\Pr[Out_V(P^*(x, w) \leftrightarrow V(x))] > \kappa(x)$, *then*

$$\Pr[E^{P^*}(x) \in R(x)] \geq \Pr[Out_V(P^* \leftrightarrow V(x))] = 1] - \kappa(x).$$

Here we have L be the language, R be the relation, and $R(x)$ is the set such that $\forall w \in R(x)$, $(x, w) \in R$.

Given a zero-knowledge proof system, we can construct a ZK-PoK system for statement $x \in L$ with witness w as follows:


P
V
$r \leftarrow \{0,1\}^{|w|}$
$c_1 = com(r;\omega)$
$c_2 = com(r \oplus w;\phi)$
b
if $b = 0$, open c_1 to reveal r
else open c_2 to reveal $r \oplus w$
ZK Proof

The last ZK proof proves that $\exists r, w, \omega, \phi$ such that $(x, w) \in R$ and $c_1 = com(r;\omega)$, $c_2 = com(r \oplus w;\phi)$.

Exercises

Exercise 9.1. *Given a (secure against malicious adversaries) two-party secure computation protocol (and nothing else) construct a (secure against malicious adversaries) three-party secure computation protocol.*

10
Obfustopia

10.1 Witness Encryption: A Story

Imagine that a billionaire who loves mathematics, would like to award with 1 million dollars the mathematician(s) who will prove the Riemann Hypothesis. Of course, neither does the billionaire know if the Riemann Hypothesis is true, nor if he will be still alive (if and) when a mathematician will come up with a proof. To overcome these couple of problems, the billionaire decides to:

1. Put 1 million dollars in gold in a big treasure chest.
2. Choose an arbitrary place of the world, dig up a hole, and hide the treasure chest.
3. Encrypt the coordinates of the treasure chest in a message so that only the mathematician(s) who can actually prove the Riemann Hypothesis can decrypt it.
4. Publish the ciphertext in every newspaper in the world.

The goal of this lecture is to help the billionaire with step 3. To do so, we will assume for simplicity that the proof is at most 10000 pages long. The latter assumption implies that the language

$$L = \{x \text{ such that } x \text{ is an acceptable Riemann Hypothesis proof}\}$$

is in NP and therefore, using a reduction, we can come up with a circuit C that takes as input x and outputs 1 if x is a proof for the Riemann Hypothesis and 0 otherwise.

Our goal now is to design a pair of PPT machines (Enc, Dec) such that:

1. $\text{Enc}(C, m)$ takes as input the circuit C and $m \in \{0, 1\}$ and outputs a ciphertext $e \in \{0, 1\}^*$.

2. $\text{Dec}(C, e, w)$ takes as input the circuit C, the cipertext e and a witness $w \in \{0,1\}^*$ and outputs m if if $C(w) = 1$ or $\perp$ otherwise.

and so that they satisfy the following correctness and security requirements:

- **Correctness:** If $\exists w$ such that $C(w) = 1$ then $\text{Dec}(C, e, w)$ outputs m.
- **Security:** If $\nexists w$ such that $C(w) = 1$ then $\text{Enc}(C, 0) \approx^c \text{Enc}(C, 1)$ (where $\approx^c$ means "computationally indistinguishable").

10.2 A Simple Language

As a first example, we show how we can design such an encryption scheme for a simple language. Let G be a group of prime order and g be a generator of the group. For elements $A, B, T \in G$ consider the language $L = \{(a, b) : A = g^a, B = g^b, T = g^{ab}\}$. An encryption scheme for that language with the correctness and security requirements of Section 10.1 is the following:

- **Encryption**(g, A, B, T, G)**:**
  - Choose elements $r_1, r_2 \in \mathbb{Z}_p^*$ uniformly and independently.
  - Let $c_1 = A^{r_1} g^{r_2}$, $c_2 = g^m T^{r_1} B^{r_2}$, where $m \in \{0,1\}$ is the message we want to encrypt.
  - Output $c = (c_1, c_2)$
- **Decryption(**b**):**
  - Output $\frac{c_2}{c_1^b}$

Correctness: The correcntess of the above encryption scheme follows from the fact that if there exist $(a, b) \in L$ then:

$$\begin{aligned} \frac{c_2}{c_1^b} &= \frac{g^m T^{r_1} B^{r_2}}{(A^{r_1} g^{r_2})^b} \\ &= \frac{g^m \left(g^{ab}\right)^{r_1} \left(g^b\right)^{r_2}}{(g^a)^{r_1 b} g^{r_2 b}} \\ &= g^m \end{aligned}$$

Since $m \in \{0,1\}$ and we know g, the value of g^m implies the value of m.

Security: As far as the security of the scheme is concerned, since L is quite simple, we can actually prove that m is information-theoretically hidden. To see this, assume there does not exist $(a, b) \in L$, but an adversary has the power to compute discrete logarithms. In that case, given c_1 and c_2 the adversary could get a system of the form:

$$\begin{aligned} ar_1 + r_2 &= s_1 \\ m + rr_1 + br_2 &= s_2 \end{aligned}$$

where s_1 and s_2 are the discrete logarithms of c_1 and c_2 respectively (with base g), and $r \neq ab$ is an element of $\mathbb{Z}_p^*$ such that $T = g^r$. Observe now that for each value of m there exist numbers r_1 and r_2 so that the above system has a solution, and thus m is indeed information-theoretically hidden (on the other hand, if we had that $ab = r$ then the equations are linearly dependent).

10.3 An NP Complete Language

In this section we focus on our original goal of designing an encryption for an NP complete language L. Specifically, we will consider the NP-complete problem *exact cover*. Besides that, we introduce the n-Multilinear Decisional Diffie-Hellman (n-MDDH) assumption and the Decisional Multilinear No-Exact-Cover Assumption. The latter will guarantee the security of our construction.

10.3.1 Exact Cover

We are given as input $x = (n, S_1, S_2, \ldots, S_l)$, where n is an integer and each $S_i, i \in [l]$ is a subset of $[n]$, and our goal is to find a subset of indices $T \subseteq [l]$ such that:

1. $\cup_{i \in T} S_i = [n]$ and
2. $\forall i, j \in T$ such that $i \neq j$ we have that $S_i \cap S_j = \emptyset$.

If such a T exists, we say that T is an exact cover of x.

10.3.2 Multilinear Maps

Mutlinear maps is a generalization of bilinear maps (which we have already seen) that will be useful in our construction. Specifically, we assume the existence of a group generator $\mathcal{G}$, which takes as input a security parameter λ and a positive integer n to indicate the number of allowed operations. $\mathcal{G}(1^\lambda, n)$ outputs a sequence of groups $\vec{\mathbb{G}} = (\mathbb{G}_1, \mathbb{G}_2, \ldots, \mathbb{G}_n)$ each of large prime order $P > 2^\lambda$. In addition, we let g_i be a canonical generator of $\mathbb{G}_i$ (and is known from the group's description).

We also assume the existence of a set of bilinear maps $\{e_{i,j} : \mathbb{G}_i \times \mathbb{G}_j \to \mathbb{G}_{i+j} \mid i, j \geq 1; i + j \leq n\}$. The map $e_{i,j}$ satisfies the following relation:

$$e_{i,j}\left(g_i^a, g_j^b\right) = g_{i+j}^{ab} : \forall a, b \in \mathbb{Z}_p \tag{10.1}$$

and we observe that one consequence of this is that $e_{i,j}(g_i, g_j) = g_{i+j}$ for each valid i, j.

10.3.3 The n-MDDH Assumption

The n-Multilinear Decisional Diffie-Hellman (n-MDDH) problem states the following: A challenger runs $\mathcal{G}(1^\lambda, n)$ to generate groups and generators of order p. Then it picks random $s, c_1, \ldots, c_n \in \mathbb{Z}_p$. The assumption then states that given $g = g_1, g^s, g^{c_1}, \ldots, g^{c_n}$ it is hard to distinguish $T = g_n^{s \prod_{j \in [1,n]} c_j}$ from a random group element in G_n, with better than negligible advantage (in security parameter λ).

10.3.4 Decisional Multilinear No-Exact-Cover Assumption

Let $x = (n, S_1, \ldots, S_l)$ be an instance of the exact cover problem that has no solution. Let param $\leftarrow \mathcal{G}(1^{1+n}, n)$ be a description of a multilinear group family with order $p = p(\lambda)$. Let $a_1, a_2, \ldots, a_n, r$ be uniformly random in $\mathbb{Z}_p$. For $i \in [l]$, let $c_i = g_{|S_i|}^{\prod_{j \in S_i} a_j}$. Distiguish between the two distributions:

$$(\text{params}, c_1, \ldots, c_l, g_n^{a_1 a_2 \ldots a_n}) \text{ and } (\text{params}, c_1, \ldots, c_l, g_n^r)$$

The Decisional Multilinear No-Exact-Cover Assumption is that for all adversaries $\mathcal{A}$, there exists a fixed negligible function $\nu(\cdot)$ such that for all instances x with no solution, $\mathcal{A}$'s distinguishing advantage against the Decisional Multilinear No-Exact-Cover Problem for x is at most $\nu(\lambda)$.

10.3.5 The Encryption Scheme

We are now ready to give the description of our encryption scheme.

- $\mathrm{Enc}(x, m)$ takes as input $x = (n, S_1, \ldots, S_l)$ and the message $m \in \{0, 1\}$ and:
  - Samples $a_0, a_1, \ldots, a_n$ uniformly and independently from $\mathbb{Z}_p^*$.
  - $\forall i \in [l]$ let $c_i = g_{|S_i|}^{\prod_{j \in S_j} a_j}$
  - Sample uniformly an element $r \in \mathbb{Z}_p^*$
  - Let $d = d(m)$ be $g_n^{\prod_{j \in [n]} a_j}$ if $m = 1$ or g_n^r if $m = 0$.
  - Output $c = (d, c_1, \ldots, c_l)$
- $\mathrm{Dec}(x, T)$, where $T \subseteq [l]$ is a set of indices, computes $\prod_{i \in T} c_i$ and outputs 1 if the latter value equals to d or 0 otherwise.
- **Correctness:** Assume that T is an exact cover of x. Then, it is not hard to see that:

$$\begin{aligned} \prod_{i \in T} c_i &= \prod_{i \in T} g_{|S_i|}^{\prod_{j \in S_j} a_j} \\ &= g_n^{\prod_{j \in [n]} a_j} \end{aligned}$$

  where we have used (10.1) repeatedly and the fact that T is an exact cover (to show that $\sum_{i \in T} |S_i| = n$ and that $\prod_{i \in T} \prod_{j \in S_i} a_j = \prod_{i \in [n]} a_i$).
- **Security:** Intuitively, the construction is secure, since the only way to make $g_n^{\prod_{j \in [n]} a_i}$ is to find an exact cover of $[n]$. As a matter of fact,

observe that if an exact cover does not exist, then for each subset of indices T'(such that $\cup_{i \in T'} S_j = [n]$) we have that

$$\sum_{i=1}^{n} |S_i| > n,$$

which means that $\prod_{i \in T} \prod_{j \in S_i} a_j$ is different than $\prod_{j \in [n]} a_j$. Formally, the security is based on the Decisional Multilinear No-Exact-Cover Assumption.

10.4 Obfuscation

The problem of program obfuscation asks whether one can transform a program (e.g., circuits, Turing machines) to another semantically equivalent program (i.e., having the same input/output behavior), but is otherwise intelligible. It was originally formalized by Barak et al. who constructed a family of circuits that are non-obfuscatable under the most natural virtual black box (VBB) security.

10.5 VBB Obfuscation

As a motivation, recall that in a private-key encryption setting, we have a secret key k, encryption E_k and decryption D_k. A natural candidate for public-key encryption would be to simply release an encryption $E'_k \equiv E_k$ (i.e. E'_k semantically equivalent to E_k, but computationally bounded adversaries would have a hard time figuring out k from E'_k.

Definition 10.1 (Obfuscator of circuits under VBB). *O is an* obfuscator *of circuits if*

1. *Correctness:* $\forall c, O(c) \equiv c$.
2. *Efficiency:* $\forall c, |O(c)| \leq \text{poly}(|c|)$.
3. *VBB:* $\forall A$, *A is PPT bounded,* $\exists$ *S (also PPT) s.t.* $\forall c$,

$$\left|\Pr\left[A\left(O(c)\right) = 1\right] - \Pr\left[S^c(1^{|c|}) = 1\right]\right| \leq \text{negl}(|c|).$$

Similarly we can define it for Turing machines.

Definition 10.2 (Obfuscator of TMs under VBB). *O is an* obfuscator *of Turing machines if*

1. *Correctness: $\forall M, O(M) \equiv M$.*
2. *Efficiency: $\exists q(\cdot) = \text{poly}(\cdot), \forall M\,(M(x)$ halts in t steps $\implies O(M)(x)$ halts in $q(t)$ steps$)$.*
3. *VBB: Let $M'(t, x)$ be a TM that runs $M(x)$ for t steps. $\forall A$, A is PPT bounded, $\exists$ Sim (also PPT) s.t. $\forall c$,*

$$\left|\Pr\left[A\left(O(M)\right) = 1\right] - \Pr\left[S^{M'}(1^{|M'|}) = 1\right]\right| \leq \text{negl}(|M'|).$$

Let's show that our candidate PKE from VBB obfuscator O is semantic secure, using a simple hybrid argument.

Proof. Recall the public key $PK = O(E_k)$. Let's assume E_k is a circuit.

$$\begin{aligned}
H_0 &: A(\{(PK, E_k(m_0))\}) \\
H_1 &: S^c(\{E_k(m_0)\}) && \text{by VBB} \\
H_2 &: S^c(\{E_k(m_1)\}) && \text{by semantic security of private key encryption} \\
H_3 &: A(\{(PK, E_k(m_1))\}) && \text{by VBB}
\end{aligned}$$

□

Unfortunately VBB obfuscator for all circuits does not exist. Now we show the impossiblity result of VBB obfuscator.

Theorem 10.1. *Let O be an obfuscator. There exists PPT bounded A, and a family (ensemble) of functions $\{H_n\}$, $\{Z_n\}$ s.t. for every PPT bounded simulator S,*

$$A\left(O(H_n)\right) = 1 \;\; \& \;\; A\left(O(Z_n)\right) = 0$$
$$\left|\Pr\left[S^{H_n}\left(1^{|H_n|}\right) = 1\right] - \Pr\left[S^{Z_n}\left(1^{|Z_n|}\right) = 1\right]\right| \leq \text{negl}(n).$$

Proof. Let $\alpha, \beta \xleftarrow{\$} \{0,1\}^n$.

We start by constructing $A', C_{\alpha,\beta}, D_{\alpha,\beta}$ s.t.

$$A'\left(O(C_{\alpha,\beta}), O(D_{\alpha,\beta})\right) = 1 \;\; \& \;\; A'\left(O(Z_n), O(D_{\alpha,\beta})\right) = 0$$
$$\left|\Pr\left[S^{C_{\alpha,\beta}, D_{\alpha,\beta}}(\mathbf{1}) = 1\right] - \Pr\left[S^{Z_n, D_{\alpha,\beta}}(\mathbf{1}) = 1\right]\right| \leq \text{negl}(n).$$

$$C_{\alpha,\beta}(x) = \begin{cases} \beta, & \text{if } x = \alpha, \\ 0^n, & \text{o/w} \end{cases}$$
$$D_{\alpha,\beta}(c) = \begin{cases} 1, & \text{if } c(\alpha) = \beta, \\ 0, & \text{o/w.} \end{cases}$$

Clearly $A'(X,Y) = Y(X)$ works. Now notice that input length to D grows as the size of $O(C)$.

However for Turing machines which can have the same description length, one could combine the two in the following way:

$$F_{\alpha,\beta}(b,x) = \begin{cases} C_{\alpha,\beta}(x), & b = 0 \\ D_{\alpha,\beta}(x), & b = 1 \end{cases}.$$

Let $OF = O(F_{\alpha,\beta})$, $OF_0(x) = OF(0,x)$, similarly for OF_1, then A would be just $A(OF) = OF_1(OF_0)$.

Now assuming OWF exists, specifically we already have priavte-key encryption, we modify D as follows.

$$D_k^{\alpha,\beta}(1,i) = \mathrm{Enc}_k(\alpha_i)$$

$$D_k^{\alpha,\beta}(2,c,d,\odot) = \mathrm{Enc}_k(\mathrm{Dec}_k(c) \odot \mathrm{Dec}_k(d)), \text{where } \odot \text{ is a gate of AND, OR, NOT}$$

$$D_k^{\alpha,\beta}(3,\gamma_1,\cdots,\gamma_n) = \begin{cases} 1, & \forall i, \mathrm{Dec}_k(\gamma_i) = \beta_i, \\ 0, & \text{o/w.} \end{cases}$$

Now the adversary A just simulate $O(C)$ gate by gate with a much smaller $O(D)$, thus we can use the combining tricks as for the Turing machines. □

10.6 Indistinguishability Obfuscation

Definition 10.3 (Indistinguishability Obfuscator). *A uniform PPT machine* iO *is an* indistinguishability obfuscator *for a collection of circuits $\mathcal{C}_\kappa$ if the following conditions hold:*

- Correctness. *For every circuit $C \in \mathcal{C}_\kappa$ and for all inputs x, $C(x) = \mathsf{iO}(C(x))$.*
- Polynomial slowdown. *For every circuit $C \in \mathcal{C}_\kappa$, $|\mathsf{iO}(C)| \leq p(|C|)$ for some polynomial p.*
- Indistinguishability. *For all pairs of circuits $C_1, C_2 \in \mathcal{C}_\kappa$, if $|C_1| = |C_2|$ and $C_1(x) = C_2(x)$ for all inputs x, then $\mathsf{iO}(C_1) \overset{c}{\simeq} \mathsf{iO}(C_2)$. More precisely, there is a negligible function $\nu(k)$ such that for any (possibly non-uniform) PPT A,*

$$\left| \Pr[A(\mathsf{iO}(C_1)) = 1] - \Pr[A(\mathsf{iO}(C_2)) = 1] \right| \leq \nu(k).$$

Lemma 10.1. *Indistinguishability obfuscation implies witness encryption.*

Proof. Recall the witness encryption scheme, with which one could encrypt a message m to an instance x of an NP language L, such that

$$\mathrm{Dec}\left(x, w, \mathrm{Enc}\left(x, m\right)\right) = \begin{cases} m, \mathrm{if}(x,w) \in L, \\ \perp, \mathrm{o/w} \end{cases}$$

Let $C_{x,m}(w)$ be a circuit that on input w, outputs m if and only if $(x, w) \in L$. Now we construct witness encryption as follows: $\text{Enc}(x, m) = \text{iO}C_{x,m}, \text{Dec}(x, w, c) = c(w)$.

Semantic security follows from the fact that, for $x \notin L$, $C_{x,m}$ is just a circuit that always output $\perp$, and by indistinguishability obfuscation, we could replace it with a constant circuit (padding if necessary), and then change the message, and change the circuit back. □

Lemma 10.2. *Indistinguishability obfuscation and OWFs imply public key encryption.*

Proof. We'll use a length doubling PRG $F : \{0,1\}^n \to \{0,1\}^{2n}$, together with a witness encryption scheme (E, D). The NP language for the encryption scheme would be the image of F.

$$\text{Gen}(1^n) = (PK = F(s), SK = s), s \overset{\$}{\leftarrow} \{0,1\}^n$$
$$\text{Enc}(PK, m) = E(x = PK, m)$$
$$\text{Dec}(e, SK = s) = D(x = PK, w = s, c = e).$$

□

Lemma 10.3. *Every best possible obfuscator could be equivalently achieved with an indistinguishability obfuscation (up to padding and computationally bounded).*

Proof. Consider circuit c, the *best possible obfuscated* $BPO(c)$, and c' which is just padding c to the same size of $BPO(c)$. Computationally bounded adversaries cannot distinguish between $\text{iO}c'$ and $\text{iO}BPO(c)$.

Note that doing iO never decreases the "entropy" of a circuit, so $\text{iO}BPO(c)$ is at least as secure as $BPO(c)$. □

10.7 iO *for Polynomial-sized Circuits*

Definition 10.4 (Indistinguishability Obfuscator for NC^1). *Let $\mathcal{C}_\kappa$ be the collection of circuits of size $O(\kappa)$ and depth $O(\log \kappa)$ with respect to gates of bounded fan-in. Then a uniform PPT machine iO_{NC^1} is an* indistinguishability obfuscator *for circuit class* NC^1 *if it is an indistinguishability obfuscator for $\mathcal{C}_\kappa$.*

Given an indistinguishability obfuscator iO_{NC^1} for circuit class NC^1, we shall demonstrate how to achieve an indistinguishability obfuscator iO for all polynomial-sized circuits. The amplification relies on fully homomorphic encryption (FHE).

Definition 10.5 (Homomorphic Encryption)**.** *A* homomorphic encryption scheme *is a tuple of PPT algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *as follows:*

- $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is a semantically-secure public-key encryption scheme.*
- $\mathsf{Eval}(\mathsf{pk}, C, e)$ *takes public key* pk*, an arithmetic circuit* C*, and ciphertext* $e = \mathsf{Enc}(\mathsf{pk}, x)$ *of some circuit input* x*, and outputs* $\mathsf{Enc}(\mathsf{pk}, C(x))$*.*

As an example, the ElGamal encryption scheme is homomorphic over the multiplication function. Consider a cyclic group G of order q and generator g, and let $\mathsf{sk} = a$ and $\mathsf{pk} = g^a$. For ciphertexts $\mathsf{Enc}(\mathsf{pk}, m_1) = (g^{r_1}, g^{ar_1} \cdot m_1)$ and $\mathsf{Enc}(\mathsf{pk}, m_2) = (g^{r_2}, g^{ar_2} \cdot m_2)$, observe that

$$\mathsf{Enc}(\mathsf{pk}, m_1) \cdot \mathsf{Enc}(\mathsf{pk}, m_2) = (g^{r_1+r_2}, g^{a(r_1+r_2)} \cdot m_1 \cdot m_2) = \mathsf{Enc}(\mathsf{pk}, m_1 \cdot m_2)$$

Note that this scheme becomes additively homomorphic by encrypting g^m instead of m.

Definition 10.6 (Fully Homomorphic Encryption)**.** *An encryption scheme is* fully homomorphic *if it is both compact and homomorphic for the class of all arithmetic circuits. Compactness requires that the size of the output of* $\mathsf{Eval}(\cdot, \cdot, \cdot)$ *is at most polynomial in the security parameter* κ*.*

10.7.1 Construction

Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be a fully homomorphic encryption scheme. We require that Dec be realizable by a circuit in NC^1. The obfuscation procedure accepts a security parameter κ and a circuit C whose size is at most polynomial in κ.

1. Generate $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{Gen}(1^\kappa)$ and $(\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow \mathsf{Gen}(1^\kappa)$.
2. Encrypt C, encoded in canonical form, as $e_1 \leftarrow \mathsf{Enc}(\mathsf{pk}_1, C)$ and $e_2 \leftarrow \mathsf{Enc}(\mathsf{pk}_2, C)$.
3. Output an obfuscation $\sigma = (\mathsf{iO}_{\mathsf{NC}^1}(P), \mathsf{pk}_1, \mathsf{pk}_2, e_1, e_2)$ of program $P_{\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{sk}_1, e_1, e_2}$ as described below.

The evaluation procedure accepts the obfuscation σ and program input x.

1. Let U be a universal circuit that computes $C(x)$ given a circuit description C and input x, and denote by U_x the circuit $U(\cdot, x)$ where x is hard-wired. Let R_1 and R_2 be the circuits which compute $f_1 \leftarrow \mathsf{Eval}(U_x, e_1)$ and $f_2 \leftarrow \mathsf{Eval}(U_x, e_2)$, respectively.
2. Denote by ω_1 and ω_2 the set of all wires in R_1 and R_2, respectively. Compute $\pi_1 : \omega_1 \to \{0,1\}$ and $\pi_2 : \omega_2 \to \{0,1\}$, which yield the value of internal wire $w \in \omega_1, \omega_2$ when applying x as the input to R_1 and R_2.

3. Output the result of running $P_{\mathsf{pk}_1,\mathsf{pk}_2,\mathsf{sk}_1,e_1,e_2}(x, f_1, \pi_1, f_2, \pi_2)$.

   Program $P_{\mathsf{pk}_1,\mathsf{pk}_2,\mathsf{sk}_1,e_1,e_2}$ has pk_1, pk_2, sk_1, e_1, and e_2 embedded.

1. Check whether $R_1(x) = f_1 \wedge R_2(x) = f_2$. π_1 and π_2 enable this check in logarithmic depth.

2. If the check succeeds, output $\mathsf{Dec}(\mathsf{sk}_1, f_1)$; otherwise output $\perp$.

The use of two key pairs and two encryptions of C, similar to CCA1-secure schemes seen previously, eliminates the virtual black-box requirement for concealing sk_1 within $\mathsf{iO}_{\mathsf{NC}^1}(P_{\mathsf{pk}_1,\mathsf{pk}_2,\mathsf{sk}_1,e_1,e_2})$.

10.7.2 *Proof of Security*

We prove the indistinguishability property for this construction through a hybrid argument.

Proof. Through the sequence of hybrids, we gradually transform an obfuscation of circuit C_1 into an obfuscation of circuit C_2, with each successor being indistinguishable from its antecedent.

H_0 : This corresponds to an honest execution of $\mathsf{iO}(C_1)$. Recall that $e_1 = \mathsf{Enc}(\mathsf{pk}_1, C_1)$, $e_2 = \mathsf{Enc}(\mathsf{pk}_2, C_1)$, and $\sigma = (\mathsf{iO}_{\mathsf{NC}^1}(P_{\mathsf{pk}_1,\mathsf{pk}_2,\mathsf{sk}_1,e_1,e_2}), \ldots)$.

H_1 : We instead generate $e_2 = \mathsf{Enc}(\mathsf{pk}_2, C_2)$, relying on the semantic security of the underlying fully homomorphic encryption scheme.

H_2 : We alter program $P_{\mathsf{pk}_1,\mathsf{pk}_2,\mathsf{sk}_2,e_1,e_2}$ such that it instead embeds sk_2 and outputs $\mathsf{Dec}(\mathsf{sk}_2, f_2)$. The output of the obfuscation procedure becomes $\sigma = (\mathsf{iO}_{\mathsf{NC}^1}(P_{\mathsf{pk}_1,\mathsf{pk}_2,\mathsf{sk}_2,e_1,e_2}, \ldots)$; we rely on the properties of functional equivalence and indistinguishability of $\mathsf{iO}_{\mathsf{NC}^1}$.

H_3 : We generate $e_1 = \mathsf{Enc}(\mathsf{pk}_1, C_1)$ since sk_1 is now unused, relying again on the semantic security of the fully homomorphic encryption scheme.

H_4 : We revert to the original program $P_{\mathsf{pk}_1,\mathsf{pk}_2,\mathsf{sk}_1,e_1,e_2}$ and arrive at an honest execution of $\mathsf{iO}(C_1)$.

□

10.8 *Identity-Based Encryption*

Another use of indistinguishability obfuscation is to realize identity-based encryption (IBE).

Definition 10.7 (Identity-Based Encryption)**.** *An* identity-based encryption scheme *is a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *as follows:*

- $\mathsf{Setup}(1^\kappa)$ *generates and outputs a master public/private key pair* $(\mathsf{mpk}, \mathsf{msk})$.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ *derives and outputs a secret key* $\mathsf{sk}_{\mathsf{id}}$ *for identity* id.
- $\mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, m)$ *encrypts message m under identity* id *and outputs the ciphertext.*
- $\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, c)$ *decrypts ciphertext c and outputs the corresponding message if c is a valid encryption under identity* id, *or* $\perp$ *otherwise.*

We combine an indistinguishability obfuscator iO with a digital signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$.

- Let $\mathsf{Setup} \equiv \mathsf{Gen}$ and $\mathsf{KeyGen} \equiv \mathsf{Sign}$.
- Enc outputs $\mathsf{iO}(P_m)$, where P_m is a program that outputs (embedded) message m if input sk is a secret key for the given id, or $\perp$ otherwise.
- Dec outputs the result of $c(\mathsf{sk}_{\mathsf{id}})$.

However, this requires that we have encryption scheme where the "signatures" do not exist. We therefore investigate an alternative scheme. Let (K, P, V) be a non-interactive zero-knowledge (NIZK) proof system. Denote by $\mathsf{Com}(\cdot; r)$ the commitment algorithm of a non-interactive commitment scheme with explicit random coin r.

- Let σ be a common random string. $\mathsf{Setup}(1^\kappa)$ outputs $(\mathsf{mpk} = (\sigma, c_1, c_2), \mathsf{msk} = r_1)$, where $c_1 = \mathsf{Com}(0; r_1)$ and $c_2 = \mathsf{Com}(0^{|\mathsf{id}|}; r_2)$.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ produces a proof $\pi = P(\sigma, x_{\mathsf{id}}, s)$ for the following language L: $x \in L$ if there exists s such that

$$\underbrace{c_1 = \mathsf{Com}(0; s)}_{\text{Type I witness}} \vee \underbrace{(c_2 = \mathsf{Com}(\mathsf{id}^*; s) \wedge \mathsf{id}^* \neq \mathsf{id})}_{\text{Type II witness}}$$

- Let $P_{\mathsf{id},m}$ be a program which outputs m if $V(\sigma, x_{\mathsf{id}}, \pi_{\mathsf{id}}) = 1$ or outputs $\perp$ otherwise.

  $\mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, m)$ outputs $\mathsf{iO}(P_{\mathsf{id},m})$.

  We briefly sketch the hybrid argument:

H_0 : This corresponds to an honest execution as described above.

H_1 : We let $c_2 = \mathsf{Com}(\mathsf{id}^*; r_2)$, relying on the hiding property of the commitment scheme.

H_2 : We switch to the Type II witness using $\pi_{\mathsf{id}_i} \forall i \in [q]$, corresponding to the queries issued by the adversary during the first phase of the selective-identity security game.

H_3 : We let $c_1 = \mathsf{Com}(1; r_1)$.

10.9 *Digital Signature Scheme via Indistinguishable Obfuscation*

A digital signature scheme can be constructed via indistinguishable obfuscation (iO). A digital signature scheme is made up of $(\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$.

```
$(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^k)$:
    sk = key of puncturable function and the seed of the PRF $F_k$
    vk = $iO(P_k)$ where $P_k$ is the program:
        $P_k(m, \sigma)$:
            for some OWF function $f$
                return 1 if $f(\sigma) = f(F_k(m))$
                return 0 otherwise
$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$: Output $F_k(m)$.
$\mathsf{Verify}(\mathsf{vk}, m, \sigma)$: Output $P_k(m, \sigma)$.
```

Our security requirements will be that the adversary does wins the following game negligibly:

Challenger Adversary

$(\mathsf{vk}, \mathsf{sk}) = \mathsf{Setup}(1^k)$ and
picks random m

$\xrightarrow{\mathrm{P_k,m}}$

$\xleftarrow{\text{œ},\mathrm{m}^*}$

Adversary wins game if $\mathsf{Verify}(\mathsf{vk}, m^*, \sigma) = 1$

To prove the security of this system, we use a hybrid argument. H_0 is as above.

H_1: Adjust vk so that $\mathsf{vk} = iO(P_{k,m,\alpha})$ where $\alpha = F_k(m)$ and $P_{k,m,\alpha}$ is the program such that:

```
$P_{k,m,\alpha}(m^*, \sigma)$:
    for some OWF $f$
        if $m = m^*$:
            if $f(\sigma) = f(\alpha)$ return 1
            otherwise return 0
        else proceed as $P_k$ from before
            if $f(\sigma) = f(F_k(m^*))$ return 1
            otherwise return 0
```

Note that this program does not change its output for any value. This is indistinguishable from H_0 by indistinguishability obfuscation.

H_2: Adjust α so that it is a randomly sampled value. The indistinguishability of H_2 and H_1 follows from the security of PRG.

H_3: Adjust the program such that instead of α it relies on some β that is compared instead $f(\alpha)$ in the third line.

Any adversary that can break H_3 non-negligibly can break the OWF f with at the value β.

10.10 Public Key Encryption via Indistinguishable Obfuscation

A public key encryption scheme can be constructed via indistinguishable obfuscation. A public key encryption scheme is made up of (Gen, Enc, Dec). The PRG used below is a length doubling PRG.

```
$(\mathsf{pk}, \mathsf{sk}) \leftarrow Gen(1^k)$
  $\mathsf{sk}$ = key of puncturable function and the seed of the PRF $F_k$
  $\mathsf{pk} = iO(P_k)$ where $P_k$ is the program:
    $P_k(m, r)$:
      $t = PRG(r)$
      Output $c = (t, F_k(t) \oplus m)$
$Enc(\mathsf{pk}, m)$: Sample $r$ and output $(\mathsf{pk}(m, r))$.
$Dec(\mathsf{sk} = k, c = (c_1, c_2))$: Output $F_k(\mathsf{sk}, c_1) \oplus c_2$.
```

Our security requirements will be that the adversary does wins the following game negligibly:

Challenger Adversary

$(\mathsf{pk}, \mathsf{sk}) = Gen(1^k)$ and

Randomly sample b from $\{0, 1\}$ and

$c^* = Enc(\mathsf{pk}, b)$ and

$\xrightarrow{\mathrm{P_k, c^*}}$

$\xleftarrow{\mathrm{b^*}}$

Adversary wins game if $b = b^*$

To prove the security of this system, we use a hybrid argument. H_0 is as above.

H_1: Adjust pk so that $\mathsf{pk} = iO(P_{k,\alpha,t})$ where $\alpha = F_k(t)$ and $P_{k,\alpha,t}$ is the program such that:

```
$P_{k,\alpha,t}(m, r)$:
  $t^* = PRG(r)$
  if $t^* = t$, output $(t^*, \alpha \oplus m)$
  else output $(t^*, F_k(t^*) \oplus m)$
```

Note that this program does not change its output for any value. This is indistinguishable from H_0 by indistinguishability obfuscation.

H_2: Adjust α so that it is a randomly sampled value.

H_3: Adjust the program such that t^* is randomly sampled and is not in the range of the PRG.

Any adversary that can win H_3 can guess a random value non-negligibly.

10.11 Indistinguishable Obfuscation Construction from NC^1 iO

A construction of indistinguishable obfuscation from *iO* for circuits in NC^1 is as follows:

Let $P_{k,C}(x)$ be the circuit that outputs the garbled circuit $\widetilde{UC(C,x)}$ with randomness $F_k(x)$ which is a punctured (at k) PRF in NC^1

Note that $UC(C,x)$ outputs $C(x)$ (UC is the "universal" circuit)

$iO(C) \rightarrow$ sample k randomly from $\{0,1\}^{|x|}$ and output $iO_{NC^1}(P_{k,C})$ padded to a length l

As before, we use a hybrid argument to show the security for *iO*.
H_0: $iO(C) = iO_{NC^1}(P_{k,C})$ as above.
$H_{final} = H_{2^n}$: $iO(\mathsf{pk}, c_2)$
$H_1 \cdots H_i$: Create a program $Q_{k,c_1,c_2,i}(x)$ and obfuscate it.

```
$Q_{k,c_1,c_2,i}(x)$:
Sample $k$ randomly
if $x \geq i$, return $P_{k,c_1}(x)$
else , return $P_{k,c_2}(x)$
```

Note that H_i and H_{i+1} are indistinguishable for any value other than $x = i$.
$H_{i,1}$ (between H_i and H_{i+1}): Create a program $Q_{k,c_1,c_2,i,\alpha}(x)$, where $\alpha = Q_{k,c_1,c_2,i}(x)$ and obfuscate it.

```
$Q_{k,c_1,c_2,i,\alpha}(x)$:
Sample $k$ randomly
if $x = i$, return $\alpha$
else , return $Q_{k,c_1,c_2,i}(x)$
```

$H_{i,2}$: Replace α with a random β using fresh coins
$H_{i,3}$: Create the $c_2(x)$ value using fresh coins
$H_{i,4}$: Create the $c_2(x)$ value using $F_k(x)$
$H_{i,5}$: Finish the migration to $Q_{k,c_1,c_2,i+1}$

Note that at H_{final}, the circuit being obfuscated is completely changed from c_1 to c_2.

Bibliography

[1] Mihir Bellare. A note on negligible functions. *Journal of Cryptology*, 15(4):271–284, September 2002.

[2] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532, Gold Coast, Australia, December 9–13, 2001. Springer, Berlin, Heidelberg, Germany.

[3] Sanjam Garg and Mohammad Hajiabadi. Trapdoor functions from the computational Diffie-Hellman assumption. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 362–391, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.

[4] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *21st Annual ACM Symposium on Theory of Computing*, pages 44–61, Seattle, WA, USA, May 15–17, 1989. ACM Press.