

SANJAM GARG

# A COURSE IN THEORY OF CRYPTOGRAPHY

Copyright © 2026 Sanjam Garg

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

This work is licensed under a [Creative Commons “Attribution-NonCommercial-NoDerivatives 4.0 International”](#) license.



*This draft was compiled on Thursday 22<sup>nd</sup> January, 2026.*

# Contents

1	<i>Mathematical Background</i>	7
2	<i>One-Way Functions</i>	17
3	<i>Pseudorandomness</i>	37
4	<i>Private-Key Cryptography</i>	53
5	<i>Digital Signatures</i>	67
6	<i>Public Key Encryption</i>	89
7	<i>Advanced Encryption Schemes</i>	109
8	<i>Proving Computation Integrity</i>	123
9	<i>Secure Computation</i>	179
10	<i>Obfuscation</i>	195



# *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*

This chapter establishes the mathematical foundations necessary for modern cryptography. The concepts introduced here — probabilistic polynomial-time adversaries, negligible probabilities, and computational hardness assumptions — form the bedrock upon which all cryptographic security definitions rest. Understanding these foundations is crucial because they allow us to formalize what it means for a cryptographic scheme to be "secure" in a way that is both mathematically rigorous and practically meaningful.

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<sup>1</sup> — an information-theoretically secure cipher — is not very useful because it requires very large keys.

Below, 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 very small, or negligible, probability (section 1.2).

<sup>1</sup> 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  $m = c \oplus k$ .

### 1.1 *Probabilistic Polynomial Time*

A probabilistic Turing Machine is a generic computer that is allowed to make random choices during its execution. This randomness is essential in cryptography: adversaries may use randomness in their attacks, and cryptographic algorithms themselves require randomness for security. A probabilistic *polynomial time* Turing Machine is one which halts in time polynomial in its input length. Note that any deterministic polynomial-time algorithm is trivially a PPT algorithm (it simply ignores its random tape), but the converse is not neces-

sarily true—probabilistic algorithms can sometimes solve problems more efficiently than deterministic ones. 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. Non-uniformity allows each machine  $M_n$  to have "advice" specific to inputs of length  $n$ , which can be thought of as a polynomial-sized lookup table. This models adversaries who may have precomputed information or special knowledge about specific input lengths. While non-uniform PPT is a stronger model than uniform PPT (any uniform PPT algorithm is also non-uniform PPT), it is often more convenient for security reductions and captures realistic adversarial capabilities.

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

## 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  $\mu(\cdot) : \mathbb{Z}^+ \rightarrow [0, 1]$  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  $\mu(\cdot) : \mathbb{Z}^+ \rightarrow [0, 1]$  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  $n_0 = c^2$ . We will show that for all  $n \geq n_0$ , we have  $\mu(n) = 2^{-n} < n^{-c}$ .

First, we establish that  $2^{-n} = n^{-\frac{n}{\log_2 n}}$ . This follows from taking logarithms:

$$\log_2(2^{-n}) = -n = \log_2(n^{-\frac{n}{\log_2 n}}) = -\frac{n}{\log_2 n} \cdot \log_2 n = -n.$$



Now, we need to show that for  $n \geq n_0 = c^2$ , we have  $\frac{n}{\log_2 n} \geq c$ . Since the function  $f(n) = \frac{n}{\log_2 n}$  is increasing for  $n \geq e$ , it suffices to show that  $f(c^2) \geq c$ . We have:

$$f(c^2) = \frac{c^2}{\log_2(c^2)} = \frac{c^2}{2 \log_2 c} = \frac{c}{2} \cdot \frac{c}{\log_2 c}.$$

For  $c \geq 4$ , we have  $\frac{c}{\log_2 c} \geq 2$  (since  $4/\log_2 4 = 2$  and the function  $c/\log_2 c$  is increasing for  $c \geq e$ ), so  $f(c^2) \geq \frac{c}{2} \cdot 2 = c$ . For  $c \in \{1, 2, 3\}$ , we can verify directly:  $f(1) = 0$ ,  $f(4) = 2$ ,  $f(9) \approx 2.85$ , so we may need to choose a larger  $n_0$  for small  $c$ . However, since we only need existence of some  $n_0$ , we can set  $n_0 = \max(c^2, 16)$  for  $c \leq 3$ , and then  $f(n_0) \geq f(16) = 4 \geq c$ .

Therefore, for  $n \geq n_0$ , we have  $\frac{n}{\log_2 n} \geq c$ , which implies  $-\frac{n}{\log_2 n} \leq -c$ . Since  $n > 1$ , this gives us:

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

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!<sup>2</sup> 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.

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

This function is obtained by interleaving negligible and noticeable functions. For even  $n$ ,  $\mu(n) = 2^{-n}$  which is negligible. For odd  $n$ ,  $\mu(n) = n^{-3}$  which is noticeable. Specifically, it's not negligible because  $\mu(n) = n^{-3} > n^{-4}$  for all odd  $n$ , violating the definition. It's not noticeable because  $\mu(n) = 2^{-n} < n^{-c}$  for any fixed  $c$  and all sufficiently large even  $n$ , so there's no  $n_0$  such that  $\mu(n) > n^{-c}$  for all  $n \geq n_0$ .

*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.

*Practical Intuition.* To build intuition, consider that a negligible probability means the event is so rare it's essentially impossible in practice. For example, if a cryptographic scheme has a negligible probability  $\mu(n) = 2^{-128}$  of being broken, this is smaller than the probability of being struck by lightning twice in one's lifetime (approximately  $10^{-12}$ ). Even for  $n = 64$ , we have  $2^{-64} \approx 5 \times 10^{-20}$ , which is astronomically small. This is why we can safely ignore negligible probabilities when reasoning about cryptographic security.

**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 functions:

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.  $\psi_4(n) = \text{poly}(\mu(n))$ , where  $\text{poly}(\cdot)$  is an unspecified polynomial function. (Assume that the output is also clamped to  $[0, 1]$  to satisfy the definition)

*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  $\mu$  and  $\nu$  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  $\forall n \geq n_2, \nu(n) < n^{-c}$ . 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  $\mu$  and  $\nu$  are negligible, there exist  $n_1$  and  $n_2$  such that  $\forall n \geq n_1, \mu(n) \leq n^{-c-1}$  and  $\forall n \geq n_2, \nu(n) \leq n^{-c-1}$ . Setting  $n_0 = \max(n_1, n_2, 3)$  we have that for every  $n \geq n_0$ ,

$$\psi_2(n) = \min\{\mu(n) + \nu(n), 1\} < n^{-c-1} + n^{-c-1} = 2n^{-c-1} = 2n^{-1} \cdot n^{-c} \leq n^{-c}$$

since  $n \geq 3$  implies  $2n^{-1} \leq 1$ .

3. The proof that  $\psi_3(n) = \mu(n) \cdot \nu(n)$  is negligible follows similarly: since both  $\mu(n)$  and  $\nu(n)$  are less than  $n^{-c}$  for sufficiently large  $n$ , their product is less than  $n^{-2c} < n^{-c}$ .

4. For  $\psi_4(n) = \text{poly}(\mu(n))$ , since  $\mu(n)$  is negligible, we have  $\mu(n) < n^{-c}$  for any  $c$  and sufficiently large  $n$ . If  $\text{poly}$  is a polynomial of degree  $d$ , then  $\text{poly}(\mu(n)) < \text{poly}(n^{-c}) = O(n^{-cd})$ , which is still negligible. The details are left as an exercise.

□

### 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 provide concrete instantiations. These assumptions—discrete logarithm, Diffie-Hellman, and factoring—form the foundation for most practical public-key cryptography. They have been studied for decades and are widely believed to be hard, though proving their hardness would resolve major open problems in complexity theory (such as showing  $P \neq NP$ ).

#### 1.3.1 The Discrete-Log Family of Problem

Consider a group  $G$  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  $G$ . In this group, given  $g^x$  for a random  $x \in \{1, \dots, p-1\}$  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 call a group ensemble.

*Group Ensemble.* A group ensemble is a set of finite cyclic groups  $\mathcal{G} = \{G_n\}_{n \in \mathbb{N}}$ . For the group  $G_n$ , we assume that given two group elements in  $G_n$ , their group operation (which we denote additively or multiplicatively depending on context) can be computed in polynomial in  $n$  time. Additionally, we assume that given  $n$  the generator  $g$  of  $G_n$  can be computed in polynomial time.

**Definition 1.5** (Discrete-Log Assumption). *We say that the discrete-log assumption holds for the group ensemble  $\mathcal{G} = \{G_n\}_{n \in \mathbb{N}}$ , 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

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

is a negligible function.

**Definition 1.7** (Decisional Diffie-Hellman (DDH) Assumption). We say that the Decisional 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

$$\mu_{\mathcal{A}}(n) = \left| \Pr_{x,y \leftarrow |\mathbb{G}_n|} [\mathcal{A}(g, g^x, g^y, g^{xy}) = 1] - \Pr_{x,y,z \leftarrow |\mathbb{G}_n|} [\mathcal{A}(g, g^x, g^y, g^z) = 1] \right|$$

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  $\mathcal{G}$ . Figure 1.1 illustrates the relationship between these assumptions.

*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  $p$ .<sup>3</sup> For this group the CDH Assumption is conjectured to be true.

However, the DDH Assumption in this group can be shown to be false. Can you show how?

*Legendre Symbol.* 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

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

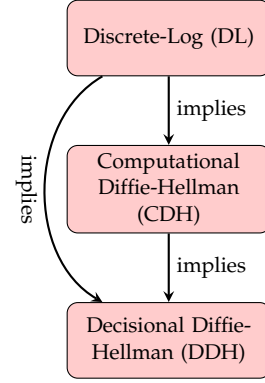


Figure 1.1: Relationship between discrete-log assumptions. DL is the weakest assumption; breaking DL allows breaking CDH, and breaking CDH allows breaking DDH. The reverse implications are not known to hold.

<sup>3</sup> 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.

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

*DDH is easy in this group.* 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.

*Deployment.* The group  $\mathbb{Z}_p^*$  has been widely deployed in classical cryptographic protocols, including the original Diffie-Hellman key exchange, ElGamal encryption, and the Digital Signature Algorithm (DSA). However, due to the weakness of the DDH assumption in this group, modern protocols typically use subgroups of prime order or elliptic curve groups instead. The group  $\mathbb{Z}_p^*$  remains relevant for protocols that only require CDH security, such as some variants of ElGamal encryption.

2. Let  $p = 2q + 1$  where both  $p$  and  $q$  are prime.<sup>4</sup> Next, let  $Q$  be the order- $q$  subgroup of quadratic residues in  $\mathbb{Z}_p^*$ . For this group, the DDH assumption is believed to hold.

<sup>4</sup> 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.

*Deployment.* The order- $q$  subgroup of quadratic residues in  $\mathbb{Z}_p^*$  (where  $p = 2q + 1$  is a safe prime) has been used in various cryptographic protocols, including some implementations of the Diffie-Hellman key exchange and ElGamal encryption that require DDH security. However, these groups require larger key sizes (typically 2048-4096 bits) to achieve security levels comparable to elliptic curves, making them less efficient than modern elliptic curve alternatives. They are still used in some legacy systems and standards, but elliptic curve groups are generally preferred for new deployments due to their efficiency advantages.

3. Let  $N = pq$  where  $p, q, \frac{p-1}{2}$  and  $\frac{q-1}{2}$  are primes. Let  $QR_N$  be the cyclic subgroup of quadratic residues of order  $\phi(N) = (p-1)(q-1)$ . For this group  $QR_N$ , the DDH assumption is also believed to hold.

*Deployment.* The group  $QR_N$  is closely related to RSA, which operates in  $\mathbb{Z}_N^*$  (where  $N = pq$ ). While standard RSA encryption and decryption work in the full group  $\mathbb{Z}_N^*$  rather than specifically in

$\mathbb{QR}_N$ , the quadratic residues play an important role in RSA's security analysis and some variants. The group  $\mathbb{QR}_N$  has been used in cryptographic constructions that require both DDH security and a connection to the factoring problem (as we will see, CDH in  $\mathbb{QR}_N$  implies factoring). However, for discrete-log-based protocols, this group is less commonly deployed in practice compared to elliptic curve groups or safe prime subgroups, as it requires composite moduli and the security depends on both the discrete logarithm and factoring assumptions. It appears primarily in advanced cryptographic constructions and security reductions.

4. Elliptic Curve groups: An elliptic curve over a finite field  $\mathbb{F}_p$  is defined by an equation of the form  $y^2 = x^3 + ax + b$  (for Weierstrass form) or  $y^2 = x^3 + ax^2 + x$  (for Montgomery form), where  $a, b \in \mathbb{F}_p$  and the curve is non-singular. The set of points  $(x, y) \in \mathbb{F}_p \times \mathbb{F}_p$  satisfying the curve equation, together with a special "point at infinity"  $\mathcal{O}$ , forms an abelian group under a geometric addition operation. For cryptographic applications, we typically work with a cyclic subgroup of prime order  $q$  of the elliptic curve group. The discrete-log, CDH, and DDH assumptions are all believed to hold for well-chosen elliptic curve groups.<sup>5</sup>

*secp256k1.* The secp256k1 curve is a Koblitz curve (a special type of elliptic curve) defined by the equation  $y^2 = x^3 + 7$  over the prime field  $\mathbb{F}_p$  where  $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$ . It has prime order and provides approximately 128 bits of security. secp256k1 is most notably used in Bitcoin and Ethereum for ECDSA signatures, making it one of the most widely deployed elliptic curves in terms of economic value. Unlike Curve25519, secp256k1 uses the Weierstrass form, and its parameters were standardized by the Standards for Efficient Cryptography Group (SECG).

*Curve25519.* Curve25519 is a widely-used elliptic curve designed by Daniel J. Bernstein in 2005. It is a Montgomery curve defined by the equation  $y^2 = x^3 + 486662x^2 + x$  over the prime field  $\mathbb{F}_p$  where  $p = 2^{255} - 19$ . The curve has prime order  $q = 2^{252} + 2774231777372353535851937790883648493$ , providing approximately 128 bits of security. The base point (generator) has  $x$ -coordinate  $x = 9$ . Curve25519 offers several advantages: (1) it enables fast, constant-time implementations that resist side-channel attacks, (2) it provides strong security with relatively small key sizes (256 bits), and (3) its parameters were chosen transparently

<sup>5</sup> There are infinitely many elliptic curves over various finite fields, allowing us to define group ensembles. The security of elliptic curve cryptography relies on the hardness of the elliptic curve discrete logarithm problem (ECDLP), which is believed to be exponentially harder than the discrete logarithm problem in finite fields of similar size.

for mathematical simplicity, avoiding concerns about hidden backdoors. Curve25519 is widely deployed in modern cryptographic protocols, including the Ed25519 signature scheme and the X25519 key exchange protocol.

*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 a 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  $N$ . The reduction works by using  $\mathcal{A}$  to find square roots that help factor  $N$ . Specifically,  $\mathcal{B}$  on input  $N$  proceeds as follows:

1. Sample a random value  $r$  in  $\mathbb{Z}_N^*$  and compute  $v := r^2 \bmod N$ . Note that  $v \in \mathbb{QR}_N$ . Then compute  $g := v^2 \bmod N$ . Note that  $g$  is a generator of  $\mathbb{QR}_N$  with high probability, since  $v$  is a random quadratic residue and squaring preserves the cyclic structure.
2. Sample  $x, y \leftarrow [N]$ .<sup>6</sup> The statistical closeness follows from the fact that  $N$  and  $\phi(N) = (p-1)(q-1)$  are relatively close:  $|N - \phi(N)| = p + q - 1 = O(\sqrt{N})$ , so the statistical distance is  $O(1/\sqrt{N})$ , which is negligible.
3. Let  $u := \mathcal{A}(g, g^x \cdot v, g^y \cdot v)$ .<sup>7</sup> The inputs  $g^x \cdot v$  and  $g^y \cdot v$  are constructed to "hide" the actual CDH inputs while still allowing  $\mathcal{A}$  to compute something useful. Then compute  $w := \frac{u}{g^{xy} \cdot v^{x+y}}$ .
4. If  $w^2 \equiv v \pmod{N}$  and  $w \neq \pm r$ , then compute the factors of  $N$  as  $\gcd(N, w - r)$  (or  $\gcd(N, w + r)$ ) and  $N/\gcd(N, w - r)$  (or  $N/\gcd(N, w + r)$ ). Otherwise, output  $\perp$ .

<sup>6</sup> Note that sampling  $x, y$  uniformly from  $[N]$  is statistically close to sampling  $x, y$  uniformly from  $[\phi(N)]$ .

<sup>7</sup> Note that  $g^x \cdot v$  where  $x \leftarrow [N]$  is statistically close to  $g^x$  where  $x \leftarrow [N]$ .

To understand why this works, observe that if  $\mathcal{A}$  correctly computes the CDH value, then the returned values  $u = g^{(x+2^{-1})(y+2^{-1})} = v^{2xy+x+y+2^{-1}}$ . Here,  $2^{-1}$  denotes the multiplicative inverse of 2 modulo the order of the group, which exists since the order is odd (as

$\phi(N) = (p-1)(q-1)$  and both  $\frac{p-1}{2}$  and  $\frac{q-1}{2}$  are prime, implying  $p$  and  $q$  are odd). Consequently, the computed value  $w = v^{2^{-1}}$ . This follows from:  $w = \frac{u}{g^{xy} \cdot v^{x+y}} = \frac{v^{2xy+x+y+2^{-1}}}{v^{2xy+x+y}} = v^{2^{-1}}$ . This means  $w^2 = (v^{2^{-1}})^2 = v$ , so  $w$  is a square root of  $v$ . In  $\mathbb{Z}_N^*$  where  $N = pq$ , each quadratic residue has exactly 4 square roots. The residue  $v$  has four square roots:  $\pm r$  (the two "easy" roots that we already know) and  $\pm w$  (the two roots computed via the CDH oracle). With probability  $\frac{1}{2}$ , we have that  $w \neq \pm r$ . If  $w^2 \equiv v \pmod{N}$  but  $w \neq \pm r \pmod{N}$ , then  $w^2 - r^2 = (w-r)(w+r) \equiv 0 \pmod{N}$ , which means  $N$  divides  $(w-r)(w+r)$  but neither factor individually. Since  $N = pq$ , this allows us to recover the factors via  $\gcd(N, w-r)$  (or  $\gcd(N, w+r)$ ) and the complementary factor. In this case,  $\mathcal{B}$  can factor  $N$ .  $\square$



## 2

# One-Way Functions

Consider password hashing: we want a function that's easy to compute (hash the password) but hard to invert (recover the password from the hash). This is exactly what one-way functions formalize. 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.*<sup>1</sup>

<sup>1</sup> Quote by Silvio Micali in personal communication with Joe Kilian.

Thus, basing cryptographic tasks on the *minimal* necessary assumptions is a key tenet in cryptography. Towards this goal, rather than 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, realizing cryptographic tasks from just one-way functions would be ideal.

### 2.1 Definition

A one-way function  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$  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.

**Definition 2.1** (One-Way Functions). A function  $f : \{0,1\}^* \rightarrow \{0,1\}^*$  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

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

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  $\mathcal{A}$ .

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  $x'$ . For instance, if  $f(x) = x \bmod 2$ , then  $f(0) = f(2) = 0$ , and the adversary succeeds if it outputs either 0 or 2.

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  $f$ . The point is that requiring perfect security (probability 0) is too strong because 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. Even a trivial adversary that always outputs a fixed value succeeds with non-zero 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.

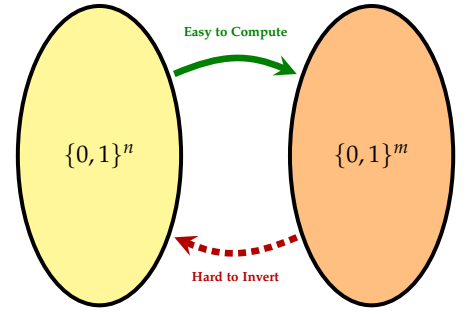


Figure 2.1: Visualizing One-way Functions

<sup>2</sup> 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.

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 imply that  $P \neq NP$  (see Exercise 2.3). This connection to complexity theory highlights the fundamental nature of one-way functions: they represent the boundary between what is efficiently computable and what is not.

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 holds for group ensemble  $\mathcal{G} = \{\mathbb{G}_n\}$  then we have that the function family  $\{f_n\}$  where  $f_n : \{1, \dots, |\mathbb{G}_n|\} \rightarrow \mathbb{G}_n$  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 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. More formally, if an adversary could break  $g$  with non-negligible probability, we could use it to break  $f$  by handling the negligible case where  $x = x_0$  separately.

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. This illustrates why cryptographic definitions are asymptotic: they allow for 'bad' behavior on finitely many inputs as long as security holds in the limit.

### 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  $f(f(x))$ . One might try to invert  $f$  by first inverting  $f^2$  to get some  $x'$  with  $f^2(x') = f^2(x)$ , then inverting  $f$  on  $f(x')$ . But this fails because  $f^2(x') = f^2(x)$  doesn't guarantee  $f(x') = f(x)$  - we might have  $f(x') \neq f(x)$  but  $f(f(x')) = f(f(x))$ . Thus, this intuition is incorrect and highlights the delicacy when working with cryptographic assumptions and primitives. In particular, assuming one-way functions exist 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  $\mu_{\mathcal{A},f}(n) = \Pr_{x \leftarrow \{0,1\}^n}[\mathcal{A}(1^{2n}, f(x)) \in f^{-1}(f(x))]$  is non-negligible. Using such an  $\mathcal{A}$  we will describe a construction of adversary  $\mathcal{B}$  such that  $\mu_{\mathcal{B},g}(n) = \Pr_{x \leftarrow \{0,1\}^n}[\mathcal{B}(1^n, g(x)) \in g^{-1}(g(x))]$  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:

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

(Here, the set  $\{0, 1\}^n \| g^{-1}(g(x))$  means all strings that start with any  $n$ -bit prefix followed by a preimage of  $g(x)$  under  $g$ .) 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. One might think that dropping a single bit can't hurt security much, but we'll show that even this seemingly minor modification can completely break one-wayness. 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  $h : \{0, 1\}^n \rightarrow \{0, 1\}^n$ . We define the function  $g : \{0, 1\}^{2n} \rightarrow \{0, 1\}^{2n}$  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,

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

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.<sup>3</sup> This contradicts the one-wayness of  $h$ .

<sup>3</sup> 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.

□

**Step 3: Prove that  $g'$  is not one-way.** We construct the new function  $g' : \{0, 1\}^{2n} \rightarrow \{0, 1\}^{2n-1}$  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  $0^{n-1} \| \bar{y}$ .

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

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

### 2.3 Hardness Amplification

In this section, we show that even a very *weak* form of one-way functions suffices for constructing one-way functions as defined previously. For this section, we refer to this previously defined notion as strong one-way functions. This is a powerful result: it shows that even if a function is only ‘weakly’ one-way (i.e., invertible with noticeable probability), we can amplify it to get a ‘strong’ one-way function (invertible with only negligible probability).

**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*

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

**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

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

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. More concretely, if the weak OWF is hard to invert on at least an  $\alpha_f(n)$  fraction of inputs, then with  $q$  independent copies, the probability that ALL are easy to invert is at most  $(1 - \alpha_f(n))^q$ , which becomes negligible for large enough  $q$ .

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

$f$ , namely  $\mu_{\mathcal{A},f}(n) = \Pr_{x \leftarrow \{0,1\}^n}[\mathcal{A}(1^n, f(x)) \in f^{-1}(f(x))] > 1 - \alpha_f(n)$  for sufficiently large  $n$ .

Note that: (1)  $\mathcal{A}(1^n, y)$  iterates at most  $T = \frac{4n^2}{\alpha_f(n)\mu_{\mathcal{B},g}(nq)}$  times and each iteration runs in polynomial time. (2)  $\mu_{\mathcal{B},g}(nq)$  is a non-negligible function. Since  $\mu_{\mathcal{B},g}$  is non-negligible, there exists a noticeable function  $\delta$  such that  $\mu_{\mathcal{B},g}(nq) \geq \delta(nq)$  for infinitely many  $n$ . For such  $n$ ,  $T$  is polynomial. This implies 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  $\Pr_{x \leftarrow \{0,1\}^n}[\mathcal{A}(1^n, f(x)) = \perp] < \alpha_f(n)$  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. The calls share the same  $x$  value, so if  $\mathcal{B}$  fails on one call, it might be more likely to fail on others. We need to carefully analyze the dependency structure. 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:

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

(Here,  $\Pr_{\mathcal{B}}$  denotes the probability over the randomness of  $\mathcal{B}$ , which includes the random choice of  $i$  and the random  $x_j$  values.) We start by proving that the size of  $S$  is small. More formally,

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

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

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

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

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

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

which is a contradiction.  $\square$

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

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

which is a contradiction.  $\square$



Then we have that:

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

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

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

for sufficiently large  $n$ . This concludes the proof.  $\square$

## 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 at least one hardness assumption holds (which we may or may not discover).

This is a remarkable result: it shows that if OWFs exist at all, we can construct one explicitly without knowing which specific assumption to rely on.

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  $f : \{0,1\}^n \rightarrow \{0,1\}^n$  be a one-way function computable in time  $n^c$ . Construct  $g : \{0,1\}^{n+n^c} \rightarrow \{0,1\}^{n+n^c}$  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  $\mu_{\mathcal{A},g}(n + n^c) = \Pr_{(x,y) \leftarrow \{0,1\}^{n+n^c}} [\mathcal{A}(1^{n+n^c}, g(x, y)) \in g^{-1}(g(x, y))]$  is non-negligible. Then we use  $\mathcal{A}$  to construct  $\mathcal{B}$  such that  $\mu_{\mathcal{B},f}(n) = \Pr_{x \leftarrow \{0,1\}^n} [\mathcal{B}(1^n, f(x)) \in f^{-1}(f(x))]$  is also non-negligible.

$\mathcal{B}$  on input  $z \in \{0,1\}^n$ , samples  $y \leftarrow \{0,1\}^{n^c}$ , 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.  $\square$

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.*  $h : \{0,1\}^n \rightarrow \{0,1\}^n$  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  $\tilde{\mathcal{B}}$  to invert  $\tilde{M}$ : on input  $y$  outputs the  $(n - \log n)$  lower-order bits of  $\mathcal{A}(1^n, \tilde{M}||y)$ . Then

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

Observe that for sufficiently large  $n$  it holds that

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

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

## 2.5 Hardness Concentrated 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  $f(x)$ ? This question is important because hard bits are useful for constructing pseudorandom generators and other cryptographic primitives. The answer is subtle: not every OWF has a hard bit, but we can always construct one that does.

**Definition 2.3** (Hard Concentrate Bit). *Let  $f : \{0,1\}^n \rightarrow \{0,1\}^n$  be a one-way function.  $B : \{0,1\}^n \rightarrow \{0,1\}$  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

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

**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. Information-theoretic security means the adversary literally has no information, while computational security means the adversary could theoretically compute it but it's computationally infeasible. The Goldreich-Levin theorem shows we can achieve computational 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  $f : \{0,1\}^n \rightarrow \{0,1\}^n$  is a one-way function, then there exists a one-way function  $g : \{0,1\}^{n+\log n} \rightarrow \{0,1\}^{n+\log n}$  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  $i^{\text{th}}$  bit of  $x$ .*

*Proof.* Define  $g : \{0,1\}^{n+\log(n)} \rightarrow \{0,1\}^{n+\log(n)}$  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  $y^{\text{th}}$  bit, and  $x_y$  is the  $y^{\text{th}}$  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  $\frac{1}{2} + \text{negl}(n)$  probability.  $\square$

### 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  $f : \{0,1\}^n \rightarrow \{0,1\}^n$  be a one-way function. Define a function  $g : \{0,1\}^{2n} \rightarrow \{0,1\}^{2n}$  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  $B(x, r) = \sum_{i=1}^n x_i r_i \pmod{2}$ .*

**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  $\epsilon$  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. While unrealistic, this case illustrates the key technique: using the hard bit to extract individual bits of  $x$ . Let  $e^i$  denote the one-hot  $n$ -bit string  $0 \cdots 010 \cdots 0$ , 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.

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

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

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. The threshold of  $3/4$  is chosen so that with two calls, we can use a union bound. With probability  $\geq 3/4 + \epsilon$  of being correct, the probability that both calls are wrong is  $\leq (1/4 - \epsilon)^2$ , which is small enough for Chebyshev's inequality to work. 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
   $x'_i \leftarrow \mathcal{A}(1^{2n}, f(x)||e^i)$ 
end for
return  $x'_1 \cdots x'_n || r$ 

```

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

```

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

```

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

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

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:

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

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

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

Assume that  $\Pr_{x \leftarrow \{0,1\}^n} [x \in G] < \frac{\epsilon(n)}{2}$ . 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  $x \in G$ :

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

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  $Y_i^1, \dots, Y_i^T$  are independent and identical random variables, and for all  $t \in \{1, \dots, T\}$ , 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  $X_1, \dots, X_m$  be i.i.d. random variables taking values 0 or 1. Let  $\Pr[X_i = 1] = p$ .

$$\begin{aligned} \text{By Chebyshev's Inequality, } \Pr[|\sum X_i - pm| \geq \delta m] &\leq \frac{1}{4\delta^2 m}. \\ &\leq \frac{1}{4\epsilon(n)^2 T} = \frac{1}{2n}. \end{aligned}$$

Then, completing the argument, we have

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

**Real Case.** 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. The "magic" here is the use of pairwise independence: by guessing  $\log T$  values and using XOR combinations, we get  $T$  pairwise independent samples. This is a beautiful technique that appears in many cryptographic proofs. Just like the previous two cases, we start by describing the algorithm  $\mathcal{B}$  in Figure 2.5.

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  $b_\ell$ 's are correct, i.e.,

```

 $T = \frac{2n}{\epsilon(n)^2}$ 
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  $L \subseteq \{1, 2, \dots, \log T\}$  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
   $x_i' \leftarrow \text{majority of } \{x_i^\emptyset, \dots, x_i^{\{\log T\}}\}$ 
end for
return  $x_1' \dots x_n' || R$ 

```

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



$b_\ell = B(x, s_\ell)$ . 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  $k^{\text{th}}$ -bit of  $s$  with  $(s)_k$ ):

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

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  $Y_i^\emptyset, \dots, Y_i^{\lceil \log T \rceil}$  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

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

and with an identical argument we obtain that:

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

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

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

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  $Y_i^\emptyset, \dots, Y_i^{\lceil \log T \rceil}$  are pairwise indepen-

**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  $\{X_1, \dots, X_m\}$  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  $X_1, \dots, X_m$  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

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

Observe that

$$\begin{aligned}
\text{Var}(Y) &= \mathbb{E} [Y^2] - (\mathbb{E}[Y])^2 \\
&= \sum_{i=1}^m \sum_{j=1}^m (\mathbb{E} [X_i X_j] - \mathbb{E} [X_i] \mathbb{E} [X_j]) \\
&\text{By pairwise independence, for } i \neq j, \\
&\mathbb{E} [X_i X_j] = \mathbb{E} [X_i] \mathbb{E} [X_j]. \\
&= \sum_{i=1}^m \mathbb{E} [X_i^2] - \mathbb{E} [X_i]^2 \\
&= mp(1-p).
\end{aligned}$$

Hence

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

□

dent and  $\Pr[Y_i^L = 1] = \Pr[x_i^L = x_i] \geq \frac{1}{2} + \frac{\epsilon(n)}{2}$ .

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

Completing the proof, we have that:

$$\begin{aligned}
&\Pr_{x,r}[\mathcal{B}(1^{2n}, g(x, r)) = (x, r)] \\
&\geq \Pr_{\{b_\ell, s_\ell\}_\ell} [F] \cdot \Pr_x[x \in G] \cdot \Pr[x'_1 = x_1, \dots, x'_n = x_n \mid x \in G \wedge F] \\
&\geq \frac{1}{T} \cdot \frac{\epsilon(n)}{2} \cdot \left( 1 - \sum_{i=1}^n \Pr[x'_i \neq x_i \mid x \in G \wedge F] \right) \\
&\geq \frac{\epsilon(n)^2}{2n} \cdot \frac{\epsilon(n)}{2} \cdot \left( 1 - n \cdot \frac{1}{2n} \right) = \frac{\epsilon(n)^3}{8n}
\end{aligned}$$

□

## 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  $\mu(f(\cdot))$ <sup>4</sup> are negligible functions.

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

**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  $f : \{0, 1\}^n \rightarrow \{0, 1\}^{\lceil \log_2 n \rceil}$ .

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

**Exercise 2.6.** Prove or disprove: If  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  is a one-way function, then  $g : \{0, 1\}^n \rightarrow \{0, 1\}^{n - \log n}$  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  $i \stackrel{\$}{\leftarrow} \{1, 2, \dots, q\}$ .

**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  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  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>



# 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<sup>1</sup>. 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.

<sup>1</sup> 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.

**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  $S = \text{Support}(X_n) \cup \text{Support}(Y_n)$ .*

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  $\Omega$  be the sample space for  $X_n$  and  $Y_n$ .

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

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

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

Now we will show the main result of the lemma.

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

□

### 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.  $\square$

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  $t : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$  let the  $t$ -product of  $\{Z_n\}_n$  be*

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

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

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

*then*

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

*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

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

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

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

Similarly, when  $i = t$ :

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

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

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

Since  $|P_0 - P_t| \geq \varepsilon(n)$ , it follows that  $|P_0 - P_1| + |P_1 - P_2| + \dots + |P_{t-1} - P_t| \geq \varepsilon(n)$ . 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:

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

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

<sup>2</sup>  $\mathcal{B}$  is non-uniform, so it can "know" which value of  $k$  it should use.

This is a contradiction.  $\square$

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:

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

This implies that

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

which is the same thing as saying that

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

$\square$

### 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  $G : \{0, 1\}^n \rightarrow \{0, 1\}^{n+m}$  with  $m = \text{poly}(n)$  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  $G : \{0, 1\}^n \rightarrow \{0, 1\}^{n+1}$ , we construct a new PRG  $F : \{0, 1\}^n \rightarrow \{0, 1\}^{n+l}$  as follows ( $l$  is polynomial in  $n$ ).

- Input:  $S_0 \xleftarrow{\$} \{0, 1\}^n$ .
- $\forall i \in [l] = \{1, 2, \dots, l\}$ ,  $(\sigma_i, S_i) := G(S_{i-1})$ , where  $\sigma_i \in \{0, 1\}$ ,  $S_i \in \{0, 1\}^n$ .
- Output:  $\sigma_1 \sigma_2 \dots \sigma_l S_l$ .

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

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

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

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

there exists  $k \in \{0, 1, \dots, l-1\}$  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:

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

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

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

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

□

## 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  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  be a OWP. We construct  $G : \{0, 1\}^{2n} \rightarrow \{0, 1\}^{2n+1}$  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:

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

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

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

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

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

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

$$H'_1 = f(x) || r || (1 - B(x, r)), \text{ where } x, r \xleftarrow{\$} \{0, 1\}^n.$$

Then we have

$$\begin{aligned} \Pr[\mathcal{A}(H_1) = 1] &= \Pr[\sigma = B(x, r)] \Pr[\mathcal{A}(H_0) = 1] + \Pr[\sigma = 1 - B(x, r)] \Pr[\mathcal{A}(H'_1) = 1] \\ &= \frac{1}{2} \Pr[\mathcal{A}(H_0) = 1] + \frac{1}{2} \Pr[\mathcal{A}(H'_1) = 1]. \end{aligned}$$

Hence

$$\begin{aligned} \Pr[A(H_1) = 1] - \Pr[A(H_0) = 1] &= \frac{1}{2} \Pr[A(H'_1) = 1] - \frac{1}{2} \Pr[A(H_0) = 1], \\ \frac{1}{2} |\Pr[A(H_0) = 1] - \Pr[A(H'_1) = 1]| &= |\Pr[A(H_1) = 1] - \Pr[A(H_0) = 1]| \geq v(n), \\ |\Pr[A(H_0) = 1] - \Pr[A(H'_1) = 1]| &\geq 2v(n). \end{aligned}$$

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  $\sigma \xleftarrow{\$} \{0, 1\}$ . Then we have

$$\begin{aligned} &\Pr[\mathcal{B}(f(x)||r) = B(x, r)] \\ &= \Pr[\sigma = B(x, r)] \Pr[\mathcal{A}(f(x)||r||\sigma) = 1 | \sigma = B(x, r)] + \\ &\quad \Pr[\sigma = 1 - B(x, r)] \Pr[\mathcal{A}(f(x)||r||\sigma) = 0 | \sigma = 1 - B(x, r)] + \\ &= \frac{1}{2} (\Pr[\mathcal{A}(f(x)||r||B(x, r)) = 1] + 1 - \Pr[\mathcal{A}(f(x)||r||1 - B(x, r)) = 1]) \\ &= \frac{1}{2} + \frac{1}{2} (\Pr[A(H_0) = 1] - \Pr[A(H'_1) = 1]) \\ &\geq \frac{1}{2} + v(n). \end{aligned}$$

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

### 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  $\text{poly}(n)$  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  $F_1, F_2, \dots, F_n, \dots$  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  $\phi$  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<sup>3</sup>. 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.

<sup>3</sup> An efficiently computable function requires only  $n$  bits (the key) to describe, while a random function requires  $n \cdot 2^n$  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*

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

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  $G : \{0, 1\}^n \rightarrow \{0, 1\}^{2n}$ , 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  $F^{(K)} : \{0, 1\}^n \rightarrow \{0, 1\}^n$  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,  $i = K$ .

The construction can be viewed as a binary tree of depth  $n$ , as shown in Figure 3.1<sup>4</sup>.

**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  $i \in \{0, 1, \dots, n\}$  where the hybrid  $i$  is defined as follows:

$$H_i^{(K_i)}(x_1 x_2 \dots x_n) := G_{x_n}(G_{x_{n-1}}(\dots (G_{x_{i+1}}(K_i(x_1 \dots x_{i-1} x_i))) \dots)),$$

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  $\mathcal{A}$ <sup>5</sup>. 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  $R_i : \{0, 1\}^i \rightarrow \{0, 1\}^n$  and  $S_i : \{0, 1\}^{i+1} \rightarrow \{0, 1\}^n$  be two random functions. We define sub-hybrid  $H_{i,j}^{(R_i, S_i)}(x_1 x_2 \dots x_n)$  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  $(x_1 \dots x_i) \in L$ <sup>6</sup>:
  - (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  $y \leftarrow R_i(x_1 \dots x_i)$ .

<sup>4</sup> 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  $y$ .

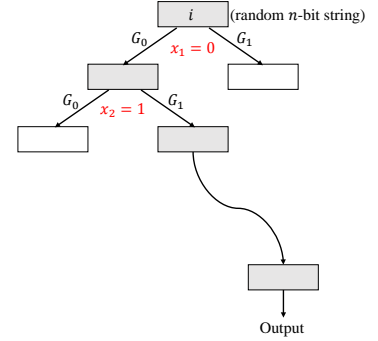


Figure 3.1: View the construction as a binary tree

<sup>5</sup> 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.

<sup>6</sup> Captures the first  $j$  queries or any query with repeated  $i$ -bit prefix to a previous query.

(b) For  $a \in i + 1 \dots n$ : 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<sup>7</sup>. 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  $G(U_n)$ .  $\mathcal{B}^{A, i^*, j^*}(1^n, z)$ , where  $z \in \{0, 1\}^{2n}$  ( $z$  could be either from  $U_{2n}$  or  $G(U_n)$ ) and we assume the knowledge of  $i^*, j^*$ <sup>8</sup>, 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  $H_{i^*, j^*}$ <sup>9</sup>.
3. For the  $j^* + 1$ -th query  $(x_1 \dots x_n)$ , do the following:
  - (a) Set  $y \leftarrow z_{x_{i^*+1}}$ .
  - (b) For  $a \in i^* + 2 \dots n$ : 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. □

<sup>7</sup> 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  $i^*, j^*$ .

<sup>8</sup> As mentioned before, it can be guessed with slight loss in distinguishing advantage.

<sup>9</sup> 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.

### 3.5 PRFs from DDH: Naor-Reingold PRF

We will now describe a PRF function family  $F_n : \mathcal{K} \times \{0, 1\}^n \rightarrow \mathbb{G}_n$  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  $K = (h, u_1, \dots, u_n)$ , where  $u, u_0 \dots u_n$  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  $\{0, 1\}^n$ .



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

Next, we will prove that the function  $F_n$  is a pseudo-random function or that  $\{F_n\}$  is a pseudo-random function ensemble.<sup>10</sup>

**Lemma 3.4.** *Assuming the DDH Assumption (see Definition 1.7) for  $\{G_n\}$  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<sup>11</sup>.

Let  $R_n$  be random function from  $\{0, 1\}^n \rightarrow G_n$ . 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  $S_n^j : \{0, 1\}^j \rightarrow \{0, 1, \dots, |G_n| - 1\}$ , then hybrid  $H_n^j$  is defined as<sup>12</sup>:

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

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  $R_n$ <sup>13</sup>. 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  $C_n^j : \{0, 1\}^j \rightarrow \{0, \dots, |G_n| - 1\}$  and  $D_n^j : \{0, 1\}^{j+1} \rightarrow \{0, \dots, |G_n| - 1\}$  be two random functions, and  $C_n^0(\cdot) = u$ . We define sub-hybrid  $H_n^{j,k}((u, u_{j+1} \dots u_n), (x_1 \dots x_n))$  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  $(x_1 \dots x_j) \in L$ :
  - (a) Set  $y \leftarrow D_n^j(x_1 \dots x_{j+1})$ .
  - (b) Append  $(x_1 \dots x_j)$  to  $L$ .

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

<sup>11</sup> 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.

<sup>12</sup> 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  $g^y$ .

<sup>13</sup> 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<sup>14</sup>  $B^y$ .
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

<sup>14</sup> Recall that  $B = g^b$ , so  $B^y = g^{y \cdot b} = g^{y \cdot b \cdot \frac{x}{j^*+1}}$ . 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.  $\square$

## Exercises

**Exercise 3.1.** Prove or disprove: If  $f$  is a one-way function, then the following function  $B : \{0,1\}^* \rightarrow \{0,1\}$  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  $\pi = 3.141592653589 \dots$  after the decimal in binary ( $\pi$  in its binary notation looks like 11.00100100001111110110101010001000100001...).

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

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

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

**Exercise 3.3.** If  $f : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$  is PRF, then in which of the following cases is  $g : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$  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 ( $\text{Key}_F$ ,  $\text{Puncture}_F$ , and  $\text{Eval}_F$ ), satisfying the following conditions:

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

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

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

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

where  $K \leftarrow \text{Key}_F(1^n)$  and  $K_S = \text{Puncture}_F(K, x^*)$ .  $U_n$  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  $\Pi$  is a tuple  $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ , where  $\text{Gen}$ ,  $\text{Enc}$ , and  $\text{Dec}$  are algorithms such that:*

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

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  $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$  is (perfectly) correct if  $\forall n, k \in \text{Gen}(1^n), m \in \{0, 1\}^*$ ,*

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

That is, if  $c = \text{Enc}(k, m)$ , then Bob is guaranteed to recover  $m$  by running  $\text{Dec}(k, c)$ . 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*

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

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,  $\text{Enc}(k, \cdot)$ . 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.

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

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

*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<sup>1</sup>.

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

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

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  $\text{Dec}(k, \cdot)$ . Then  $\text{Priv-IND-CCA}_{\Pi}^{\mathcal{A}}(n)$  is defined as:

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

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

*is a negligible function.*

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

- 
- 1 :  $b \xleftarrow{\$} \{0, 1\}$
  - 2 :  $k \xleftarrow{\$} \text{Gen}(1^n)$
  - 3 :  $(\text{state}, m_0, m_1) \xleftarrow{\$} \mathcal{A}^{\text{Enc}(k, \cdot)}(1^n)$
  - 4 :  $c \xleftarrow{\$} \text{Enc}(k, m_b)$
  - 5 :  $b' \xleftarrow{\$} \mathcal{A}^{\text{Enc}(k, \cdot)}(\text{state}, c)$
  - 6 : **return**  $b = b' \wedge |m_0| = |m_1| = l(n)$

<sup>1</sup> 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.

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

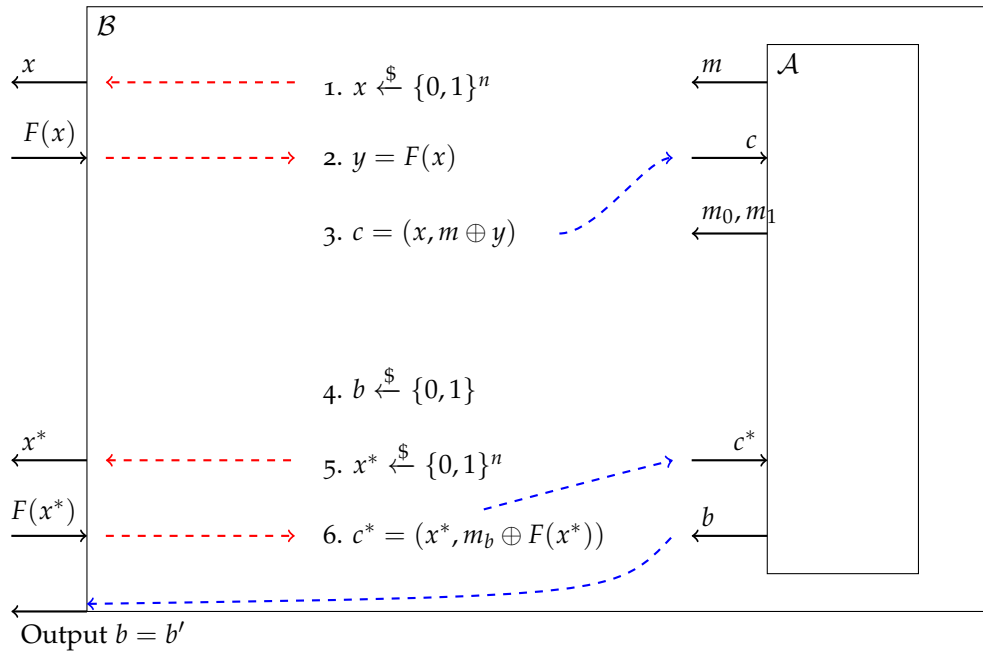
- 
- 1 :  $b \xleftarrow{\$} \{0, 1\}$
  - 2 :  $k \xleftarrow{\$} \text{Gen}(1^n)$
  - 3 :  $(\text{state}, m_0, m_1) \xleftarrow{\$} \mathcal{A}^{\text{Enc}(k, \cdot), \text{Dec}(k, \cdot)}(1^n)$
  - 4 :  $c \xleftarrow{\$} \text{Enc}(k, m_b)$
  - 5 :  $b' \xleftarrow{\$} \mathcal{A}^{\text{Enc}(k, \cdot), \text{Dec}(k, \cdot)}(\text{state}, c)$
  - 6 : **return**  $b = b' \wedge |m_0| = |m_1| \wedge c \notin L$

$\Pi$  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  $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$  given below is a secure encryption scheme for length  $n$ .*

- $\text{Gen}(1^n)$ :
  1. output  $k \xleftarrow{\$} \{0,1\}^n$
- $\text{Enc}(k, m)$ :
  1.  $r \xleftarrow{\$} \{0,1\}^n$
  2. output  $(r, F_k(r) \oplus m)$
- $\text{Dec}(k, c = (c_1, c_2))$ :
  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  $\Pi$ . Then we can construct a nu-PPT adversary  $\mathcal{B}$  that breaks the PRF  $F$ . The strategy is outlined in the figure below:



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

Now we argue that

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



is non-negligible.

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

□

**Theorem 4.2.** *No deterministic encryption scheme  $\Pi$  can be CPA Secure.*

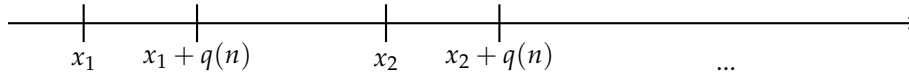
*Proof.* The proof of this claim is simple. If we have a deterministic encryption scheme, then when we get  $c^*$ , we can again try to encrypt a message and check if  $c = c^*$  □

#### 4.1.1 Counter Mode Encryption

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

- $\text{Enc}(k, (m_1, \dots, m_\ell)) :$ 
  - 1:  $r \xleftarrow{\$} \{0, 1\}^n$
  - 2: Output  $c = (r, m_1 \oplus F_k(r+1), m_2 \oplus F_k(r+2), \dots, m_\ell \oplus F_k(r+\ell))$

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  $\text{Verify}(k, m, t)$ . He accepts the message if  $\text{Verify}(k, m, t) = 1$ , otherwise he rejects the message. The formal definition is stated below:

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

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

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

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

**Definition 4.7** (MAC Correctness).

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

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  $\Pi = (\text{Gen}, \text{MAC}, \text{Verify})$  is EUF-CMA-secure if  $\forall$  nu-PPT  $\mathcal{A}$ ,

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

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

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  $t^{(i)} \leftarrow \text{MAC}(k, m^{(i)})$  and sends it to the adversary. The adversary may submit any polynomial number of message queries.  
Let  $\mathcal{Q} = \{(m^{(1)}, t^{(1)}), \dots, (m^{(q)}, t^{(q)})\}$  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  $\text{Verify}(k, m^*, t^*) = 1$ . 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  $\ell(n)$ .

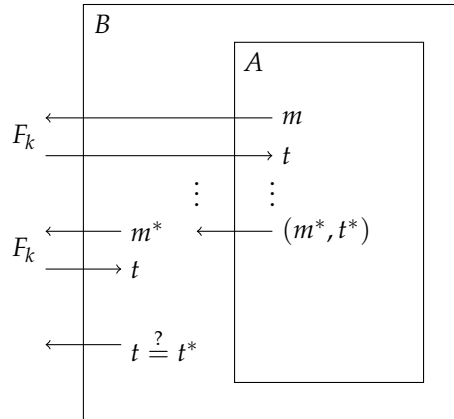
**Theorem 4.3.** If  $F : \{0,1\}^n \rightarrow \{0,1\}^n$  is a secure PRF, then the MAC scheme  $\Pi = (\text{Gen}, \text{Mac}, \text{Verify})$  constructed below has EUF-CMA security.

- $\text{Gen}(1^n) :$   
Output  $k \xleftarrow{\$} \{0,1\}^n$
- $\text{MAC}(k, m) :$   
Output  $t = F_k(m)$
- $\text{Verify}(k, m, t)$   
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  $\Pi$ . 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  $t = t^*$ .



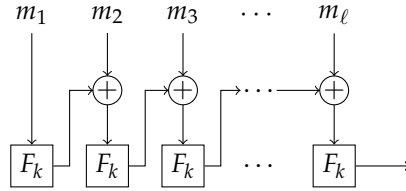
Analyzing the probability for  $B$ , we have

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

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.  $\square$

#### 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  $m_1, \dots, m_\ell$ , 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  $h : \mathcal{F} \times \mathcal{F}^* \rightarrow \mathcal{F}$  (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  $\ell$ ),

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

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:

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

**Claim 4.1.** The function defined by

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

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

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

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

**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.)*

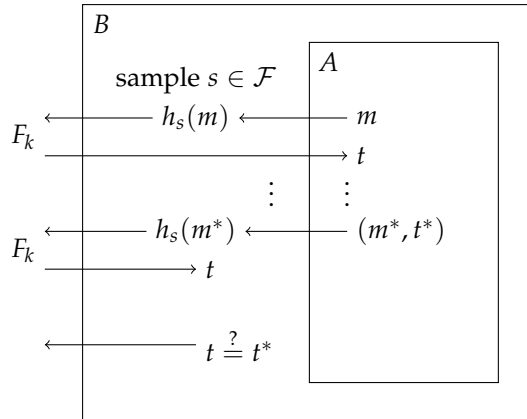
*Proof.* Suppose for contradiction that there exists a nu-PPT  $A$  that breaks the security of this scheme.

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.

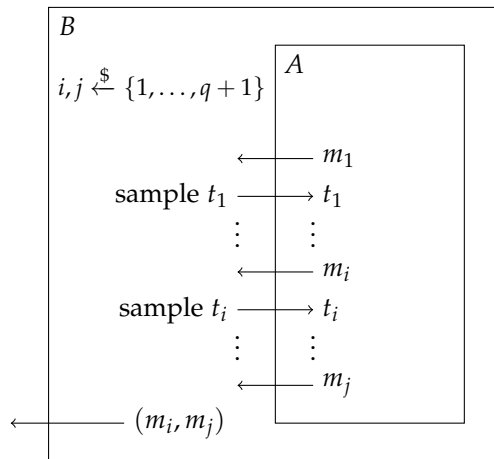


Let  $E$  be the event that there exists an  $m, m' \in L \cup \{m^*\}$ , such that  $h_s(m) = h_s(m')$ . If  $E$  does not happen, then the hash function never collides. This means that the attacker only sees random values depending on distinct inputs, so this reduces to the case from earlier (when 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  $i, j \in \{1, \dots, q+1\}$  (here suppose  $i < j$ ), where  $q$  is the number of MAC queries. We then run  $A$  until the  $j$ th query. Taking the  $i$ th and  $j$ th 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.  $\square$

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  $\Pi$  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  $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ .*

```

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

*A scheme has ciphertext integrity if for all nu-PPT  $A$ ,  $\Pr(\text{CI}_{\Pi}^A)$  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  $\Pi_e = (\text{Gen}_e, \text{Enc}_e, \text{Dec}_e)$  be a CPA-secure encryption scheme, and let  $\Pi_m = (\text{Gen}_m, \text{Mac}_m, \text{Verify}_m)$  be an EUF-CMA-secure MAC scheme.

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

```

1: function GEN( $1^n$ )
2:    $k_e \leftarrow \text{Gen}_e(1^n)$ 
3:    $k_m \leftarrow \text{Gen}_m(1^n)$ 
4:   return  $(k_e, k_m)$ 
5: end function

6: function ENC( $(k_e, k_m), m$ )
7:    $c \leftarrow \text{Enc}_e(k_e, m)$ 
8:    $t \leftarrow \text{Mac}_m(k_m, c)$ 
9:   return  $(c, t)$ 
10: end function

11: function DEC( $(k_e, k_m), (c, t), m$ )
12:   if  $\text{Verify}_m(k_m, c, t)$  then
13:     return  $\text{Dec}_e(k_e, c)$ 
14:   else
15:     return  $\perp$ 
16:   end if
17: end function

```

*Proof.* Suppose for contradiction that we have an adversary  $A$  that breaks the CPA security of  $\Pi$ . The CPA game allows for queries of the ciphertext for messages  $m$ , produces a pair  $m_0, m_1$ , and then gets  $c^* = \text{Enc}(k, m_B)$ , 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  $\Pi_e$ . The only difference here is the MACs, so  $B$  can sample a  $k_m \leftarrow \text{Gen}_m(1^n)$ , and perform all of the MACs itself.

In particular, when  $A$  asks for the ciphertext of  $M$ , we pass it to the oracle for  $\Pi_e$ , and attach  $t \leftarrow \text{Mac}_m(k_m, c)$ . 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  $\Pi_e$ .





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  $(\text{Gen}, \text{Sign}, \text{Verify})$  with the following syntax:

1.  $\text{Gen}(1^n) \rightarrow (vk, sk)$ : On input the message length (in unary)  $1^n$ ,  $\text{Gen}$  outputs a secret signing key  $sk$  and a public verification key  $vk$ .
2.  $\text{Sign}(sk, m) \rightarrow \sigma$ : On input a secret key  $sk$  and a message  $m$  of length  $n$ , the  $\text{Sign}$  algorithm outputs a signature  $\sigma$ .
3.  $\text{Verify}(vk, m, \sigma) \rightarrow \{0, 1\}$ : On input the verification key  $vk$ , a message  $m$  and a signature  $\sigma$ , the  $\text{Verify}$  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  $\forall m \in \{0, 1\}^n$ ,

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

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

1. The challenger first samples  $(vk, sk) \leftarrow \text{Gen}(1^n)$ . 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  $\sigma \leftarrow \text{Sign}(sk, m)$ .
3. **Forgery.** The adversary outputs a message, signature pair  $(m^*, \sigma^*)$  where  $m^*$  is different from the queries that adversary has made to the signing oracle.
4. The adversary wins the game if  $\text{Verify}(vk, m^*, \sigma^*) = 1$ .

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

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

- $\text{Gen}(1^n)$ : 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  $sk = \begin{bmatrix} x_{1,0} & \dots & x_{n,0} \\ x_{1,1} & \dots & x_{n,1} \end{bmatrix}$
- $\text{Sign}(sk, m)$ : On input a secret key  $sk$  and a message  $m \in \{0, 1\}^n$ , the Sign algorithm outputs a signature  $\sigma = x_{1,m_1} \| x_{2,m_2} \| \dots \| x_{n,m_n}$ .
- $\text{Verify}(vk, m, \sigma)$ : On input the verification key  $vk$ , a message  $m$  and a signature  $\sigma$ , 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  $\text{abort}_1$ . Otherwise, it uses its knowledge of  $x_{i,b}$  for  $(i, b) \neq (i^*, b^*)$  to output a signature on  $m$ .
6.  $\mathcal{A}$  outputs a valid forgery  $(m^*, \sigma^*)$ . If  $m_{i^*}^* = m_{i^*}$  then  $\mathcal{B}$  aborts and outputs a special symbol  $\text{abort}_2$ . If it does not abort, then it parses  $\sigma^*$  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  $\text{abort}_1$  or  $\text{abort}_2$ , the probability that  $\mathcal{B}$  outputs a valid preimage of  $y$  is  $\mu(n)$ . Now, probability  $\mathcal{B}$  does not output  $\text{abort}_1$  or  $\text{abort}_2$  is  $1/2n$  (this is because  $\text{abort}_1$  is not output with probability  $1/2$  and conditioned on not outputting  $\text{abort}_1$ ,  $\text{abort}_2$  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 will now upgrade this one-time signature scheme to a full digital signature scheme. For this purpose, we can use either universal one-way hash functions or collision-resistant hash functions.

### 5.3 Universal One-way Hash Function – UOWHF

We now introduce a universal one-way hash function (UOWHF). UOWHF is pronounced as “woof”. UOWHFs are stronger than universal hash functions but weaker than collision-resistant hash functions (CRHFs).

All three primitives (universal hash functions, UOWHFs, and CRHFs) guarantee that an attacker cannot find two colliding inputs,

but they differ in when the attacker must output their collision. Universal hash functions require the attacker to output both colliding inputs *before* seeing the hash function's key. Collision-resistant hash functions allow the attacker to output both colliding inputs *after* seeing the hash function's key. UOWHFs are in between: they require the attacker to output one colliding input before seeing the key and allow the attacker to output the second colliding input after seeing the key.

Universal hash functions can be constructed unconditionally, without computational assumptions. UOWHFs can be constructed from OWFs. Finally, the existence of collision-resistant hash functions is believed to be a stronger assumption than the existence of OWFs.

We now give a formal definition of universal one-way hash functions.

**Definition 5.1** (Universal One-way Hash Function (UOWHF)).

Let  $\ell: \mathbb{N} \rightarrow \mathbb{N}$ . A collection of functions  $\mathcal{H} = \{h_s : \{0, 1\}^* \rightarrow \{0, 1\}^{\ell(|s|)}\}_{s \in \{0, 1\}^*}$  is a universal one-way hash function if:

1. (Sampling):  $\exists$  PPT machine  $I$  that takes  $1^n$  and samples  $s$ .
2. (Easy to compute):  $\exists$  a deterministic machine  $M$  such that  $M(s, x) = h_s(x)$ .
3. (Hard to find collisions): Given a PPT attacker  $\mathcal{A}$ , the probability that  $\mathcal{A}$  wins the UOWHF game (given below) is negligible.

$$\Pr[\text{UOWHF}_{\mathcal{A}}^{\mathcal{H}}(n) = 1] = \text{negl}(n)$$

Where  $\text{UOWHF}_{\mathcal{A}}^{\mathcal{H}}$  is the following game:

1.  $(\text{state}, x) \leftarrow \mathcal{A}(1^n)$ .
2.  $s \leftarrow I(1^n)$ .
3.  $y \leftarrow \mathcal{A}(\text{state}, s)$
4. Output 1 if  $x \neq y$  and  $h_s(x) = h_s(y)$ . Output 0 otherwise.

### 5.3.1 Construction

We construct a UOWHF from a one-way permutation in several steps, starting with a length-restricted notion, called a  $(d, r)$ -UOWHF, and relaxing the restriction gradually.

**Definition 5.2** ( $(d, r)$ -UOWHFs). Let  $d, r: \mathbb{N} \rightarrow \mathbb{N}$  such that  $d(n) > r(n)$  for all  $n \in \mathbb{N}$ . A  $(d, r)$ -UOWHF is a collection of functions  $\mathcal{H} = \{h_s : \{0, 1\}^{d(|s|)} \rightarrow \{0, 1\}^{r(|s|)}\}_{s \in \{0, 1\}^*}$  that satisfies the conditions of a UOWHF.

The construction of a UOWHF from a OWP proceeds in the following 4 steps.

**Step I:  $(d, d - 1)$ -UOWHFs.** We construct a UOWHF that takes an input of arbitrary length and truncates its length by 1.

**Construction 5.1.** Let  $f: \{0, 1\}^d \rightarrow \{0, 1\}^d$  be a one-way permutation, and  $a, b \in GF(2^d)$ . We construct  $\mathcal{H}^{d, d-1} = \{h_s : \{0, 1\}^d \rightarrow \{0, 1\}^{d-1}\}_{s \in \{0, 1\}^*}$  as follows:

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

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

**Claim 5.1.** The  $\mathcal{H}^{d, d-1}$  given in construction 5.1 is a  $(d, d - 1)$ -UOWHF.

*Proof.* We prove this by contradiction. If an adversary  $\mathcal{A}$  can break the UOWHF security of  $\mathcal{H}^{d, d-1}$ , then we can construct an adversary  $\mathcal{B}$  that breaks the OWP security of  $f$ .

1.  $\mathcal{B}$  receives from its challenger a value  $y = f(x^*)$  for a uniformly random  $x^* \in \{0, 1\}^d$ .
2.  $\mathcal{B}$  runs  $\mathcal{A}$  internally.  $\mathcal{A}$  outputs the first preimage  $x_0$ .
3.  $\mathcal{B}$  computes a value of  $s = (a, b)$  such that

$$a \cdot y + b = a \cdot f(x_0) + b + (1||0^{d-1})$$

In other words  $h_s(x^*) = h_s(x_0)$ . Then  $\mathcal{B}$  sends  $s$  to  $\mathcal{A}$ .

4. Finally,  $\mathcal{A}$  outputs  $x_1$ , which  $\mathcal{B}$  outputs as its guess for  $x^*$ .

Every output of  $h$  has exactly two pre-images. This is because the mapping  $x \rightarrow a \cdot f(x) + b$  is one-to-one (as long as  $a \neq 0$ ), and the mapping  $y \rightarrow chop(y)$  is two-to-one.

The only two preimages of  $h_s(x^*)$  are  $x^*$  and  $x_0$  (assuming that  $x^* \neq x_0$ , which is true with overwhelming probability). In order to break UOWHF security,  $\mathcal{A}$  must output a preimage  $x_1$  of  $h_s(x_0)$  that is different from  $x_0$ . The only correct answer that  $\mathcal{A}$  can give is  $x_1 = x^*$ . If  $x_1 = x^*$ , then  $\mathcal{B}$  wins the OWP security game.  $\square$

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

**Construction 5.2.** For any given  $d \in \{n, \dots, 2n\}$ , let  $\mathcal{H}^{d, d-1} = \{h_s^{d, d-1} : \{0, 1\}^d \rightarrow \{0, 1\}^{d-1}\}_{s \in \{0, 1\}^*}$  be a  $(d, d - 1)$ -UOWHF. For a given  $n$ , we construct  $\mathcal{H}^{2n, n} = \{H_{s_1 \dots s_n}^{2n, n} : \{0, 1\}^{2n} \rightarrow \{0, 1\}^n\}_{s \in \{0, 1\}^*}$  as follows:

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

**Claim 5.2.** *The  $\mathcal{H}^{2n,n}$  constructed in construction 5.2 is a  $(2n, n)$ -UOWHF.*

*Proof.* We will show that if we have an attacker  $\mathcal{A}$  that can break the UOWHF security of  $H$ , then we can construct an attacker  $\mathcal{B}$  that can break the UOWHF security of  $h$ . The intuition is that the attacker  $\mathcal{B}$  can randomly inject the challenge  $s$  into a random layer of  $H$ , and  $\mathcal{A}$  will, with non-negligible probability, find a collision in that layer.

Let  $H_{\text{partial}}^i$  compute the first  $i$  layers of  $H$ . For any given  $i \in [n]$  and  $x \in \{0, 1\}^{2n}$ :

$$H_{\text{partial}}^i(x) = h_{s_{n-i+1}}^{2n-i+1, 2n-i}(\dots h_{s_n}^{2n, 2n-1}(x) \dots)$$

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

1. Sample  $i \xleftarrow{\$} [n]$ .
2.  $\forall i' \neq i$ , sample  $s_{i'}$ .
3. Receive  $x_0$  from  $\mathcal{A}$ .
4. Compute  $x'_0 = H_{\text{partial}}^{i-1}(x_0)$ , which is the input of  $i$ -th layer.
5. Output  $x'_0$  to the challenger.
6. Receive  $s$  from the challenger, and set  $s_i = s$ . Then send  $(s_1, \dots, s_n)$  to  $\mathcal{A}$ .
7. Receive  $x_1$  from  $\mathcal{A}$ .
8. Compute  $x'_1 = H_{\text{partial}}^{i-1}(x_1)$  and output it to the challenger.

If  $\mathcal{A}$  can find two distinct inputs  $x_0$  and  $x_1$  that collide in  $H$ , then there must exist some layer  $i^* \in [n]$  such that  $H_{\text{partial}}^{i^*-1}(x_0) \neq H_{\text{partial}}^{i^*-1}(x_1)$ , but  $H_{\text{partial}}^{i^*}(x_0) = H_{\text{partial}}^{i^*}(x_1)$ . That means  $H_{\text{partial}}^{i^*-1}(x_0)$  and  $H_{\text{partial}}^{i^*-1}(x_1)$  represent a collision in  $h_{s_{n-i^*+1}}^{2n-i^*+1, 2n-i^*}$ , the  $i^*$ -th layer of  $H$ .

Furthermore, the index  $i^*$  at which the collision occurs will, with probability  $= 1/n$ , be the  $i$ -value sampled by  $\mathcal{B}$ . This is because  $\mathcal{A}$  has no information about which  $i$ -value  $\mathcal{B}$  sampled, so  $i$  is uniformly random and independent of  $i^*$ .

Then  $\Pr[\mathcal{B} \text{ finds a collision in } h] \geq \frac{1}{n} \cdot \Pr[\mathcal{A} \text{ finds a collision in } H]$ . If  $\mathcal{A}$ 's success probability is non-negligible, then so is  $\mathcal{B}$ 's success probability.  $\square$

**Step III: UOWHF that shrinks any input by a factor of two.** Let us define a  $(2*, *)$ -UOWHF to be a hash function that takes inputs  $x$  of arbitrary length, outputs a string that is half the length of the input, and satisfies the conditions of a UOWHF.



**Construction 5.3** (a  $(2^*, *)$ -UOWHF). For a given  $n$ , let  $\mathcal{H}^{2n,n} = \{h_s : \{0,1\}^{2n} \rightarrow \{0,1\}^n\}_{s \in \{0,1\}^*}$  be a  $(2n, n)$ -UOWHF. Then we construct a function  $H_s : \{0,1\}^{2^*} \rightarrow \{0,1\}^*$  as follows.

For any  $x \in \{0,1\}^*$ , let us split it into chunks:  $x = x_1 || \dots || x_t$  where  $|x_i| = 2n$  for  $i = 1, \dots, t-1$ , and  $0 \leq |x_t| \leq 2n$ . Then

$$\text{let } H_s(x) = h_s(x_1) \cdots h_s(x_t 10^{2n-|x_t|-1})$$

Finally, let  $\mathcal{H}^{2^*,*} = \{H_s\}_{s \in \{0,1\}^*}$ .

Note that we reuse the same  $s$  for every component of the output  $h_s(x_i)$ , which is an important gain in efficiency.

**Claim 5.3.**  $\mathcal{H}^{2^*,*}$  is a  $(2^*, *)$ -UOWHF.

*Proof.* If there is an adversary  $\mathcal{A}$  that can create a collision,  $x^0$  and  $x^1$ , in  $H$ , then we can construct an adversary  $\mathcal{B}$  that can create a collision in  $h$ . The construction of  $\mathcal{B}$  is as follows:

1. Run  $\mathcal{A}$  until it outputs  $x^0$ . Compute  $t = \lceil \frac{|x^0|}{2n} \rceil$ .
2. Sample  $i \xleftarrow{\$} [t]$ . Then output  $x'^0 = x_i^0$  to the challenger.
3. Receive seed  $s$  from the challenger and send  $s$  to  $\mathcal{A}$ .
4. Run  $\mathcal{A}$  until it outputs  $x^1$ . Then output  $x'^1 = x_i^1$ .

If  $\mathcal{A}$  breaks the UOWHF security of  $H$ , then with non-negligible probability, it outputs an  $x^0$  and  $x^1$  such that  $x^0 \neq x^1$  but  $h_s(x_{i'}^0) = h_s(x_{i'}^1)$  for every  $i' \in [t]$ . If  $x^0 \neq x^1$ , then there is at least one component  $i^* \in [t]$  for which  $x_{i^*}^0 \neq x_{i^*}^1$ . With probability  $\geq \frac{1}{t}$ ,  $\mathcal{B}$  will sample an  $i \in [t]$  such that  $x_i^0 \neq x_i^1$ .  $\square$

#### Step IV: Full-Fledged UOWHFs.

**Construction 5.4** ((a UOWHF)). Let  $\mathcal{H}^{2^*,*} = \{h_s : \{0,1\}^* \rightarrow \{0,1\}^*\}_{s \in \{0,1\}^*}$  be a  $(2^*, *)$ -UOWHF. Then for any  $(s_1, \dots, s_n) \in \{0,1\}^*$ , any  $n, t \in \mathbb{N}$  and  $x \in \{0,1\}^{2^{t \cdot n}}$ , we define

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

Also let  $\mathcal{H} = \{H_s\}_{s \in \{0,1\}^*}$ .

**Claim 5.4.**  $\mathcal{H}$  is a secure UOWHF.

The proof is similar to the one 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.

### 5.3.2 Removing Length-Restriction from One-Time Digital Signatures

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

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

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  $I(1^n)$ :

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

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  $(\text{Gen}, \text{Sign}, \text{Verify})$ . We will now construct a nu-PPT adversary  $\mathcal{B}$  who succeeds with non-negligible probability at the digital signature security game for  $(\text{Gen}_\ell, \text{Sign}_\ell, \text{Verify}_\ell)$ , a contradiction.  $\mathcal{B}$  receives  $pk_\ell$  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  $m_\ell$  to receive  $\sigma_\ell$  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  $\Pr[s^* \neq s \wedge \mathcal{A} \text{ succeeds}]$  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  $\Pr[s^* = s \wedge \mathcal{A} \text{ succeeds}]$  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  $(\text{Gen}_\ell, \text{Sign}_\ell, \text{Verify}_\ell)$  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  $(\text{Gen}, \text{Sign}, \text{Verify})$  into a ef-ema no-length restriction digital signature scheme by utilizing a pseudorandom function  $\text{PRF}$ . For each  $\alpha \in \{\epsilon\} \cup \{0, 1\}^{\leq n}$ , let  $pk_\alpha, sk_\alpha = \text{Gen}(1^n; \text{PRF}_s(\alpha 10 \dots 0))$  such that  $|\alpha 10 \dots 0| = n + 1$  (i.e.  $\text{Gen}$  is run with randomness determined by the PRF on an input specified by  $\alpha$ ). 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  $\text{Sign}$  is called, we require the WOOF to be run with a deterministic seed  $s_\alpha^W = \text{PRF}_{s'}(\alpha 10 \dots 0)$ . This way paths through the tree will deterministically map to the corresponding signatures. The construction is as follows:

- $\text{GEN}(1^n)$ : 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.
- $\text{SIGN}(sk, m)$ :
  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  $i = 0, 1, \dots, n - 1$ , let  $\alpha_i = r_1 r_2 \dots r_i$ ,  $m_i = pk_{\alpha_i || 0} || pk_{\alpha_i || 1}$ , and  $\sigma_i = \text{Sign}(sk_{\alpha_i}, m_i)$ .
  3. Let  $\sigma_n = \text{Sign}(sk_r, m)$ .
  4. Output  $\Sigma = (r, m_0, \sigma_0, \dots, m_{n-1}, \sigma_{n-1}, \sigma_n)$ .
- $\text{VERIFY}(pk, m, \Sigma = (r, \sigma_0, \dots, \sigma_{n-1}, \sigma_n))$ : Use  $r$  and the  $m_i$  to determine  $pk_{\alpha_i}$  and for each  $i \in [n]$  run  $\text{Verify}(pk_{\alpha_i}, m_i, \sigma_i)$ , accepting if all of those do.

The idea is that because the root  $pk_\epsilon$  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 (GEN, SIGN, 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  $(M^*, \Sigma^*)$  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  $\Sigma_j$ 's along the path  $r^* = r_j$  until doing a forgery at the leaf, breaking the (Gen, Sign, 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 (Gen, Sign, Verify) scheme for the corresponding  $\alpha_i^*$  which is distinct from all of the other  $\alpha_i^j$  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 (Gen, Sign, 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

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

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

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

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

### 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.

- $\text{Sampler}(1^n)$  : 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  $(g, h)$ .
- $\text{Eval}((g, h), (r, s))$  : 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.<sup>1</sup>

<sup>1</sup> 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.

### 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  $\ell$  and the root is associated with the empty string. We will assign a tag  $\in \{0, 1\}^n$  to each node in the tree. The  $i$ -th leaf is assigned  $\text{tag}_i$  equal to the  $i$ -block of the string  $x$ . Each intermediate node  $y$  is assigned a  $\text{tag}_y = h(\text{tag}_{y||0} || \text{tag}_{y||1})$ . 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 exists one intermediate node  $y$  such that for two different values  $\text{tag}_{y||0}, \text{tag}_{y||1}$  and  $\text{tag}'_{y||0}, \text{tag}'_{y||1}$  we have,  $h(\text{tag}_{y||0}, \text{tag}_{y||1}) = \text{tag}'_{y||0}, \text{tag}'_{y||1}$ . 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-

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  $(\text{Gen}', \text{Sign}', \text{Verify}')$  be a one-time signature scheme for signing longer messages.

1.  $\text{Gen}(1^n)$  : Run  $\text{Gen}'(1^n)$  using to obtain  $sk, vk$ . Sample a PRF key  $K$ . The signing key is  $(sk, K)$  and the verification key is  $vk$ .
2.  $\text{Sign}((sk, K), m)$  : 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  $\epsilon$  is the empty string).
  - (c) For each  $i \in [\ell]$  do:
    - i. Evaluate  $\text{PRF}(m_1 \parallel \dots \parallel m_{i-1} \parallel 0)$  and  $\text{PRF}(m_1 \parallel \dots \parallel m_{i-1} \parallel 1)$  to obtain  $r_0$  and  $r_1$  respectively. Run  $\text{Gen}'(1^n)$  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  $\sigma_i = \text{Sign}(sk_{i-1, m_{i-1}}, vk_{i,0} \parallel vk_{i,1})$
    - iii. If  $i = \ell$ , then set  $\sigma_{\ell+1} = \text{Sign}(sk_{i, m_i}, m)$ .
  - (d) Output  $\sigma = (\sigma_1, \dots, \sigma_{\ell+1})$  along with all the verification keys as the signature.
3.  $\text{Verify}(vk, \sigma, m)$ : Check if all the signatures in  $\sigma$  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  $\{f_s : D_s \rightarrow D_s\}_{s \in \{0,1\}^*}$  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:

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

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  $\Pi^H = (A^H, B^H, C^H, \dots)$ , 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  $\Pi^O = (A^O, B^O, C^O, \dots)$ , 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  $\Pi^O$  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  $\Pi^O$ , 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.

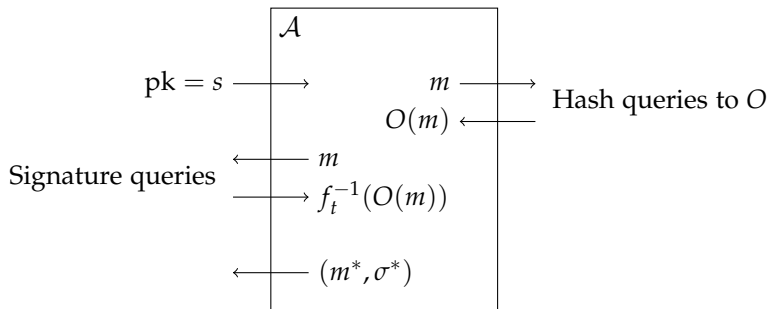
Note here that the adversary  $\mathcal{A}$  in this case is forced to explicitly call the oracle for its hash function queries—the fact that we can see these calls is called *observability*. In the standard model, we can't actually see the queries that the adversary makes, since it just runs the predefined 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  $\text{pk} = s$ . 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  $(m^*, \sigma^*)$ . 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  $(m^*, \sigma^*)$ , 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  $\text{pk}$ .  $\mathcal{B}$  first samples an  $i^* \leftarrow \{1, \dots, q_h\}$ . We then set the output of the  $i^{\text{th}}$  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  $i^{\text{th}}$  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.  $\square$

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  $H : \{0, 1\}^* \rightarrow \mathbb{Z}_q$ , we define

- $\text{GEN}(1^n) = (\text{pk} = g^x, \text{sk} = x \leftarrow \mathbb{Z}_q)$
- $\text{SIGN}(\text{sk}, m)$ :
  - $k \leftarrow \mathbb{Z}_q$
  - $r = g^k$
  - $h = H(m \| r)$

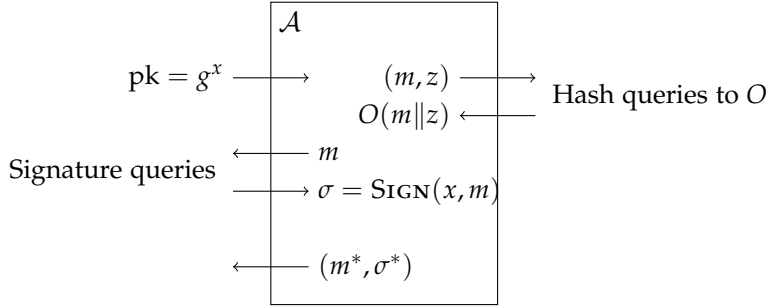
$$s = k + hx$$

$$\sigma = (h, s)$$

- $\text{VERIFY}(\text{pk}, m, \sigma)$ :  
output  $h \stackrel{?}{=} H(m \| \frac{g^s}{\text{pk}^h})$

**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  $\text{pk} = g^x$ , can make signing queries  $m \mapsto \text{SIGN}(x, m)$  and hashing queries  $(m, z) \mapsto O(m \| z)$ , for  $z \in G$ .  $\mathcal{A}$  then returns a forgery  $(m^*, \sigma^*)$ .



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

We'll define a modified signing algorithm as follows:

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

The main idea here is to provide a random signature, consistent with the definition of  $\text{SIGN}$ , so that  $\text{VERIFY}$  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  $(m^*, \sigma^*)$ . In particular,  $\mathcal{A}'$  depends on  $\text{pk}, q_1, h_1, \dots, q_H, h_H$ , but all of the queries  $q_1, \dots, q_H$  are deterministic depending on the previous hash output (or dependent on  $\text{pk}$  in the case of  $q_1$ ). This means that  $\mathcal{A}'$  can actually be thought of as a function of

$$\mathcal{A}'(\text{pk}, h_1, h_2, \dots, h_H).$$

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  $q_{i^*}, q_{i^*+1}, \dots, q_h$ , and outputted  $(m^*, \sigma^*)$ . In the second execution we would have gotten queries  $q'_{i^*}, q'_{i^*+1}, \dots, q'_h$ , and outputted  $(m'^*, \sigma'^*)$ .

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  $\alpha = (\text{pk}, h_1, \dots, h_{i^*-1})$  and  $\beta = (h_{i^*}, \dots, h_{q_h})$ . 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

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

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.  $\square$

### 5.9 BLS Signatures

*Bilinear Pairings.* Recall that we used prime order groups  $G$  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  $G_1, G_2$  and  $G_T$  be groups of prime order  $p$ . We define a bilinear pairing  $e : G_1 \times G_2 \rightarrow G_T$  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:**  $e(g_1, g_2) \neq 1_{G_T}$ .

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  $G_1, G_2$  be groups of prime order with generators  $g_1$  and  $g_2$  and let  $e : G_1 \times G_2 \rightarrow G_T$  be an efficiently computable bilinear pairing. The co-CDH assumption states that for all non-uniform PPT adversaries  $\mathcal{A}$ :

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

*BLS Signature scheme.* We are now ready to describe the BLS signature scheme <sup>2</sup>.

- **Gen( $1^\lambda$ ):** Sample  $sk \leftarrow \mathbb{Z}_p$  and set  $vk \leftarrow g_2^{sk}$ . Output  $(vk, sk)$ .
- **Sign( $sk, m$ ):** Given a message  $m \in \{0, 1\}^*$ , output  $\sigma \leftarrow H(m)^{sk}$ , where  $H : \{0, 1\}^* \rightarrow G_1$  is a hash function that maps arbitrary strings to elements in  $G_1$ .
- **Verify( $vk, m, \sigma$ ):** Output  $e(\sigma, g_2) \stackrel{?}{=} e(H(m), vk)$ .

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

*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 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  $(m^*, \sigma^*)$ , where  $m^*$  has not been previously queried.
- At the start of the protocol,  $\mathcal{B}$  sets  $\text{vk} = g_1^b$  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  $\sigma_{i^*}$  as its output in the co-CDH game.

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

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

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  $(\text{Gen}, \text{Sign}, \text{Verify})$  for which  $\text{Sign}$  is probabilistic, provide a construction of a digital signature scheme  $(\text{Gen}', \text{Sign}', \text{Verify}')$  where  $\text{Sign}'$  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<sup>1</sup> 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  $(\text{Gen}, \text{Enc}, \text{Dec})$  defined as follows:

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

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

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

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.

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

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

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

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

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

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

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

- $(\text{pk}, \text{sk}) \leftarrow \text{Gen}(1^n)$ .
- $b \leftarrow \$ \{0, 1\}$ .
- $\mathcal{A}^{\text{Dec}(\text{sk})}(\text{pk}) \rightarrow (m_0, m_1, \text{st})$ .
- $c^* \leftarrow \text{Enc}(\text{pk}, m_b)$ .
- $\mathcal{A}^{\text{Dec}(\text{sk})}(c^*, \text{st}) \rightarrow b'$ , 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  $(\text{Gen}, f, f^{-1}, D)$  defined as follows:

- $\text{Gen}(1^n) \rightarrow (s, t)$ : outputs an index  $s$  and corresponding trapdoor  $t$ .
- $D(s)$ : Outputs a description of the domain of the trapdoor function.
- $f(s, x) \rightarrow y$ : Takes as input an index  $s$  and a element  $x$  from the domain of the trapdoor function and outputs an element  $y$ .
- $f^{-1}(t, y) \rightarrow x$ : 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  $n \in \mathbb{N}$ ,  $(s, t) \leftarrow \text{Gen}(1^n)$ ,  $x \in D(s)$ , it holds that  $\Pr[f^{-1}(t, f(s, x)) \neq x] = \text{negl}(n)$ . 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  $\Pr[\mathcal{A}(s, y) \rightarrow x \mid (s, t) \leftarrow \text{Gen}(1^n), x \leftarrow D(s), y \leftarrow f(s, x)] = \text{negl}(n)$ .

## 6.3 Public Key Encryption from Trapdoor Functions

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

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

*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 <sup>2</sup> 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  $G$  be a prime order group of order  $p$  with generator  $g$ , where the CDH problem is assumed to be hard.

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

*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  $O(\text{poly}(n))$ . Encrypting a message with  $M$  bits, results in a ciphertext of size  $O(M \cdot \text{poly}(n))$ . 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:

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

- $\text{Dec}(\text{sk}, c)$ : Parse  $c = (y, z)$  and compute  $x \leftarrow f^{-1}(t, y)$ . Decrypt  $z$  using the symmetric encryption scheme with key  $H(x)$  as  $m \leftarrow z \oplus \text{Sym.Dec}(H(x), z)$ . 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  $\text{pk} := s$ .  $\mathcal{A}$  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  $z^* \leftarrow \text{Sym.Enc}(k^*, m_b)$ .  $\mathcal{B}$  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  $k^*$  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.  $\square$

## 6.6 Fujisaki-Okamoto Transformation

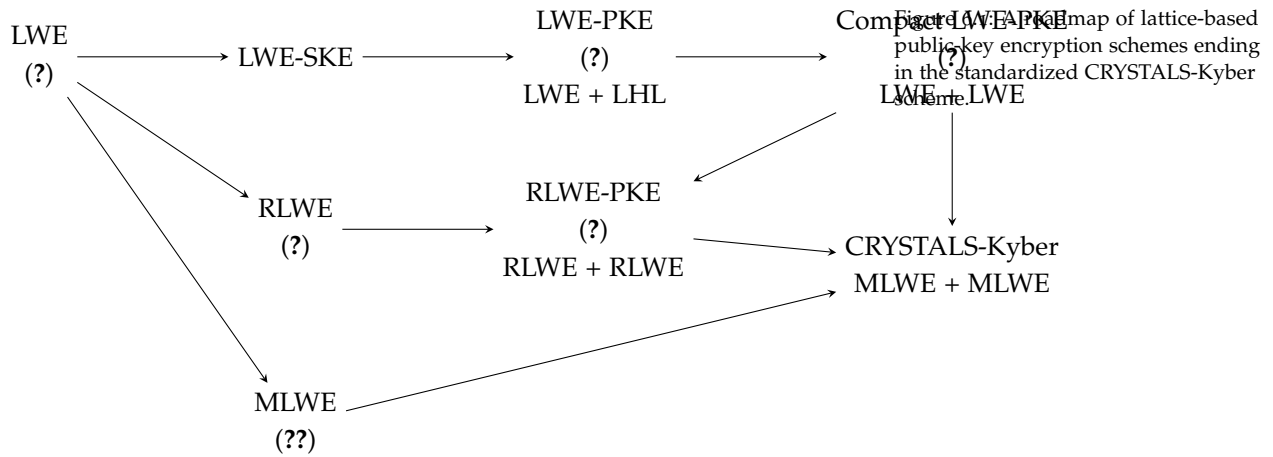
## 6.7 Post-Quantum Public Key Encryption

The public-key encryption schemes discussed so far, rely on the difficulty of problems like factorization or discrete logarithms. In 1994, Peter Shor showed that these problems can be solved efficiently on a quantum computer<sup>3</sup>. Even though we do not have large-scale quantum computers capable of breaking current encryption schemes, there are two reasons to begin the transition of public-key encryption to quantum-resistant schemes:

- Encrypted messages captured today can be stored and decrypted in the future when a large scale quantum computer is available. This is commonly referred to as the "harvest now, decrypt later" risk.

- Transition to new encryption schemes is a slow process and it is important to start the transition well before large scale quantum computers are available.

The national institute of standards and technology (NIST) opened a call for post-quantum public-key encryption and signature schemes in 2016. In Nov 2019, it received 59 submissions for public-key encryption and 23 submissions for digital signatures. In July 2022, NIST announced the first batch of winners for public-key encryption and digital signatures. In Aug 2024, the final standard for these schemes was published and they are now making their way into existing infrastructure. There was one winner (CRYSTALS-Kyber) for public-key encryption (based on lattices) and three winners for digital signatures (two based on lattices and one based on hash functions).



**Definition 6.6** (Learning With Errors Assumption (Search)). *Let  $m, n, q \in \mathbb{N}$  and  $\chi$  be a distribution over  $\mathbb{Z}_q$ . The Learning With Errors (LWE)  $\text{LWE}_{n,m,q,\chi}$  problem is defined as follows:*

$$\Pr[\mathcal{A}(A, b) \rightarrow s \mid s \leftarrow \mathbb{Z}_q^n, A \leftarrow \mathbb{Z}_q^{m \times n}, b = A \cdot s + e] \leq \text{negl}(n)$$

**Definition 6.7** (Learning With Errors (Decision)). *Let  $m, n, q \in \mathbb{N}$  and  $\chi$  be a distribution over  $\mathbb{Z}_q$ . The Learning With Errors (LWE)  $\text{LWE}_{n,m,q,\chi}$  problem is defined as follows:*

$$\begin{aligned} & \left| \Pr[\mathcal{A}(A, b) \rightarrow 1 \mid s \leftarrow \mathbb{Z}_q^n, A \leftarrow \mathbb{Z}_q^{m \times n}, b = A \cdot s + e] \right. \\ & \quad \left. - \Pr[\mathcal{A}(A, b) \rightarrow 1 \mid s \leftarrow \mathbb{Z}_q^n, A \leftarrow \mathbb{Z}_q^{m \times n}, b \leftarrow \mathbb{Z}_q^m] \right| \leq \text{negl}(n) \end{aligned}$$

The Learning With Errors assumption is commonly referred to

as a Lattice based assumption because there is a reduction from Search/Decision LWE to a “worst-case” lattice problem.

The above assumptions have been stated with respect to some abstract distribution  $\chi$  over  $\mathbb{Z}_q$ . But what do we actually choose this distribution to be? In the extreme case of  $\chi = 0$ , the LWE problem is trivial as one can simply use Gaussian Elimination. In the other extreme if  $\chi$  is uniform over  $\mathbb{Z}_q$ , then the LWE problem is information theoretically hard (but not very useful for cryptography). We will be interested in the intermediate case where  $\chi$  is a *small* distribution over  $\mathbb{Z}_q$ , centered around 0. For eg:  $\chi$  is a uniform distribution over  $[-B, B]$  for some  $B \ll q/2$ . This will allow us to do build interesting cryptographic primitives like public key encryption and signature schemes. For provable reductions to lattice problems, we set  $\text{stddev}(\chi) = \Omega(\sqrt{n})$ . However, there is a gap between the best known attacks on LWE and the best known reductions to lattice problems. As a result, much more aggressive parameters are used in practice, chosen based on the best known attacks. Typical parameters for LWE are  $n = 512$ ,  $q = 3329$ ,  $\text{supp}(\chi) = [-3, 3]$ , and  $m = 768$ .<sup>4</sup> provides a lattice estimator <https://github.com/malb/lattice-estimator> that can be used to estimate the number of bits of security provided by a given set of LWE parameters.

### 6.7.1 LWE $\rightarrow$ LWE-SKE

As a first step, we will see how to build a symmetric key encryption scheme from LWE.

- Gen: Sample  $s \leftarrow \$ \mathbb{Z}_q^n$  and output  $\text{sk} = s$ .
- Enc( $\text{sk}, \mu$ ): Sample  $a \leftarrow \$ \mathbb{Z}_q^n$ ,  $e \leftarrow \$ \chi$ , and compute  $b \leftarrow \langle a, s \rangle + e$ . Output  $c = (c_0 = a, c_1 = b + \lfloor \frac{q}{2} \rfloor \mu)$ .
- Dec( $\text{sk}, c$ ): Parse  $c = (c_0, c_1)$  and compute  $m \leftarrow \text{Decode}(c_1 - \langle c_0, s \rangle)$ , where  $\text{Decode}(\hat{\mu}) \rightarrow \{0, 1\}$  takes a value from  $\mathbb{Z}_q$  and outputs 0 if  $\hat{\mu} \in [\lfloor \frac{-q}{4} \rfloor, \lfloor \frac{q}{4} \rfloor]$  and 1 otherwise.

**Theorem 6.2.** *If the decisional variant of  $\text{LWE}_{n,1,q,\chi}$  is hard, then the above scheme is an IND-CPA secure symmetric key encryption scheme.*

*Proof.* Let the ciphertext  $c = (c_0, c_1)$ . Then  $\hat{\mu} = c_1 - \langle c_0, s \rangle = b + \lfloor \frac{q}{2} \rfloor \mu - \langle a, s \rangle = \lfloor \frac{q}{2} \rfloor \mu + e$ . Upon applying the Decode function, we recover the message  $\mu$  if  $|e| < \lfloor \frac{q}{4} \rfloor$ . One can choose the noise distribution  $\chi$  such that  $\Pr[|e| \geq \lfloor \frac{q}{4} \rfloor] \leq \text{negl}(n)$ , to obtain a symmetric key encryption scheme with a small correctness error. Alternatively, one can also set  $\chi$  such that  $\Pr[|e| \geq \lfloor \frac{q}{4} \rfloor] = 0$  to obtain a perfectly correct symmetric key encryption scheme. Security reduces to hardness of the decisional  $\text{LWE}_{n,1,q,\chi}$  problem.  $\square$

### 6.7.2 How to build a public key encryption scheme from LWE?

Our first observation is that LWE samples offer a limited amount of homomorphism. Given LWE samples  $(a_1, b_1 = \langle a_1, s \rangle + e_1)$  and  $(a_2, b_2 = \langle a_2, s \rangle + e_2)$ , one can *add* them up to get  $(a_1 + a_2, b_1 + b_2 = \langle a_1 + a_2, s \rangle + (e_1 + e_2))$ . Observe that as long as  $e_1 + e_2$  is small, we can still build useful cryptography as done in the symmetric key encryption scheme above.

Now to build public key encryption, we *delegate* the power to create LWE samples, without revealing the secret  $s$ . We achieve this by setting the public key to be many symmetric key encryptions of 0. The scheme below was originally proposed by Regev in 2005 <sup>5</sup>.

- Gen: Sample  $s \leftarrow \mathbb{Z}_q^n$ ,  $A \leftarrow \mathbb{Z}_q^{m \times n}$ ,  $e \leftarrow \chi^m$ ,  $b \leftarrow As + e$  and output  $(pk = (A, b), sk = s)$ .
- Enc( $pk, \mu$ ): Sample  $u \leftarrow \{0, 1\}^m$ . Output  $c = (c_0 = A^T u, c_1 = b^T u + \lfloor \frac{q}{2} \rfloor \mu)$ .
- Dec( $sk, c$ ): Output Decode( $c_1 - \langle c_0, s \rangle$ ).

Observe that  $b$  is actually just  $m$  encryptions of 0. Moreover,  $m$  influences the size of the public key and we would like to make this as small as possible. The exact parameter will be chosen based on what we can actually prove security for.

*Proof.* For correctness observe that as before  $c_1 - \langle c_0, s \rangle = b^T u + \lfloor \frac{q}{2} \rfloor \mu - \langle A^T u, s \rangle = u^T e + \lfloor \frac{q}{2} \rfloor \mu$ . Decode will recover  $\mu$  if  $|u^T e| < \lfloor \frac{q}{4} \rfloor$ , and one can choose  $\chi$  to ensure this with high probability.

For security, we describe a sequence of hybrids starting from the public-key IND-CPA game, where the challenger receives  $(\mu_0, \mu_1)$ , and responds with an encryption of  $\mu_0$ . We then end in a hybrid where the challenger responds with an encryption of  $\mu_1$ , while showing that consecutive hybrids are indistinguishable.

- **Hybrid 0:** This is the original public-key IND-CPA game, where the challenger samples  $(pk, sk)$  as described in the protocol and receives  $(\mu_0, \mu_1)$  from  $\mathcal{A}$ . The challenger responds with an encryption of  $\mu_0$  and  $\mathcal{A}$  outputs a guess  $b'$ .
- **Hybrid 1:** This hybrid is identical to Hybrid 0, except that the challenger sets  $pk = (A \leftarrow \mathbb{Z}_q^{m \times n}, b \leftarrow \mathbb{Z}_q^m)$ . From the hardness of the decisional  $LWE_{n,m,q,\chi}$  problem, these two hybrids are computationally indistinguishable.
- **Hybrid 2:** In this hybrid we replace the ciphertext with a uniformly random string  $c \in \mathbb{Z}_p^n \times \mathbb{Z}_p$ . We now argue that this hybrid is statistically indistinguishable from the previous hybrid. Let



$u \leftarrow \$ \{0, 1\}^m$ , and  $\gamma \leftarrow \$ \mathbb{Z}_p^n$ . Then  $\mathbb{E}_{C \in \mathbb{Z}_q^{n \times m}} [\text{SD}(Cu, \gamma)] \leq \frac{1}{2} \sqrt{\frac{q^n}{2^m}}$ . If we set  $m = n \log q + 2\lambda$ , then the statistical distance is negligible in the security parameter.

To prove the above we will use the leftover hash lemma. Let  $p_c(w) = \Pr_{u \leftarrow \$ \{0, 1\}^m} [Cu = w]$ . Then,

$$\sum_{w \in \mathbb{Z}_q^n} p_c(w)^2 = \Pr_{u_0, u_1 \leftarrow \$ \{0, 1\}^m} [Cu_0 = Cu_1] = \frac{1}{2^m} + \mathbb{E}_C \left[ \Pr_{u_0 \neq u_1} [C(u_0 - u_1) = 0] \right] = \frac{1}{2^m} + \frac{1}{q^n}$$

Thus,  $\mathbb{E}_{C \in \mathbb{Z}_q^{n \times m}} [\text{SD}(Cu, \gamma)] = \frac{1}{2} \mathbb{E}_C [\sum_w |p_c(w) - \frac{1}{q^n}|] \leq \frac{1}{2} \sqrt{q^n} \mathbb{E}_C \left[ \sqrt{\sum_w (p_c(w) - \frac{1}{q^n})^2} \right]$ , where the inequality follows from Cauchy-Schwarz. Next, using Jensen's inequality we have

$$\frac{1}{2} \sqrt{q^n} \mathbb{E}_C \left[ \sqrt{\sum_w (p_c(w) - \frac{1}{q^n})^2} \right] \leq \frac{1}{2} \sqrt{q^n} \sqrt{\mathbb{E}_C \left[ \sum_w (p_c(w) - \frac{1}{q^n})^2 \right]} = \frac{1}{2} \sqrt{\frac{q^n}{2^m}}$$

The remaining hybrids are symmetric to end in the IND-CPA game with challenge response set to an encryption of  $\mu_1$ .  $\square$

### 6.7.3 Improving Efficiency

It is worthwhile asking whether the parameter  $m$  can be made smaller. Recall that when using the leftover hash lemma we needed to set  $m = n \log q + 2\lambda$  but we got a statistical guarantee of indistinguishability. This *overkill* in some sense because we still rely on the LWE assumption in a different hybrid. We can then ask the question of whether it's possible to improve efficiency and obtain *compact* LWE public-key encryption if we relied on computational hardness instead of a statistical guarantee.

This was the approach taken in <sup>6</sup>, where they used the hardness of LWE instead of the leftover hash lemma to switch ciphertexts to uniformly random strings in the proof of IND-CPA security. The scheme is essentially identical to the previous construction except, the secret key is also sampled from the error distribution instead of uniformly over the entire field. This is necessary for correctness. But we are no longer working with LWE samples as the secret key is no longer uniform. These new type of instance are referred to as Hermite normal form LWE instances. We will later show that distinguishing HNF-LWE samples from a uniform distribution is just as hard as distinguishing LWE samples from the uniform distribution.

<sup>6</sup>

**Definition 6.8** (HNF-LWE). Let  $m, n, q \in \mathbb{N}$  and  $\chi$  be a distribution over  $\mathbb{Z}_q$ . The Learning With Errors (LWE)  $\text{LWE}_{n,m,q,\chi}$  problem is defined as follows:

$$\begin{aligned} & |\Pr[\mathcal{A}(A, b) \rightarrow 1 \mid s \leftarrow \chi_q^n, A \leftarrow \mathbb{Z}_q^{m \times n}, b = A \cdot s + e] \\ & - \Pr[\mathcal{A}(A, b) \rightarrow 1 \mid A \leftarrow \mathbb{Z}_q^{m \times n}, b \leftarrow \mathbb{Z}_q^m]| \leq \text{negl}(n) \end{aligned}$$

**Lemma 6.1** ( $\text{LWE}_{n,q,\chi,n,m} \leq \text{HNF-LWE}_{n,q,\chi,m-n}$ ).

*Proof.* Parse  $A$  and  $b$  as:

$$A : \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \quad b : \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} A_1 s + e_1 \\ A_2 s + e_2 \end{bmatrix}$$

and abort if  $A_1$  is not invertible. This happens with probability  $\leq 0.91$ . Next, provide

$$\begin{pmatrix} \tilde{A} := A_2 \cdot A_1^{-1} \\ \tilde{b} := -A_2 \cdot A_1^{-1} \cdot b_1 + b_2 \end{pmatrix}$$

to the HNF-LWE solver. Note that  $\tilde{b}$  simplifies to

$$\begin{aligned} \tilde{b} &= A_2 A_1^{-1} (A_1 s + e_1) + A_2 s + e_2 \\ &= A_2 s - A_2 A_1^{-1} e_1 + A_2 s + e_2 \\ &= A_2 e_1 + e_2 \end{aligned}$$

which is actually an HNF-LWE sample. Hence, if the HNF-LWE solver succeeds with non-negligible probability, then the LWE solver also succeeds with non-negligible probability.  $\square$

**Construction 6.1** (Compact LWE-PKE (?)).

- Gen: Sample  $s \leftarrow \chi_q^n$ ,  $A \leftarrow \mathbb{Z}_q^{n \times n}$ ,  $e \leftarrow \chi^n$ ,  $b \leftarrow As + e$  and output  $(\text{pk} = (A, b), \text{sk} = s)$ .
- Enc( $\text{pk}, \mu$ ): Sample  $u \leftarrow \mathbb{Z}_q^n$ ,  $e_0 \leftarrow \chi^n$  and  $e_1 \leftarrow \chi$ . Output  $c = (c_0 = A^\top u + e_0, c_1 = b^\top u + e_1 + \lfloor \frac{q}{2} \rfloor \mu)$ .
- Dec( $\text{sk}, c$ ): Output Decode( $c_1 - \langle c_0, s \rangle$ ).

The proof of security follows a similar strategy as earlier, except when replacing the ciphertext with a uniformly random string, we use the hardness of HNF-LWE.

#### 6.7.4 Ring-LWE

From the same motivation as above, we would like to improve the efficiency of our scheme by decreasing the number of random elements we sample (to get the matrix  $A$ ). Note that the methods we used

above require sampling  $O(n^2)$  random elements to form the random matrix.

- Instead, we could replace  $A$  with a structured (cyclic) matrix where each row is a cyclic shift of the previous row. This only requires  $O(n)$  random elements to be sampled. However, this turns out to not be secure.

Assuming that  $q = 1 \pmod n$ , we can write this in polynomial form as  $a(x) \cdot s(x) + e(x) = b(x) \pmod{(x^n - 1)}$ . The assumption on  $q$  is made so that number theoretic transforms can be used to efficiently multiply polynomials. Since  $x^n - 1$  is divisible by  $x - 1$ , we have the additional relation  $a(1) \cdot s(1) + e(1) = b(1)$  where both  $s(1)$  and  $e(1)$  are small. This allows us to construct a distinguisher.

- To prevent this, we would like to choose a irreducible polynomial instead, say,  $x^n + 1$ . Using negacyclic matrices instead (and  $q = 1 \pmod{2n}$ ), we get a similar relation  $a(x) \cdot s(x) + e(x) = b(x) \pmod{(x^n + 1)}$ . However, this does not have the same issue as before. This forms the basis of the Ring-LWE assumption. Other cyclotomic polynomials can also be used and we work in the ring  $\mathbb{Z}_q[x] / \langle x^n + 1 \rangle$ .

This construction was shown to be secure in [7].

7

**Definition 6.9 (HNF-RLWE).** Let  $n$  be the ring dimension,  $q$  be the modulus, and  $\chi$  be a distribution over  $R_q$  where  $R_q = \mathbb{Z}_q[x] / \langle x^n + 1 \rangle$ . The HNF-RLWE problem is defined as follows:

$$\begin{aligned} a(x) &\stackrel{\$}{\leftarrow} R_q \\ s(x) &\stackrel{\$}{\leftarrow} \chi \\ e(x) &\stackrel{\$}{\leftarrow} \chi \\ b(x) &\leftarrow a(x) \cdot s(x) + e(x) \end{aligned}$$

Distinguish  $(a(x), b(x))$  from  $(a(x), b'(x))$  where  $b'(x) \stackrel{\$}{\leftarrow} R_q$

- RLWE has similar worst-case to average-case reductions to hard lattice problems as LWE. However, these hard problems are on "ideal lattices" which are structured lattices.

Worst-case "ideal lattice" problems  $\leq$  Search RLWE  $\leq$  Decision RLWE

- RLWE  $\leq$  HNF-RLWE
- On the plus side, state of the art attacks on RLWE do not exploit the ring structure and instead treat it as a LWE instance.

We can also build a public-key encryption scheme from RLWE, similar to the compact LWE-PKE scheme. The security proof is also similar.

**Construction 6.2 (RLWE-PKE).**

- Gen: Sample  $s(x) \leftarrow \chi_q$ ,  $a(x) \leftarrow R_q$ ,  $e(x) \leftarrow \chi$ ,  $b(x) \leftarrow a(x) \cdot s(x) + e(x)$  and output  $(pk = (a(x), b(x)), sk = s(x))$ .
- Enc(pk,  $\mu$ ): Sample  $u(x) \leftarrow R_q$ ,  $e_0(x) \leftarrow \chi$  and  $e_1(x) \leftarrow \chi$ . Output  $c = (c_0 = a(x) \cdot u(x) + e_0(x), c_1 = b(x) \cdot u(x) + e_1(x) + \lfloor \frac{q}{2} \rfloor \mu)$ .
- Dec(sk,  $(c_0, c_1)$ ): Output Decode( $c_1(x) - c_0(x) \cdot s(x)$ ).

	LWE	RLWE
Public Key Size	$(n^2 + n) \log q$	$2n \log q$
Ciphertext Size	$(n + 1) \log q$	$2n \log q$
Message Length	1	$n$
Computation Cost	$O(n^2)$	$\tilde{O}(n)$

Table 6.1: Comparison of LWE and RLWE

## 6.7.5 Module LWE

<sup>8</sup>, along with the first implementable FHE scheme, introduced the Module-LWE assumption. This is a generalization of LWE and RLWE, and was initially created for notational purposes to avoid having to write two different definitions. Setting parameters to certain values returns LWE or RLWE. However, there are middle cases that are not covered by LWE or RLWE.

The Module-LWE assumption is defined as follows:

**Definition 6.10 (Module-LWE).** Let  $n$  be the degree,  $q$  be the modulus, and  $\chi$  be a distribution over  $R_q$ . Additionally, let  $m$  be the number of samples and  $k$  be the dimension. The Module-LWE problem is defined as follows:

$$\begin{aligned}
 A &\leftarrow R_q^{m \times k} \\
 s &\leftarrow \chi^k \\
 e &\leftarrow \chi^m \\
 b &\leftarrow A \cdot s + e
 \end{aligned}$$

Distinguish  $(A, b)$  from  $(A, b')$  where  $b' \leftarrow R_q^m$

Note that this is the same as LWE when  $k = 1$  and RLWE when  $n = 1$ .

- Worst-case “rank- $k$ ” lattice problems  $\leq$  Search Module-LWE  $\leq$  Decision Module-LWE
- Module-LWE  $\leq$  HNF-Module-LWE <sup>9</sup>
- Why use MLWE? - Efficiency and Flexibility.

8

9

- Powers of two are sparse, we would need to jump from  $n = 512$  to  $n = 1024$  for instance.
- New Hope is another NIST candidate based on RLWE. It has parameters  $n = 1024$ ,  $q = 12289$  while Kyber has  $n = 256$ ,  $q = 3329$ . The ring LWE condition requires  $q = 1 \pmod{2n}$ , and hence the larger  $q$ .
- Other engineering considerations for Kyber:
  - The Fujisaki-Okamoto transform is used to obtain CCA security.
  - Note that Kyber's prime does not satisfy  $q = 1 \pmod{2n}$ , and they only get incomplete NTT speedups.
  - The matrix  $A$  can be compressed using a random oracle or hash.
  - The error distribution is restricted to  $[-3, 3]$ . In addition, for easier sampling, instead of using a uniform distribution, they use a centered binomial distribution (sampling from this only requires sampling some bits and adding them).
  - Another optimization for proof size is to drop the last bit of the ciphertext.
- Finally, Kyber gets a public key size of 800 bytes, a ciphertext size of 768 bytes, and decryption failure probability of  $2^{-139}$ .

## 6.8 Cramer-Shoup Construction

The Cramer-Shoup cryptosystem is a public-key encryption scheme that achieves security against adaptive chosen ciphertext attacks (CCA2). To understand its significance, let's first examine the ElGamal encryption scheme and its limitations.

### 6.8.1 Background: ElGamal Encryption

ElGamal encryption is a simple and elegant public-key system based on the Diffie-Hellman key exchange. The scheme operates as follows:

- $\text{GEN}(1^n)$ :
  - Select prime  $q$  where  $|q| = n$
  - Choose generator  $g$  of group  $G$  of order  $q$
  - Sample  $x \leftarrow \mathbb{Z}_q$  randomly
  - Compute  $h = g^x$
  - Output  $(\text{pk} = (g, h), \text{sk} = x)$
- $\text{ENC}(\text{pk}, m \in G)$ :
  - Sample  $r \leftarrow \mathbb{Z}_q$  randomly

Compute  $c_1 = g^r$   
 Compute  $c_2 = m \cdot h^r$   
 Output  $(c_1, c_2)$

- $\text{DEC}(\text{sk}, (c_1, c_2))$ :  
 Compute  $m = c_2 / c_1^x$   
 Output  $m$

**Proof of Correctness:**

$$\begin{aligned} c_2 / c_1^x &= (m \cdot h^r) / (g^r)^x \\ &= (m \cdot (g^x)^r) / (g^r)^x \\ &= m \cdot g^{xr} / g^{rx} \\ &= m \end{aligned}$$

However, ElGamal is malleable, meaning an adversary can modify a ciphertext to create a related ciphertext. Given a ciphertext  $(c_1, c_2)$  encrypting message  $m$ , an adversary can create  $(c_1, k \cdot c_2)$  which will decrypt to  $k \cdot m$  for any  $k$ . This malleability makes ElGamal insecure against chosen-ciphertext attacks.

### 6.8.2 Hash Proof Systems

To address these limitations, we introduce Hash Proof Systems (HPS), a powerful tool for constructing CCA-secure encryption schemes.

### 6.8.3 Formal Definition

Let  $\mathbb{X}$  be a set and  $\mathbb{L} \subset \mathbb{X}$  be a language defined by:

$$\mathbb{L} = \{x \in \mathbb{X} \mid \exists w \text{ s.t. } (x, w) \in \mathbb{R}\}$$

where  $\mathbb{R}$  is a binary relation and  $w$  is called a witness.

A Hash Proof System consists of three algorithms:

- $\mathcal{KGen}(1^n)$ : Generates key pair  $(pk, sk)$
- $\mathcal{H}_{sk} : \mathbb{X} \rightarrow \Pi$  (private evaluation)
- $\mathcal{H}_{pk} : \mathbb{L} \times \mathbb{W} \rightarrow \Pi$  (public evaluation)

### 6.8.4 Key Properties

1. **Correctness:** For all  $(x, w) \in \mathbb{R}$ :

$$\mathcal{H}_{sk}(x) = \mathcal{H}_{pk}(x, w)$$

**Proof:** This follows from the construction where both evaluations compute the same mathematical operation, but  $\mathcal{H}_{pk}$  requires a witness while  $\mathcal{H}_{sk}$  uses the secret key.

2. **Smoothness:** For all  $x \notin \mathbb{L}$ :

$$\{pk, \mathcal{H}_{sk}(x)\} \stackrel{s}{\approx} \{pk, U_{\Pi}\}$$

where  $U_{\Pi}$  is uniform over  $\Pi$ .

**Proof Sketch:** This property follows from the DDH assumption in our concrete construction. When  $x \notin \mathbb{L}$ , the hash value appears random to any computationally bounded adversary.

3. **Universal Property:** For all  $x \in \mathbb{L}$  and  $y_1, \dots, y_t \in \mathbb{L}$ :

$$\{pk, \mathcal{H}_{sk}(x), \{\mathcal{H}_{pk}(y_i)\}_i\} \stackrel{s}{\approx} \{pk, U_{\Pi}, \{\mathcal{H}_{sk}(y_i)\}_i\}$$

### 6.8.5 Concrete DDH-based Construction

Let's construct a specific HPS based on the Decision Diffie-Hellman (DDH) assumption:

- Let  $G$  be a cyclic group of prime order  $q$
- Fix generators  $g_1, g_2 \in G$
- Define  $\mathbb{X} = G^2$
- Define  $\mathbb{L} = \{(h_1, h_2) \in \mathbb{X} \mid \exists r \in \mathbb{Z}_q : h_1 = g_1^r \text{ and } h_2 = g_2^r\}$

The construction:

$$\mathcal{KGen}(1^n) : \text{Sample } d_1, d_2 \leftarrow \mathbb{Z}_q$$

$$\text{Output } (pk = (g_1^{d_1} g_2^{d_2}), sk = (d_1, d_2))$$

$$\mathcal{H}_{sk}(x = (h_1, h_2)) = h_1^{d_1} h_2^{d_2}$$

$$\mathcal{H}_{pk}(x = (h_1, h_2), w = r) = pk^r$$

**Proof of Correctness:** For  $(x, w) \in \mathbb{R}$  where  $x = (g_1^r, g_2^r)$ :

$$\begin{aligned} \mathcal{H}_{sk}(x) &= (g_1^r)^{d_1} (g_2^r)^{d_2} \\ &= g_1^{rd_1} g_2^{rd_2} \\ &= (g_1^{d_1} g_2^{d_2})^r \\ &= pk^r \\ &= \mathcal{H}_{pk}(x, w) \end{aligned}$$

### 6.8.6 Basic Scheme and Its Limitations

Let's analyze our first attempt at constructing a secure encryption scheme using HPS:

$\mathcal{Gen}(1^n)$ :

- $(pk, sk) \leftarrow \mathcal{KGen}(1^n)$

$\text{Enc}(pk, m \in G)$ :

1. Sample  $x \leftarrow \mathbb{L}$  along with witness  $w$
2. Compute  $e = \mathcal{H}_{pk}(x, w) \cdot m$
3. Output  $(x, e)$

$\text{Dec}(sk, (x, e))$ :

- Output  $e / \mathcal{H}_{sk}(x)$

### 6.8.7 Security Analysis of Basic Scheme

While this construction achieves CPA security through the following hybrid argument:

1.  $\mathcal{H}_0$ : Real encryption game
2.  $\mathcal{H}_1$ : Replace  $e = \mathcal{H}_{sk}(x) \cdot m$  (using  $sk$  instead of  $pk$ )
  - This is identical to  $\mathcal{H}_0$  because  $x \in \mathbb{L}$  and by HPS correctness
  - Formally: For any  $(x, w) \in \mathbb{R}$ ,  $\mathcal{H}_{pk}(x, w) = \mathcal{H}_{sk}(x)$
3.  $\mathcal{H}_2$ : Replace  $x \leftarrow \mathbb{X} \setminus \mathbb{L}$ 
  - Indistinguishable by property 2 of HPS
  - Adversary cannot tell if  $x$  is sampled from  $\mathbb{L}$  or  $\mathbb{X} \setminus \mathbb{L}$
4.  $\mathcal{H}_3$ : Replace  $e$  with uniform random element
  - Indistinguishable by smoothness property of HPS
  - When  $x \notin \mathbb{L}$ ,  $\mathcal{H}_{sk}(x)$  is statistically close to uniform

However, this scheme fails to achieve CCA-2 security. Here's a concrete attack:

**Theorem 6.3** (CCA-2 Attack on Basic Scheme). *The basic HPS construction is not CCA-2 secure.*

*Proof.* Consider the following attack in the CCA-2 game:

1. Adversary receives challenge ciphertext  $(x^*, e^*)$  for message  $m_b$
2. Adversary creates modified ciphertext  $(x^*, k \cdot e^*)$  for some  $k \in G$



3. Adversary submits modified ciphertext to decryption oracle
4. Let  $m'$  be the decrypted result. Then:

$$\begin{aligned}
 m' &= (k \cdot e^*) / \mathcal{H}_{sk}(x^*) \\
 &= k \cdot (e^* / \mathcal{H}_{sk}(x^*)) \\
 &= k \cdot m_b
 \end{aligned}$$

5. Since  $k$  is known, adversary can recover  $m_b$  and win the CCA-2 game

□

### 6.8.8 The Need for Ciphertext Integrity

The key insight is that we need to prevent ciphertext manipulation. The full Cramer-Shoup construction addresses this by:

1. Adding a second HPS instance ( $\mathcal{H}'$ ) that acts as a "proof of well-formedness"
2. Using a collision-resistant hash function to bind all components together
3. Verifying the proof  $\pi$  before decryption

The second HPS instance must satisfy a stronger security property called "2-smoothness":

**Definition 6.11** (2-Smoothness). For all  $x_1, x_2 \in \mathbb{L}$  and  $t_1, t_2 \in T$  with  $t_1 \neq t_2$ :

$$\{pk, \mathcal{H}_{sk}(x_1, t_1), \mathcal{H}_{sk}(x_2, t_2)\} \stackrel{s}{\approx} \{pk, U_{\Pi}, U_{\Pi}\}$$

### 6.8.9 CCA-2 Secure Construction

The final Cramer-Shoup construction achieves CCA-2 security by combining two HPS instances with a collision-resistant hash function:

#### 6.8.10 The Scheme

$\mathcal{G}\text{en}(1^n)$ :

$$(pk, sk) \leftarrow \mathcal{KG}\text{en}(1^n)$$

$$(pk', sk') \leftarrow \mathcal{KG}\text{en}(1^n)$$

Output  $((pk, pk'), (sk, sk'))$

Enc( $pk, m$ ):

Sample  $x \leftarrow \mathbb{L}$  with witness  $w$

$$e = \mathcal{H}_{pk}(x, w) \cdot m$$

$$t = \text{CRHF}(x, e)$$

```

 $\pi = \mathcal{H}'_{pk}(x, w, t)$ 
Output  $(x, e, \pi)$ 
Dec( $sk, (x, e, \pi)$ ):
if  $\pi \neq \mathcal{H}'_{sk}(x)$  then
    Output  $\perp$ 
end if
Output  $e / \mathcal{H}_{sk}(x)$ 

```

### 6.8.11 Final Security Theorem

**Theorem 6.4.** *If  $\mathcal{H}$  is a 1-smooth HPS,  $\mathcal{H}'$  is a 2-smooth HPS, and CRHF is a collision-resistant hash function, then the scheme is CCA-2 secure.*

*Proof.* We prove security through a sequence of hybrid games, showing each transition is indistinguishable to a PPT adversary. Let  $A$  be any PPT adversary in the CCA2 game.

First, we establish a crucial lemma:

**Lemma 6.2** (Key Soundness Lemma). *For any ciphertext decryption query  $(x, e, \pi)$  that  $A$  makes, if  $\text{Dec}((sk, sk'), (x, e, \pi)) \neq \perp$  then  $x \in \mathbb{L}$  except with negligible probability.*

*Proof of Lemma.* Suppose for contradiction that  $x \notin \mathbb{L}$  but the decryption doesn't return  $\perp$ . This means:

$$\pi = \mathcal{H}'_{sk'}(x, \text{CRHF}(x, e))$$

By the 2-smoothness property of  $\mathcal{H}'$ , when  $x \notin \mathbb{L}$ , the value  $\mathcal{H}'_{sk'}(x, t)$  is statistically indistinguishable from random, even given  $pk'$ . Therefore, the probability that  $A$  can generate such a  $\pi$  is at most  $1/|\Pi|$ , which is negligible.  $\square$

Now we proceed with the hybrid games:

- $\mathcal{G}_0$ : The real CCA2 game.
- $\mathcal{G}_1$ : Same as  $\mathcal{G}_0$ , but the challenger computes the challenge ciphertext's  $e^*$  component using  $sk$  instead of  $pk$ :

$$e^* = \mathcal{H}_{sk}(x^*) \cdot m_b$$

instead of

$$e^* = \mathcal{H}_{pk}(x^*, w) \cdot m_b$$

By the correctness property of HPS, these are identical when  $x^* \in \mathbb{L}$ , so:

$$\Pr[\mathcal{G}_0 = 1] = \Pr[\mathcal{G}_1 = 1]$$

- $\mathcal{G}_2$ : Same as  $\mathcal{G}_1$ , but now the challenger samples  $x^* \leftarrow \mathbb{X} \setminus \mathbb{L}$  instead of from  $\mathbb{L}$ .

By the hardness of distinguishing elements in  $\mathbb{L}$  from elements in  $\mathbb{X} \setminus \mathbb{L}$  (which follows from the DDH assumption in our concrete construction):

$$|\Pr[\mathcal{G}_1 = 1] - \Pr[\mathcal{G}_2 = 1]| \leq \epsilon_{\text{DDH}}$$

- $\mathcal{G}_3$ : Same as  $\mathcal{G}_2$ , but replace  $\mathcal{H}_{sk}(x^*)$  with a uniform random value  $u \leftarrow \Pi$  in computing  $e^*$ :

$$e^* = u \cdot m_b$$

By the smoothness property of  $\mathcal{H}$ , since  $x^* \notin \mathbb{L}$ :

$$|\Pr[\mathcal{G}_2 = 1] - \Pr[\mathcal{G}_3 = 1]| \leq \epsilon_{\text{smooth}}$$

- $\mathcal{G}_4$ : Same as  $\mathcal{G}_3$ , but replace  $e^*$  with a uniform random value in  $G$ . Since  $u$  is uniform in  $\Pi$  and independent of  $m_b$ ,  $e^*$  is uniform in  $G$  and independent of  $m_b$ , so:

$$\Pr[\mathcal{G}_3 = 1] = \Pr[\mathcal{G}_4 = 1]$$

In  $\mathcal{G}_4$ , the challenge ciphertext is independent of  $m_b$ , so  $A$  has no advantage. Therefore:

$$\text{Adv}_A^{\text{CCA2}} \leq \epsilon_{\text{DDH}} + \epsilon_{\text{smooth}}$$

To complete the proof, we need to show that the decryption oracle queries in all games can be answered properly. For any decryption query  $(x, e, \pi)$ :

1. If  $(x, e) = (x^*, e^*)$  but  $\pi \neq \pi^*$ : By 2-smoothness of  $\mathcal{H}'$ , generating a valid  $\pi$  is infeasible.
2. If  $(x, e) \neq (x^*, e^*)$  but  $\text{CRHF}(x, e) = \text{CRHF}(x^*, e^*)$ : This breaks collision resistance of CRHF.
3. If  $\text{CRHF}(x, e) \neq \text{CRHF}(x^*, e^*)$ : By 2-smoothness of  $\mathcal{H}'$ , any  $\pi$  that verifies must correspond to a witness  $w$  such that  $(x, w) \in \mathbb{R}$ , meaning  $x \in \mathbb{L}$ . Therefore, the decryption oracle can be simulated properly.

Thus, all hybrid games are indistinguishable to  $A$ , and the scheme is CCA2-secure.  $\square$



# 7

## Advanced Encryption Schemes

### 7.1 Identity-Based Encryption

We introduce Bilinear Maps and two of its applications: NIKÉ, Non-Interactive Key Exchange; and IBE, Identity Based Encryption.

### 7.2 Diffie-Hellman Key Exchange

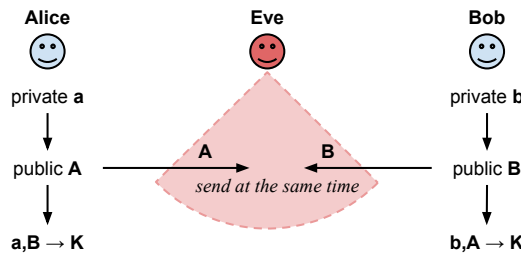


Figure 7.1: Diffie-Hellman Key Exchange

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,  $\Pr[k = \text{Eve}(A, B)] = \text{neg}(k)$ , where  $k$  is the length of  $a$ .

#### 7.2.1 Discussion 1

Assume that  $\forall (g, p)$ , and  $a_1, b_1 \xleftarrow{\$} \mathbb{Z}_p^*$ , and  $a_2, b_2, r \xleftarrow{\$} \mathbb{Z}_p^*$ , we have  $(g^{a_1}, g^{b_1}, g^{a_1 b_1}) \stackrel{\$}{\sim} (g^{a_2}, g^{b_2}, g^r)$ . 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  $b, r \leftarrow \mathbb{Z}_p^*$  ( $g^b, mA^b + (1 - m)g^r$ )

Alice  $Dec(sk, (c_1, c_2))$

$c_1^a \stackrel{?}{=} c_2$

## 7.3 Bilinear Maps

### Definition 7.1. Bilinear Maps

Bilinear Maps is  $(G, P, G_T, g, e)$ , where  $e$  is an efficient function  $G \times G \rightarrow G_T$  such that

- if  $g$  is generator of  $G$ , then  $e(g, g)$  is the generator of  $G_T$ .
- $\forall a, b \in \mathbb{Z}_p$ , 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

$$\begin{aligned} a, b, c &\xleftarrow{\$} \mathbb{Z}_p^*, r \xleftarrow{\$} \mathbb{Z}_p^* \\ A &= g^a, B = g^b, C = g^c \\ K &= e(g, g)^{abc} \end{aligned}$$

## 7.5 IBE: Identity-Based Encryption

When two parties communicate secure messages through a public key infrastructure, they need to go through a time-consuming and error-prone process to get each other's key and verify each other's identity through a Certificate Authority. Identity-based cryptography (IBC) seeks to reduce these barriers by requiring no preparation on

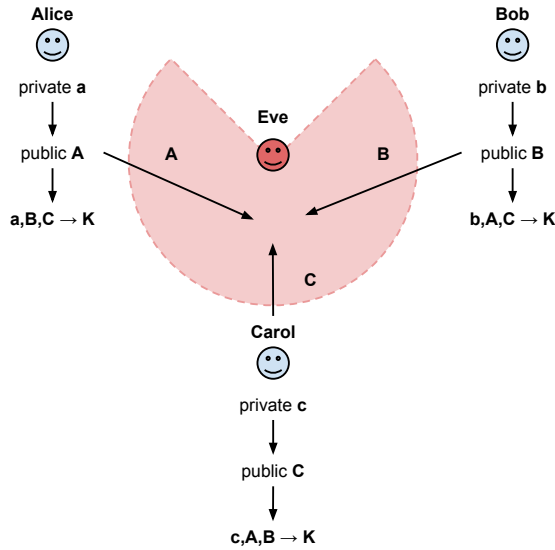


Figure 7.2: Tripartite Diffie-Hellman Key Exchange

the part of the message recipient, therefore saving the initial round trip. Identity based encryption can also be used to construct CCA-secure public key encryption and digital signatures.

IBE contains four steps: *Setup*, *KeyGen*, *Enc*, and *Dec*. We illustrate it in Figure 7.3. IBE relies on a trusted third party called Private Key Generator(PKG). In first step, PKG gets a Master Public Key ( $mpk$ ) and Master Signing Key ( $msk$ ) from  $Gen(1^n)$ . Then a user with an ID (in this example, “Mike”), sends his ID to the PKG. The PKG generates the Signing Key of Mike with  $KeyGen(msk, id)$  and sends it back. Another user, 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.

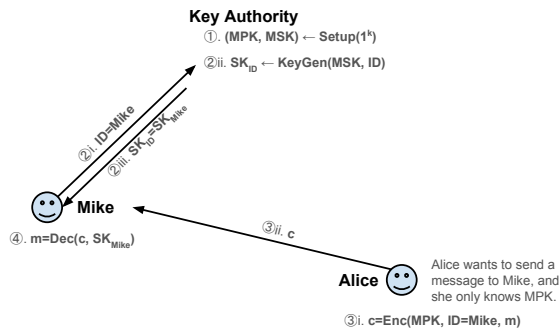


Figure 7.3: Identity-Based Encryption

Then we define IBE formally,

**Definition 7.2** (Identity-Based Encryption). *An identity based encryption scheme  $\mathcal{E}_{id} = (G, K, E, D)$  is a tuple of four efficient algorithms: a setup algorithm  $G$ , a key generation algorithm  $K$ , an encryption algorithm  $E$ , and a decryption algorithm  $D$ .*

- $G$  is a probabilistic algorithm invoked as  $(mpk, msk) \xleftarrow{\$} G(1^n)$ , where  $mpk$  is called the **master public key** and  $msk$  is called the **master secret key** for the IBE scheme.
- $K$  is a probabilistic algorithm invoked as  $sk_{id} \xleftarrow{\$} K(msk, id)$ , where  $msk$  is the master secret key (as output by  $G$ ),  $id \in \mathcal{ID}$  is an identity, and  $sk_{id}$  is a secret key for  $id$ .
- $E$  is a probabilistic algorithm invoked as  $c \xleftarrow{\$} E(mpk, id, m)$ .
- $D$  is a deterministic algorithm invoked as  $m \leftarrow D(sk_{id}, c)$ . Here  $m$  is either a message or a special reject value  $\perp$  (distinct from all messages).

As usual, we define the correctness of IBE to be the decryption undoes encryption, formally we have

**Definition 7.3** (Correctness of IBE).  $\forall n, id, m$ , we have

$$\Pr \left[ \begin{array}{l} (mpk, msk) \leftarrow G(1^n), \\ sk_{id} \leftarrow K(msk, ID), \\ c \leftarrow E(mpk, id, m), \\ m \leftarrow D(sk_{id}, c) \end{array} \right] = 1 - \text{negl}(n)$$

Next we define the security of IBE scheme. The basic security definition considers an adversary who obtains the secret keys for a number of identities of its choice. The adversary should not be able to break semantic security for some other identity of its choice for which it does not have the secret key.

**Definition 7.4** (Security of IBE). *For an IBE scheme  $\mathcal{E}_{id} = (G, K, E, D)$  is secure if  $\forall$  nuPPT  $\mathcal{A}$ ,*

$$\Pr[\text{Exp}_{\pi, \mathcal{A}}^{\text{IBE}, \text{CPA}}(n) = 1] = \text{negl}(n)$$

We then define the experiment  $\text{Exp}_{\pi, \mathcal{A}}^{\text{IBE}, \text{CPA}}$ .



**Definition 7.5** (IBE-CPA Experiment). We denote the experiment in the following order:

1. The challenger invokes  $(mpk, msk) \xleftarrow{\$} G(1^n)$  and send  $mpk$  to adversary  $\mathcal{A}$ .
2.  $\mathcal{A}$  can make multiple key queries and generate desired ID  $id^*$  and two message  $(m_0, m_1)$ .
3. Challenger random selects  $b \xleftarrow{\$} \{0, 1\}$  and encrypt  $c^* = E(mpk, id^*, m_b)$  and send  $c^*$  to  $\mathcal{A}$ .
4.  $\mathcal{A}$  can make more encryption queries based on  $c^*$  and other id and message  $(m_0, m_1)$ . Note that throughout the process  $\mathcal{A}$  can **never** make any query on  $id^*$ .
5. At the end,  $\mathcal{A}$  generate  $b'$  and send to challenger.
6. Challenger output 1 if  $b = b'$ , 0 otherwise.

### 7.5.1 IBE Construction from Pairing

We then present a concrete IBE construction from pairing. First, we will give the hardness assumption in this scheme called **bilinear Diffie-Hellman** assumption, or BDH. This assumption says that given random element  $g_0^\alpha, g_0^\beta, g_0^\gamma \in \mathbb{G}$  and a few additional terms, the quantity  $e(g_0, g_1)^{\alpha\beta\gamma} \in \mathbb{G}_T$  is computationally indistinguishable from a random element in  $\mathbb{G}_T$ .

**Definition 7.6** (Decisional bilinear Diffie-Hellman). Let  $e : \mathbb{G}_0 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$  be a pairing where  $\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_T$  are cyclic groups of prime order  $q$  with generators  $g_0 \in \mathbb{G}_0$  and  $g_1 \in \mathbb{G}_1$ . For a given adversary  $\mathcal{A}$ , the following distribution is distinguishable:

$$\{g_0^\alpha, g_1^\alpha, g_0^\beta, g_1^\beta, e(g_0, g_1)^{\alpha\beta\gamma}, \alpha, \beta, \gamma \xleftarrow{\$} \mathbb{Z}_q\} \approx^c \{g_0^\alpha, g_1^\alpha, g_0^\beta, g_1^\beta, e(g_0, g_1)^\delta, \alpha, \beta, \gamma, \delta \xleftarrow{\$} \mathbb{Z}_q\}$$

Note that this assumption work even with  $g_0 = g_1$ .

We then present our IBE construction.

- $G(1^n)$ :

$$\alpha \xleftarrow{\$} \mathbb{Z}_q, mpk \leftarrow g^\alpha, msk \leftarrow \alpha$$

and output  $(mpk, msk)$

- $K(msk = \alpha, id)$ :

$$sk_{id} \leftarrow H(id)^\alpha$$

where  $H$  is a hash function  $H : \{0, 1\}^* \rightarrow \mathbb{G}$

- $E(mpk, id, m)$ :

$$\beta \xleftarrow{\$} \mathbb{Z}_q, c_1 \leftarrow g^\beta, c_2 \leftarrow e(mpk, H(id)^\beta) \cdot m$$

and output  $(c_1, c_2)$ .

- $D(sk_{id}, c = (c_1, c_2))$ :

$$m = \frac{c_2}{e(c_1, sk_{id})}$$

By the property of bilinear map, we can verify the correctness of this scheme,

$$\begin{aligned} m &= \frac{c_2}{e(c_1, sk_{id})} \\ &= \frac{e(mpk, H(id)^\beta) \cdot m}{e(c_1, sk_{id})} \\ &= \frac{e(g^\alpha, H(id)^\beta) \cdot m}{e(g^\beta, H(id)^\alpha)} \\ &= \frac{e(g, H(id))^{\alpha\beta}}{e(g, H(id))^{\alpha\beta}} \cdot m \\ &= m \end{aligned}$$

We then prove the security property under random oracle model.

**Theorem 7.1.** *If decision BDH holds for  $e$ ,  $H$  is modelled as a random oracle, then the above construction is a secure IBE scheme.*

*Proof.* Let  $\mathcal{A}$  be an adversary that breaks the IBE scheme, we can construct another adversary  $\mathcal{B}$  such that it breaks the DBDH assumption.

The adversary  $\mathcal{B}$  works as follows:

1.  $\mathcal{B}$  receives 5 elements as input  $\{g^\alpha, g^\beta, g^\gamma, z, \alpha, \beta, \gamma \xleftarrow{\$} \mathbb{Z}_q, z \in \mathbb{G}\}$ , and  $\mathcal{B}$  need to determine whether  $z = e(g, g)^{\alpha\beta\gamma}$  or not.
2.  $\mathcal{B}$  send IBE public parameter  $mpk = g^\alpha$  to IBE adversary  $\mathcal{A}$ .
3. Then  $\mathcal{A}$  will make multiple  $sk_{id}$  queries to  $\mathcal{B}$ .  $\mathcal{B}$  responds them by  
 (1) Choose  $\rho \xleftarrow{\$} \mathbb{Z}_q$  (2) Setting  $H(id) = g^\rho$  (3) set the secret key be  $sk_{id} = H(id)^\alpha = g^{\rho\alpha}$ . One **exception** is that  $\mathcal{B}$  will set a random  $id'$  whose  $H(id') = g^\beta$ .
4. After receiving the key query,  $\mathcal{A}$  outputs  $(id^*, m_0, m_1)$ .
5. When  $\mathcal{B}$  receives encryption query  $(id^*, m_0, m_1)$ , it first check if  $\mathcal{A}$  have previously query  $sk_{id^*}$  before, if yes, then abort. Otherwise,  $\mathcal{B}$  choose  $b \xleftarrow{\$} \{0, 1\}$  and encrypt  $m_b$  using  $(c_1 = g^\gamma, c_2 = e(msk, H(id')^\gamma) \cdot m_b)$  and send back to  $\mathcal{A}$ .
6.  $\mathcal{A}$  eventually output  $b'$  to  $\mathcal{B}$ . And  $\mathcal{B}$  output 1 if  $b' = b$  and 0 otherwise.

Since  $e(msk, H(id')^\gamma) = e(g^\alpha, g^{\beta\gamma}) = e(g, g)^{\alpha\beta\gamma}$ ,  $\mathcal{B}$  can embedded the challenge to  $\mathcal{A}$  and break DBDH.  $\square$   $\square$

### 7.5.2 Digital Signature from IBE

We can directly derive a secure signature scheme from IBE. Given a secure IBE  $\mathcal{E}_{id} = (G, K, E, D)$  with id space  $\mathcal{ID}$  and message space  $\mathcal{M}_{IBE}$ , we construct a secure digital signature scheme  $\mathcal{S} = (G', S', V')$  as follows:

- $G'(1^n)$ : run  $(mpk, msk) \xleftarrow{\$} G(1^n)$  and output  $(mpk, msk)$  as the sign key pair, with  $mpk$  as verification key and  $msk$  as sign key.
- $S'(msk, m)$ : Given message  $m \in \mathcal{ID}$ , compute  $\sigma \xleftarrow{\$} K(msk, m)$ , output  $\sigma$  as signature.
- $V'(mpk, m, \sigma)$ : Choose  $r \xleftarrow{\$} \mathcal{M}_{IBE}$ , compute  $c \xleftarrow{\$} E(mpk, m, r)$ , and accept if  $D(\sigma, c) = r$ .

We have the following theorem,

**Theorem 7.2.** *Let  $\mathcal{E}_{id}$  be a secure IBE with message space is super-poly. Then the derive signature scheme  $\mathcal{S}$  is a secure digital signature scheme.*

*Proof.* Let  $\mathcal{A}$  be an adversary that breaks the digital signature scheme, we can construct another adversary  $\mathcal{B}$  such that breaks the IBE security.

The BIE adversary  $\mathcal{B}$  is modelled as follows:

1.  $\mathcal{B}$  receives  $mpk$  from the challenger, and forward  $mpk$  as a signature public key to  $\mathcal{A}$ .
2.  $\mathcal{A}$  makes a series of signing queries  $m_0, \dots, m_n$  to  $\mathcal{B}$ .  $\mathcal{B}$  responds by issuing the corresponding key query to the challenger and forwarding the answer back the response to  $\mathcal{A}$ .
3.  $\mathcal{A}$  will output a signature forgery  $(m, \sigma)$  which it didn't issue the sign query  $m$ .
4.  $\mathcal{B}$  then choose two random message  $t_0, t_1 \xleftarrow{\$} \mathcal{M}_{IBE}$  and issue encryption query with the identity  $m$ .
5.  $\mathcal{B}$  gets back the ciphertext  $c \xleftarrow{\$} E(mpk, m, t_b)$  for  $b \in \{0, 1\}$ . Then it runs  $t' \leftarrow D(\sigma, c)$  and output  $b' = t'$ .

We observe that

- when  $b = 1$ , then  $c \xleftarrow{\$} E(mpk, m, t_1)$ , and  $\mathcal{B}$  outputs 1 with probability the same as the probability of  $\mathcal{A}$  breaks digital signature, we note as  $SIGadv[\mathcal{A}, \mathcal{S}]$

- when  $b = 0$ , then  $c \xleftarrow{\$} E(mp_k, m, t_0)$ , and  $\mathcal{B}$  outputs 1 with probability  $1/|\mathcal{M}_{IBE}|$  since  $\mathcal{B}$  can only make random guess in the message space.

We then have the probability of  $\mathcal{B}$  break IBE scheme

$$\frac{1}{2}(SIGadv[\mathcal{A}, \mathcal{S}] + 1/|\mathcal{M}_{IBE}|)$$

which is not negligible if  $SIGadv[\mathcal{A}, \mathcal{S}]$  is not negligible. □

□

## 7.6 Fully Homomorphic Encryption

So far, we've seen private and public key encryption and different security properties (CPA, CCA). We've also seen some advanced encryption schemes like Identity-Based Encryption (IBE) that allow us to encrypt to an identity rather than a public key.

Consider an example where Alice has a complex tax return to fill out and decides to use a tax return preparation service. The current pipeline is as follows:

1. Alice sends her tax forms (W2, 1099, etc) to the service.
2. The service prepares the tax return and sends it back to Alice.
3. Alice sends the tax return to the IRS.

However, in this process, the service has access to all of Alice's tax information, which is a privacy concern.

Consider an alternate scenario where Alice encrypts her tax forms before sending them to the service. It would be ideal if the service could still prepare the tax return without decrypting the forms by performing operations on the encrypted data. In this case, the service learns nothing about Alice's tax information but is still able to prepare the tax return. This is the idea behind Fully Homomorphic Encryption (FHE). FHE was first presented in [Gentry09] <sup>1</sup> and has since been improved upon. We will present the construction from [GSW13] <sup>2</sup>.

<sup>1</sup>

<sup>2</sup>

FHE can be defined in either private or public key settings. The below construction is defined in the private key setting for message space  $\mathbb{Z}_2$ .

**Definition 7.7** (Fully Homomorphic Encryption (FHE)). *A FHE scheme for message space  $\mathbb{Z}_2$  and circuit class  $\mathcal{C}$  is a tuple of algorithms  $(\text{Gen}, \text{Enc}, \text{Dec}, \text{Eval})$  such that:*

- $\text{Gen}(1^\lambda) \rightarrow (\text{ek}, \text{sk})$ : The key generation algorithm takes a security parameter  $\lambda$  and outputs a secret key  $\text{sk}$  and evaluation key  $\text{ek}$ .
- $\text{Enc}(\text{sk}, m) \rightarrow c$ : The encryption algorithm takes a public key  $\text{sk}$  and message  $m$  and outputs a ciphertext  $c$ .
- $\text{Dec}(\text{sk}, c) \rightarrow m$ : The decryption algorithm takes a secret key  $\text{sk}$  and ciphertext  $c$  and outputs a message  $m$ .
- $\text{Eval}(\text{ek}, F, c_1, \dots, c_l) \rightarrow \tilde{c}$ : The evaluation algorithm takes an evaluation key  $\text{ek}$ , a circuit  $F \in \mathcal{C}$ , and  $l$  ciphertexts  $c_1, \dots, c_l$  and outputs a ciphertext  $\tilde{c}$ .

A FHE scheme satisfies the following properties:

- **Correctness:**  $\forall n \in \mathbb{N}, \forall F \in \mathcal{C}, \forall (\mu_1, \mu_2, \dots, \mu_l) \in \mathbb{Z}_2^l$ ,  

$$\Pr[\text{Dec}(\text{sk}, \text{Eval}(\text{ek}, F, \text{Enc}(\text{sk}, \mu_1), \dots, \text{Enc}(\text{sk}, \mu_l))) = F(\mu_1, \mu_2, \dots, \mu_l)] = 1 - \text{negl}(\lambda)$$

- **Security:** The following two distributions are computationally indistinguishable:

$$\begin{aligned} &\{(\text{ek}, \text{ct}_0) : \text{ct}_0 \leftarrow \text{Enc}(\text{sk}, 0), (\text{ek}, \text{sk}) \leftarrow \text{Gen}(1^\lambda)\} \\ &\{(\text{ek}, \text{ct}_1) : \text{ct}_1 \leftarrow \text{Enc}(\text{sk}, 1), (\text{ek}, \text{sk}) \leftarrow \text{Gen}(1^\lambda)\} \end{aligned}$$

- **Compactness:** The size of the ciphertext  $\text{Eval}(\text{ek}, F, c_1, \dots, c_l)$  is equal to a fresh encryption of the output of the circuit  $F$  on the plaintexts  $\mu_1, \dots, \mu_l$ :

$$\begin{aligned} &\forall i, c_i = \text{Enc}(\text{sk}, \mu_i) \\ &|\text{Eval}(\text{ek}, F, c_1, \dots, c_l)| = |\text{Enc}(\text{sk}, F(\mu_1, \dots, \mu_l))| \end{aligned}$$

The construction of FHE is based on the Learning With Errors (LWE) problem. The high-level construction is done in two steps:

1. **Leveled FHE:** We first construct a leveled FHE scheme that can evaluate arbitrary circuits of bounded depth.
2. **Bootstrapping:** We then use the leveled FHE scheme to construct a fully homomorphic encryption scheme.

## 7.6.1 Leveled FHE

To present some intuition for how we get homomorphic properties, consider the following construction. Let  $C \in \mathbb{Z}_q^{l \times l}$  be a matrix and  $v \in \mathbb{Z}_q^l$  be an eigenvector of this matrix. The eigenvalue is chosen as the message being encrypted.

Given this, we can easily perform operations on ciphertexts that correspond to operations on the underlying plaintexts.

- **Addition:** Given two ciphertexts  $C_1$  and  $C_2$ , we have  $(C_1 + C_2)v = C_1v + C_2v = (m_1 + m_2)v$ .
- **Multiplication:** Given two ciphertexts  $C_1$  and  $C_2$ , we have  $(C_1 \cdot C_2)v = C_1(C_2v) = C_1(m_2v) = m_1m_2v$ .

Note that this is not a secure construction as presented, since with enough samples we can solve a linear system to obtain the secret key. However, this gives us some intuition for how we can perform operations on encrypted data. To make this construction secure, we need to add noise to the ciphertexts (which is where LWE comes in). However, a naive way of doing this does not work; suppose we have that  $Cv = mv + e$  where  $e$  is the (small) noise term. Then, even a single multiplication gives us  $C_1C_2v = m_1m_2v + C_1e_2$ , where the noise term is no longer guaranteed to be small since  $C$  has no such guarantees.

The construction of the leveled FHE scheme is as follows. We use the LWE problem with parameters  $(n, m, q, \chi)$  where  $n$  is the dimension of the secret key,  $m$  is the dimension of the public key,  $q$  is the modulus, and  $\chi$  is the noise distribution. The scheme is defined for message space  $\mathbb{Z}_2$ . Additionally, set  $l = (n + 1) \log q$ .

- $\text{Gen}(1^\lambda) \rightarrow (\text{ek}, \text{sk})$ : Sample  $s' \leftarrow \mathbb{Z}_q^n$  and set  $s = \begin{bmatrix} -s' \\ 1 \end{bmatrix} \in \mathbb{Z}_q^{n+1}$ .
- $\text{Enc}(\text{sk} \in \mathbb{Z}_q^{n+1}, m \in \mathbb{Z}_2) \rightarrow C \in \mathbb{Z}_q^{l \times (n+1)}$ : Sample  $A \leftarrow \mathbb{Z}_q^{l \times n}$  and  $e \leftarrow \chi^l$ . Define  $B = A \| As' + e$  and  $C = B + m \cdot G$  for a fixed gadget<sup>3</sup> matrix  $G \in \mathbb{Z}_q^{l \times (n+1)}$ .

3

- Note that by choice of  $B$ , we have that  $Bs = A(-s') + A(s') + e = e$  is an encryption of zero (with noise). Similarly,  $Cs = Bs + mGs = e + mGs$ .
- $G$  is a block matrix containing  $(n + 1)$  block column vectors of size  $\log q$  each. Each vector is  $g = (1, 2, 4, \dots, 2^{\log q - 1})$ . Concisely, we can define  $G = I_{n+1} \otimes g$  where  $\otimes$  is the Kronecker product.

- We also define a Flatten operation on the ciphertext that converts  $C \in \mathbb{Z}_q^{l \times (n+1)}$  to  $\tilde{C} \in \mathbb{Z}_q^{l \times l}$  by bit decomposing each element of  $C$  and replacing the element with its bit vector. This ensures that each element of this matrix is a bit. Looking ahead, this allows us to multiply ciphertexts without too much noise growth.
- $\text{Dec}(s, C)$ : Compute  $v = Cs$ . If  $\|v\|_\infty < q/4$ , i.e. each entry of  $v$  is less than  $q/4$ , output 0, else output 1. (Note the exact choice of threshold is somewhat arbitrary.)
- $\text{Eval}(+, C_1, C_2)$ : Output  $C = C_1 + C_2$ .
  - Notice that  $Cs = C_1s + C_2s = (m_1Gs + e_1) + (m_2G + e_2) = (m_1 + m_2)Gs + (e_1 + e_2)$ .
  - Thus if the original ciphertexts errors  $\|e_1\|_\infty, \|e_2\|_\infty \leq B$ , then the new ciphertext error is bounded by  $2B$ .
- $\text{Eval}(+, C_1, C_2)$ : Output  $C = \text{Flatten}(C_1) \times C_2$ .
  - We utilize the Flatten operation so that the dimensions match for matrix multiplication.
  - Notice that

$$\begin{aligned}
 Cs &= \text{Flatten}(C_1) \times C_2s \\
 &= \text{Flatten}(C_1) \times (m_2Gs + e_2) \\
 &= m_2(\text{Flatten}(C_1)G)s + \text{Flatten}(C_1)e_2 \\
 &= m_2(C_1s) + \text{Flatten}(C_1)e_2 \\
 &= m_2(m_1Gs + e_1) + \text{Flatten}(C_1)e_2 \\
 &= (m_1m_2)Gs + (m_2e_1 + \text{Flatten}(C_1)e_2).
 \end{aligned}$$

- Since each entry of  $\text{Flatten}(C_1)$  is in  $\{0, 1\}$ , each entry of  $\text{Flatten}(C_1)e_2$  is a subset sum of  $e_2 \in \mathbb{Z}_q^\ell$ . Thus if the original ciphertexts errors  $\|e_1\|_\infty, \|e_2\|_\infty \leq B$ , then the new ciphertext error is bounded by  $(1 + \ell)B$ .

Notice that as  $\text{Eval}$  is applied iteratively to ciphertexts in order to implement an arithmetic circuit of depth  $d$ , the noise will stay bounded by  $(\ell + 1)^d B$ . As long as the error stays below the threshold used by  $\text{Dec}$ , i.e.  $(\ell + 1)^d B < q/4$ , then correctness will hold. Recall that  $\ell = (n + 1) \log(q)$  where  $q$  is typically exponential in  $n$ . Then the error will be manageable as long as  $d$  is sublinear in  $n$ , e.g.  $d = n^{0.99}$ .

Further, notice that since swapping ordering of  $C_1$  vs  $C_2$  changes the noise growth of multiplication from  $m_2e_1 + \text{Flatten}(C_1)e_2$  to  $m_1e_2 + \text{Flatten}(C_2)e_1$ , we can optimize based on which ciphertext

started with more noise than the other. This can be leveraged to get polynomial noise growth instead of exponential.

Now, we have cheated slightly in the above—notice that our construction for  $\text{Eval}(+, C_1, C_2)$  implements addition mod  $q$ , but we actually wanted addition mod 2. One way to address this is to simply implement NAND, which on its own is a complete gate set that can implement any circuit:

- $\text{Eval}(\text{NAND}, C_1, C_2)$ : Output  $C = I - \text{Flatten}(C_1) \times C_2$ .

Altogether, we have achieved *leveled* FHE.

### 7.6.2 Bootstrapping

The goal of bootstrapping is to reset the ciphertext's noise back to a lower level after it has built up too much. This is done by letting the server re-encrypt the ciphertext *without* using the secret key  $s$ . How is this possible? Simply evaluate Dec homomorphically! In particular, let  $P_{\text{Dec}, C}$  be a circuit which on input  $s$  outputs  $m = \text{Dec}(s, C)$ . Let  $s_1, s_2, \dots, s_\ell \in \{0, 1\}$  be  $s$  written in binary. We will set the evaluation key  $ek = (ek_1, \dots, ek_\ell)$  for  $ek_i = \text{Enc}(s, s_i)$ , i.e. it gives us an encrypted copy of the secret key. Then whenever we need to reduce the noise of a ciphertext  $C$ , we can compute  $P_{\text{Dec}, C}$  and run  $\text{Eval}(P_{\text{Dec}, C}, ek_1 \dots ek_\ell)$  to get a fresh ciphertext  $\hat{C}$ . Notice that the noise of  $\hat{C}$  only depends on the noise of the input ciphertexts  $ek_i$ , which are fresh, and the depth  $d_P$  of the circuit  $P_{\text{Dec}, C}$ , which is independent of the noise of  $C$ . Thus  $\hat{C}$  will have noise bounded by  $(\ell + 1)^{d_P} B$ . As long as our leveled scheme supports circuits of depth  $d_P + 1$ , we can achieve arbitrary depth by bootstrapping after each operation. Of course, we'd like to avoid this as much as possible since doing the decryption operation homomorphically is expensive.

In order for this approach to work, security needs to be maintained even though we're releasing encryptions of the secret key. This is known as *circular security*. There are no known attacks on circular security for GSW and other commonly used leveled FHE constructions. However, they have not been proven to be circularly secure, and there are encryption schemes for which circular security is known to not hold for certain cycle lengths. An alternative is using a new secret key to encrypt the old secret key.

## 7.7 CCA-2 Secure Encryption from IBE

Finally, we will see how to utilize CPA secure IBE to construct not only CCA-1 but CCA-2 secure encryption. Let  $(G, K, E, D)$  be a CPA secure IBE scheme and  $(\text{Gen}_{\text{sig}}, \text{Sign}, \text{Verify})$  be a digital signature



scheme. Then construct a CCA-2 secure encryption scheme as follows:

- $\text{KeyGen}(1^n)$ :
  1. Compute  $(pk, sk) \leftarrow G(1^n)$ .
  2. Output  $pp = pk$  and  $msk = sk$ .
- $\text{Enc}(pk, m)$ :
  1. **Compute**  $(vk_s, sk_s) \leftarrow \text{Gen}_{\text{sig}}(1^n)$ .
  2. **Let**  $id = vk_s$ .
  3. Let  $c = E(pp, id, m)$ .
  4. **Compute**  $\sigma = \text{Sign}(sk_s, c)$ .
  5. **Output**  $(id, c, \sigma)$ .
- $\text{Dec}(sk, (id, c, \sigma))$ :
  1. **If**  $\text{Verify}(id, c, \sigma) = 0$  **then abort**.
  2. Compute  $sk_{id} \leftarrow K(sk, id)$ .
  3. Output  $D(sk_{id}, c)$ .

The bolded lines are the ones that have been changed compared to the CCA-1 construction. The changes restrict you to only generating ciphertexts for  $id$ 's you sampled yourself. Note that for this construction we need slightly stronger forgery protection for the digital signature scheme than we've considered previously. Namely, in the forgery security game the attacker is now allowed to output  $(m^*, \sigma^*)$  as long as the tuple is fresh, even if  $m^*$  on its own is not. Notice that this stronger notion holds for any deterministic signature scheme, including the BLS scheme we saw. It is also sufficient for our purposes here to utilize a one-time signature scheme.



# 8

## Proving Computation Integrity

### 8.1 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.<sup>1</sup>

<sup>1</sup> Formally, they can be modeled as interactive PPT Turing Machines.

### 8.2 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  $\mathcal{V}$  runs in polynomial time in its input length, and both can flip coins, such that:

- *Completeness:*  $\forall x \in L$

$$\Pr_{\mathcal{P}, \mathcal{V}} [\text{Output}_{\mathcal{V}}(\mathcal{P}(x) \leftrightarrow \mathcal{V}(x)) = 1] = 1,$$

- *Soundness:*  $\forall x \notin L, \forall \mathcal{P}^* \text{ (unbounded)}$

$$\Pr_{\mathcal{V}} [\text{Output}_{\mathcal{V}}(\mathcal{P}^*(x) \leftrightarrow \mathcal{V}(x)) = 1] < \text{negl}(|x|),$$

where  $\text{Output}_{\mathcal{V}}(\mathcal{P}(x) \leftrightarrow \mathcal{V}(x))$  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.

We can also consider other variants of this definition, e.g. imperfect completeness.

To understand the above definition, let's consider two languages over a pair of graphs  $G_0$  and  $G_1$ :

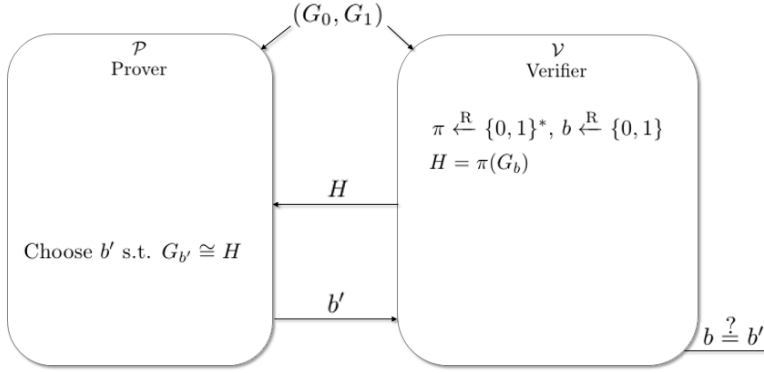
1. **Graph Isomorphism (GI):** We say that two graphs  $G_0$  and  $G_1$  are isomorphic, denoted  $G_0 \cong G_1$ , if  $\exists$  an isomorphism  $f : V(G_0) \rightarrow V(G_1)$  s.t.  $(u, v) \in E(G_0)$  iff  $(f(u), f(v)) \in E(G_1)$ , where  $V(G)$  and  $E(G)$  are the vertex and edge sets of some graph  $G$ . Let  $GI = \{(G_0, G_1) \mid G_0 \cong G_1\}$  be the language that consists of pairs of graphs that are isomorphic.
2. **Graph Non-Isomorphism (GNI):** On the other hand,  $G_0$  and  $G_1$  are said to be non-isomorphic,  $G_0 \not\cong G_1$ , if  $\nexists$  any such  $f$ , and let  $GNI = \{(G_0, G_1) \mid G_0 \not\cong G_1\}$  be the language that consists of pairs of graphs that are not isomorphic.

*Trivial Case of Graph Isomorphism (GI).* A prover can easily prove to a verifier that two graphs are isomorphic by directly providing the isomorphism  $f$  between them. The verifier can confirm the isomorphism in time polynomial in the size of the graphs (i.e., its input); hence we have perfect completeness. If the graphs are not isomorphic, no isomorphism exists, and the verifier always rejects; we have perfect soundness too. This proof was trivial, and we didn't even require (back-and-forth) interaction. We now look at a more interesting case of GNI. Moreover, looking ahead, we will see more interesting properties that we can ask of proof systems, like zero-knowledge, where this trivial proof system terribly fails, and we will revisit the GI problem to see how we can prove it with zero-knowledge.

*Interactive Proof for Graph Non-Isomorphism (GNI).* Unlike the case of GI, for GNI, there is no succinct (e.g., linear in the size of graphs) information that the prover can provide, and consequently, no "efficient" (polynomial time in the graphs) verification that the verifier can do. This is where the *power of interaction* comes in. In other words, since GNI is not believed to have short proofs, an *interactive* proof could offer the prover a mechanism to prove to a polynomially bounded verifier that two graphs are non-isomorphic. We will now describe an interactive proof system for GNI.

The intuition is simple. Consider a verifier that randomly re-names 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-time prover can figure this out. However, in case  $G_0$  and  $G_1$  are isomorphic, then the distributions resulting from 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<sup>2</sup>. More formally, given a claim  $(G_0, G_1) \in \text{GNI}$ , we define the following interactive proof system:



<sup>2</sup> This strategy is called soundness amplification by “sequential” repetition. Later, we might cover proof systems where we additionally consider “parallel” repetition to achieve different security properties.

- **Completeness:** If  $(G_0, G_1) \in \text{GNI}$ , 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  $(G_0, G_1) \notin \text{GNI}$ , then it is equiprobable that  $H$  is a random isomorphism of  $G_0$  as it is of  $G_1$ , and so  $\mathcal{P}$ 's guess for  $b'$  can be correct only with a probability  $\frac{1}{2}$ <sup>3</sup>. Repeating this protocol  $k$  times, with fresh verifier randomness each time, 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  $1 - \frac{1}{2^k}$ .

<sup>3</sup> A curious reader might notice that the challenge bit  $b$  sampled by  $\mathcal{V}$  is information-theoretically hidden from  $\mathcal{P}$  (hidden in  $H$ ) when  $\mathcal{P}$ 's claim is false. This is similar to what we saw in Hash Proof Systems before.

To conclude, the interactive proof system we described above enabled something that wasn't possible without interaction.

### 8.3 Zero Knowledge Proofs

We saw a crucial difference between GI and GNI: in GI, the prover already holds a succinct proof to back its claim, we call this a “witness”, while in GNI, no such succinct proof exists (i.e., there is nothing that the prover can directly send to the verifier to back its claim). From this point onwards, we exclusively focus on the languages of the first kind, i.e., where a witness for the claim exists; these languages cover a vast majority of the use-cases of verifiable computation, and are formalized as follows:

**Definition 8.2. (NP-Verifier)** A language  $L$  has an NP-verifier if  $\exists$  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:*  $\forall x \notin L$ , and  $\forall$  purported proof  $\pi$ , 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.

*Keeping the witness private.* The goal of a proof system is for the verifier to learn if the prover's claim is valid or not. Let's focus on what a verifier actually learns at the end of its interaction with the prover. In the trivial GI proof system we saw above, the verifier learns the entire isomorphism — in other words, the verifier learns *everything* that the prover knew. This is too much leakage. Imagine the prover holding some secret or valuable information (e.g., its secret key) which is leaked to the verifier. This is not desirable. We want the verifier to learn only the validity of the claim, and nothing more. This is where the notion of zero-knowledge comes in. For a proof system for a language with an NP-verifier, this translates to the verifier not learning the witness from the prover.

We now revisit the GI problem<sup>4</sup> for which an NP-verifier exists, as we saw earlier. Later, we will consider NP-complete languages like graph 3-coloring — giving us proof systems for all of NP.

<sup>4</sup> GI is not NP-complete.

*Hiding witness for Graph Isomorphism.* We will build the ideas for our proof system with zero-knowledge gradually by iterating through a series of straw-man approaches. On the way, we will formally define zero knowledge.

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. The prover doesn't want to reveal the isomorphism,  $w : V(G_0) \rightarrow V(G_1)$ , that they claim to have. The prover is comfortable however giving us a "scrambled" version,  $\phi$ , 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  $\pi$  is a privately chosen random permutation of  $|V| = |V(G_0)| = |V(G_1)|$  vertices. Since  $\pi$  renames vertices completely randomly, it scrambles what  $w$  is doing entirely and  $\phi$  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. Let's look at why this is still a good starting point.

If we want to be convinced that  $\phi$  really is of the form  $\pi \circ w$ , thus containing  $w$  in its definition, and isn't just a completely random permutation, 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

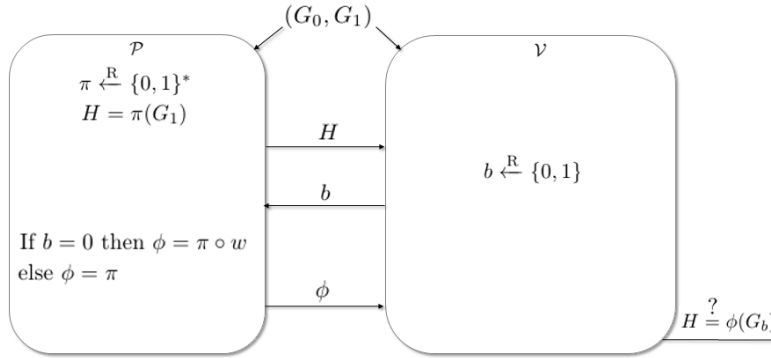
$w(G_0) = G_1$ ). Note that we started with a mapping on input  $G_0$  and ended with a mapping on input  $G_1$ . With an isomorphism, one could get from one graph to the other seamlessly; if the prover *really* has the isomorphism it claims to have, then it should have no problem displaying this ability. So, what if we force the prover to give us  $H = \pi(G_1)$  just after randomly choosing its  $\pi$  and then let it show us its ability to go from  $G_1$  to  $G_0$  with ease: give us a  $\phi$  so that  $\phi(G_0) = \pi(G_1) = H$ . The only way the prover can give a mapping that jumps from  $G_0$  to  $G_1$  is if they know an isomorphism; in fact, if the prover could find a  $\phi$  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  $\phi$  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. Importantly, the prover didn't directly tell us  $w$ , so we are headed in the right direction.

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  $\phi$  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, if we look closely, we realize that  $H = \pi(G_1) = \pi'(G_0)$ , for some  $\pi'$ , 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  $\pi'$ , 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. To our annoyance, the prover can easily fool this test. 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 we want  $H$  to equal  $\phi(G_0)$  or  $\phi(G_1)$ . In our interaction, the prover must first provide  $H = \pi(G_1)$  before we let them know which we want. By sending  $H$ , the prover locks itself into a commitment to either  $G_0$  or  $G_1$  if it is cheating, but if not, then it can easily move between the two graphs. A prover only has 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.

Therefore, 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!



- **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  $\phi$  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.

We have just shown that what we have so far is an interactive proof system. We now think of how the notion of zero-knowledge can be formalized here.

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  $\phi$  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  $\phi$  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  $(H, b, \phi)$  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 gained any knowledge about  $w$  since we didn't see anything we couldn't have manufactured on our own, yet we are entirely convinced that  $(G_0, G_1) \in \text{GI}$  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, as we do next.

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 prover. 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. We will first look at a weaker notion of zero-knowledge, called *Honest Verifier Zero Knowledge* (HVZK), where we only require that an *honest* verifier (follows the protocol steps) does not learn anything from the prover. We will then move on to the stronger notion of *Zero Knowledge* (ZK), where we extend this to all verifiers, including malicious verifiers.

**Definition 8.3. (Honest Verifier Zero Knowledge Proof [HVZK])**

$(\mathcal{P}, \mathcal{V})$  is a (perfect) HVZK proof system for a language  $L$  w.r.t. witness relation  $R$  if  $\exists$  a PPT machine  $\mathcal{S}$  (called the simulator) s.t.  $\forall x \in L$ ,  $\forall w \in R(x)$ , the following distributions are (identical) indistinguishable:

$$\{\text{View}_{\mathcal{V}}(\mathcal{P}(x, w) \leftrightarrow \mathcal{V}(x))\} \approx \{\mathcal{S}(x)\}$$

where  $\text{View}_{\mathcal{V}}(\mathcal{P}(x, w) \leftrightarrow \mathcal{V}(x))$  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.

**Remark 8.2.** *In the above definition, the order of quantifiers is quite important. We cannot change it to:  $\forall x \in L, \forall w \in R(x), \exists$  a PPT machine  $S$ . This is because the definition would be trivially satisfied by hardcoding the witness  $w$  in the simulator  $S$ .*

To prove HVZK property of the GI proof system we described earlier, we now construct a simulator  $S$ , with input  $G_0, G_1$ , as follows:

1. Sample  $b \in \{0, 1\}$  uniformly at random.
2. Sample a random permutation  $\sigma$  of the vertices.
3. Set  $H \leftarrow \sigma(G_b)$ .
4. Output  $(H, b, \sigma)$ .

It is straightforward to see that this simulator produces the same distribution as the real interaction between the prover and the verifier. This is because  $H = \sigma(G_b) = \sigma'(G_{1-b})$ , i.e.,  $H$  is a random permutation of both  $G_0$  and  $G_1$ .

To recap: 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 to 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' requirement 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.

## 8.4 Zero-Knowledge for Graph Isomorphism

In this section, we construct our final zero-knowledge interactive proof system for GI where we don't have to assume an honest verifier for zero knowledge to hold. The proof system construction is exactly the same as the one we saw earlier. What changes is the definition of zero knowledge, and therefore, the simulator.

**Definition 8.4. (Zero Knowledge Proof [ZK])**  $(\mathcal{P}, \mathcal{V})$  is a (perfect) ZK proof system for a language  $L$  w.r.t. witness relation  $R$  if  $\forall$  PPT machines  $\mathcal{V}^*$ ,  $\exists$  a PPT machine  $\mathcal{S}$  (called the simulator) s.t.  $\forall x \in L, \forall w \in R(x)$ , the following distributions are (identical) indistinguishable:

$$\{\text{View}_{\mathcal{V}^*}(\mathcal{P}(x, w) \leftrightarrow \mathcal{V}^*(x))\} \approx \{\mathcal{S}(x)\}$$

where  $\text{View}_{\mathcal{V}^*}(\mathcal{P}(x, w) \leftrightarrow \mathcal{V}^*(x))$  is the random coins of  $\mathcal{V}^*$  and all the messages  $\mathcal{V}^*$  saw.

**Remark 8.3.** Note that the order of quantifiers matters again. The definition would be stronger if we switch the order to:  $\exists$  a PPT machine  $\mathcal{S}$  (called the simulator) s.t.  $\forall$  PPT machines  $\mathcal{V}^*$ . This is because the same simulator would need to work for all possible efficient verifiers. Interestingly, the simulator we construct below for GI satisfies this stronger definition too. In fact, most simulators we know work for all verifiers (i.e. black-box simulators). It wasn't until 2008 that Boaz Barak showed that we can also construct non-black-box simulators.

Recall our protocol for graph isomorphism: the interaction is  $\mathcal{P}(x, w) \leftrightarrow \mathcal{V}(x)$  where  $x$  represents graphs  $G_0 = (V, E_0)$  and  $G_1 = (V, E_1)$  and  $w$  represents a permutation on  $V$  such that  $w(G_0) = G_1$ .

1.  $\mathcal{P}$  samples a random permutation  $\sigma : V \rightarrow V$  and sends the graph  $H = \sigma(G_1)$  to  $\mathcal{V}$ .
2.  $\mathcal{V}$  samples a random bit  $b$  and sends it to  $\mathcal{P}$ .
3. If  $b = 1$ , then  $\mathcal{P}$  defines a permutation  $\tau$  to be  $\sigma$ . If  $b = 0$ , then instead  $\tau = \sigma \circ w$ .  $\mathcal{P}$  then sends  $\tau$  to  $\mathcal{V}$ .
4.  $\mathcal{V}$  verifies that  $\tau(G_b) = H$  and accepts if so.

The reason the simulator for HVZK doesn't work anymore is because a malicious verifier  $\mathcal{V}^*$  could pick its bit  $b$  from a biased distribution (e.g.  $b$  can be a function of  $H$  seen by  $\mathcal{V}^*$ ).

For zero knowledge, consider the following simulator<sup>5</sup>  $\mathcal{S}$  with input  $G_0$  and  $G_1$  (with vertex set  $V$ ) and verifier  $\mathcal{V}^*$ :

1. For  $i = 1 \dots T$ ;  $T = \text{poly}(n)$ :
  - (a) Sample a bit  $b$  uniformly at random.
  - (b) Sample a permutation  $\sigma : V \rightarrow V$  uniformly at random
  - (c) Send  $H = \sigma(G_b)$  to  $\mathcal{V}^*$ .
  - (d) Receive  $b'$  from  $\mathcal{V}^*$ .
  - (e) If  $b = b'$ , then output  $(H, b, \sigma)$  and terminate. Otherwise, continue the loop.

<sup>5</sup> The simulator satisfies a stronger ZK property where the same simulator works for all  $\mathcal{V}^*$ . Refer to the remark above for more details.

2. Output  $\perp$ .

We construct a sequence of hybrids to prove zero-knowledge. Let  $H_0$  define the interaction between  $\mathcal{P}$  and  $\mathcal{V}^*$ :  $\mathcal{P}(x = (G_0, G_1), w) \leftrightarrow \mathcal{V}^*(x)$ . Define  $H_1$  as follows:

1. For  $i = 1 \dots T$ :

- (a) Sample a bit  $b^*$  uniformly at random.
- (b) Run  $H_0$ , i.e.,  $\mathcal{P}(x = (G_0, G_1), w) \leftrightarrow \mathcal{V}^*(x)$ .
- (c) If  $b^* = 0$ , output  $\text{View}_{\mathcal{V}^*}(\mathcal{P}(x, w) \leftrightarrow \mathcal{V}^*(x))$ .
- (d) If  $b^* = 1$ , continue with the loop.

2. Output  $\perp$ .

The hybrid  $H_1$  produces identical distribution as  $H_0$  except if  $b^*$  in all the  $T$  iterations is 0; note that  $\mathcal{P}(x = (G_0, G_1), w) \leftrightarrow \mathcal{V}^*(x)$  doesn't depend on  $b^*$ . This happens with probability at most  $1/2^T$ . Next, we define  $H_2$  where we change the logic of which transcript is thrown away. In each iteration, if  $b^* = b$ , where  $b$  is part of the transcript  $\mathcal{P}(x = (G_0, G_1), w) \leftrightarrow \mathcal{V}^*(x)$ , we output the transcript; otherwise, we continue with the loop. Note again that  $b^*$  is still not used in any of the transcript (interaction). Therefore, distribution produced by  $H_2$  is identical to  $H_1$  because the interaction hasn't changed at all. Finally, we construct our last hybrid  $H_3$  where the simulator  $S$  is run. The distribution generated by  $H_3$  is identical to  $H_2$  by same argument we used for the HVZK simulator.

**Efficient Provers.** So far, we have considered unbounded provers, but 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. We will now assume provers are also bounded to be *efficient*. Note that if the prover in our GI proof system already holds the isomorphism, then generating the proof only takes polynomial time, and it is easy to see that it satisfies the definition below.

**Definition 8.5** (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  $\forall x \notin L$ ,

$$\begin{aligned} & \Pr[\tilde{P}^*(x, w) \leftrightarrow \tilde{V}(x) \text{ rejects}] \\ &= 1 - \Pr[\forall 1 \leq i \leq k, P_i^*(x, w) \leftrightarrow V(x) \text{ accepts}] \\ &= 1 - \prod_{i=1}^k \Pr[P_i^*(x, w) \leftrightarrow V(x) \text{ accepts}] \\ &\geq 1 - \frac{1}{2^k} \end{aligned}$$

for any prover  $\tilde{P}^* = (P_1^*, \dots, P_k^*)$  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  $x \in L, w \in R_L(x), z \in \{0, 1\}^*$ ,

$$\{\text{View}_{\tilde{V}^*}(\tilde{P}(x, w) \leftrightarrow \tilde{V}^*(x, z))\} \simeq_c \{\tilde{S}^{\tilde{V}^*}(x, z)\}.$$

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 attacker 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  $(P, V)$ .

Similar to what we saw in previous definitions of zero-knowledge, the order of the quantifiers in item 4 matters. If we quantify over  $x$  and  $w$  before quantifying over the simulator, then we could hard-code  $x$  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  $z \in \{0, 1\}^*$ ,

$$\{\text{View}_{V^*}(P(x, w) \leftrightarrow V^*(x, z))\} \simeq_c \{S_{x,w}^{V^*}(x, z)\}$$

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 Zero-Knowledge for NP

An  $n$ -coloring of a graph  $G = (A, E)$  is a function  $c : A \rightarrow \{1, \dots, n\}$  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  $\phi$  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  $\phi$  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  $\phi$  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.

### 8.5.1 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 8.6.** 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.  $\square$

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.

### 8.5.2 3COL Protocol

Below we describe the protocol  $P(x, z) \leftrightarrow V(x)$ , where  $x$  describes a graph  $G = (\{1, \dots, n\}, E)$  and  $z$  describes a 3-coloring  $c$ :

1.  $P$  picks a random permutation  $\pi : \{1, 2, 3\} \rightarrow \{1, 2, 3\}$  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  $\alpha_1, \alpha_2, \dots, \alpha_n$  to  $V$ .
2.  $V$  uniformly samples an edge  $e = (i, j) \in E$  and sends it to  $P$ .
3.  $P$  opens  $\alpha_i$  and  $\alpha_j$ .
4.  $V$  will accept only if it received valid decommitments for  $\alpha_i$  and  $\alpha_j$ , 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  $r_1, \dots, r_n$  and set  $\alpha_i = C(c_i, r_i)$  for all  $i$ .
4. Let  $e' \in E$  be the response of  $V^*$  upon receiving  $\alpha_1, \dots, \alpha_n$ .
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  $\alpha_1, \dots, \alpha_n, e$ , 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  $\alpha_k$  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 is 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.

## 8.6 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  $\sigma$  (trusted to be random by both). For example, they could derive  $\sigma$  by looking at sunspot patterns.

## 8.7 Definitions

**Definition 8.7** (NIZK Proof System). A NIZK proof system for input  $x$  in language  $L$ , with witness  $\omega$ , 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.**  $\forall x \in L, \forall \omega \in R_L(x)$ :

$$\Pr[\sigma \leftarrow K(1^k), \pi \leftarrow P(\sigma, x, \omega) : V(\sigma, x, \pi) = 1] = 1.$$

There is an alternate definition of Statistical Correctness, where the probability above is  $1 - \text{negl}(n)$  instead of 1. For this explanation, though, Completeness will be used.

**Non-Adaptive Soundness.**  $\forall x \notin L$ :

$$\Pr[\sigma \leftarrow K(1^k) : \exists \pi \text{st} V(\sigma, x, \pi) = 1] = \text{negl}(k).$$

If the value of  $\sigma$  that is picked is a “bad” value, then there does not exist a proof  $\pi$  for  $x$  and  $\sigma$ . The above definition is “non-adaptive”, because it does not allow a cheating prover to decide which statement to prove after seeing the randomness  $\sigma$ . We may also consider the stronger notion of “adaptive soundness”, where the prover is allowed to decide  $x$  after seeing  $\sigma$ :

**Adaptive Soundness.**

$$\Pr[\sigma \leftarrow K(1^k) : \exists (x, \pi) \text{st} x \notin L, V(\sigma, x, \pi) = 1] = \text{negl}(k).$$

**(Non-Adaptive) Zero-Knowledge.** There exists a PPT simulator  $S$  such that  $\forall x \in L, \omega \in R_L(x)$ , the two distributions are computationally indistinguishable:

- |  |   |
|--|---|
| 1. $\sigma \leftarrow K(1^k)$            | 1. $(\sigma, \pi) \leftarrow S(1^k, x)$ |
| 2. $\pi \leftarrow P(\sigma, x, \omega)$ |   |
| 3. Output $(\sigma, \pi)$                | 2. Output $(\sigma, \pi)$               |

That is, the simulator is allowed to generate the distribution of randomness  $\sigma$  together with  $\pi$ . Note that if we did not allow  $S$  to produce  $\sigma$ , this definition would be trivial (a verifier could convince himself by running the simulator, instead of interacting with  $P$ ). Allowing  $S$  to generate  $\sigma$  still keeps the definition zero-knowledge (since a verifier sees both  $(\sigma, \pi)$  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  $\sigma$ :

**(Adaptive) Zero-Knowledge.** There 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. $\sigma \leftarrow K(1^k)$  | 1. $(\sigma, \tau) \leftarrow S_1(1^k)$         |
| 2. $(x, \omega) \leftarrow \mathcal{A}(\sigma)$ , s.t. $(x, \omega) \in R_L$ | 2. $(x, \omega) \leftarrow \mathcal{A}(\sigma)$ |
| 3. $\pi \leftarrow P(\sigma, x, \omega)$                                     | 3. $\pi \leftarrow S_2(\sigma, x, \tau)$        |
| 4. Output $(\sigma, x, \pi)$   | 4. Output $(\sigma, x, \pi)$                    |

where  $\tau$  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  $(K', P', V')$  that is adaptively sound.*

*Proof.* For  $x_0 \notin L$ , let us call a particular  $\sigma$  “bad for  $x_0$ ” if there exists a false proof for  $x_0$  using randomness  $\sigma$ :  $\exists \pi \text{st} V(\sigma, x_0, \pi) = 1$ . By non-adaptive soundness of  $(K, P, V)$ , we have  $\Pr_{\sigma}[\sigma \text{ bad for } x_0] = \text{negl}(k)$ .

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  $\text{negl}(k) \leq 2^{-2^{P(k)}}$ . 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.  $\square$

## 8.8 NIZKs from Trapdoor Permutations

**Definition 8.8** (Trapdoor One-Way Permutation). *A trapdoor one-way permutation is a collection of one-way permutations  $\{f_i : D_i \rightarrow D_i\}_{i \in I}$  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.

### 8.8.1 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 8.9.** RSA defines the functions  $(G, F, A)$  as follows.

$$\begin{aligned} G(1^k) &= ((N, e), d) \text{ where } N = pq, \text{ for primes } p, q, \\ &\quad \gcd(e, \phi(N)) = 1 \\ &\quad d = e^{-1} \pmod{\phi(N)} \\ F_{N,e}(x) &= x^e \pmod{N} \\ A((N, e), y, d) &= y^d \pmod{N} \end{aligned}$$

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  $\phi$  is Euler's Totient, and when  $p, q$  are primes,  $\phi(pq) = (p-1)(q-1)$ . (That is,  $\phi$  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.

### 8.8.2 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 8.10** (NIZK in the Hidden-Bit Model). *A NIZK in the hidden-bit model for statement  $x$  (with witness  $\omega$ ) is efficient algorithms  $(K_H, P_H, V_H)$  such that:*

1.  $r \leftarrow K_H(1^k)$  generates the hidden random string ( $\ell$ -bits).
2.  $(I, \phi) \leftarrow P_H(r, x, \omega)$  generates the indices  $I \subseteq [\ell]$  to reveal, and the proof  $\phi$ .
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  $\phi$ .

*Which satisfies the completeness, soundness, and zero-knowledge properties as previously defined.*

The above definition is not necessarily very useful in its own right,

but it is helpful as a stepping stone toward a more useful construction.

**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  $\sigma$  in the normal model be of length  $k\ell$  and partition it into  $\ell$  blocks of  $k$ -bits each:  $\sigma = \sigma_1 \dots \sigma_\ell$ . 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  $r := r_1 \dots r_\ell$ .
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,  $\sigma_i$  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  $\alpha_i$  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:

$$\Pr_{\sigma}[\exists \text{ some } f \in \mathcal{F} \text{ s.t. } P^* \text{ can cheat}] \leq 2^k \cdot 2^{-2k} = 2^{-k}.$$

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”  $\sigma_i$  using either  $\alpha_i$  or  $\alpha'_i$  – which is a problem if  $B(\alpha_i) \neq B(\alpha'_i)$ .

To prove zero-knowledge, we construct a sequence of prover-hybrids. The hybrid  $H_0$  is the “honest” scenario. Differences from the previous hybrid are in red:

- 
- $H_0$  (normal model)
- 
1.  $\sigma_1 \dots \sigma_\ell = \sigma \xleftarrow{\$} \{0, 1\}^{k\ell}$
  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.  $r_i \leftarrow \{0, 1\} \forall i \in [\ell]$
  2.  $(f, f^{-1}) \leftarrow \mathcal{F}$
  3.  $\alpha_i \xleftarrow{\$} \{0, 1\}^k$  such that  $r_i = B(f^{-1}(\sigma_i)) \forall i \in [\ell]$
  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  $H_1$ , we sample  $\alpha_i$  uniformly at random and then generate  $\sigma_i$  (instead of sampling  $\sigma_i$  and then generating  $\alpha_i$ ). This induces an exactly identical distribution, since  $f$  is a permutation.

- 
- $H_2$
- 
1.  $r_i \xleftarrow{\$} \{0, 1\} \forall i \in [\ell]$
  2.  $(f, f^{-1}) \leftarrow \mathcal{F}$
  3.  $(I, \phi) \leftarrow P_H(r, x, \omega)$
  4.  $\alpha_i \xleftarrow{\$} B^{-1}(r_i) \forall i \in [\ell]$

5.  $\sigma_i = f(\alpha_i) \forall i \in [\ell]$
6. Output  $(\sigma, f, I, \{\alpha_i\}_{i \in I}, \phi)$

In  $H_2$ , we again switch the sampling order: first sample the (un-biased) bit  $r_i$ , then sample  $\alpha_i$  from the pre-image of  $r_i$  (which can be done efficiently by simply trying random  $\alpha_i$ '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.  $r_i \xleftarrow{\$} \{0, 1\} \forall i \in [\ell]$
3.  $\alpha_i \xleftarrow{\$} B^{-1}(r_i) \forall i \in [\ell]$
4.  $\sigma_i = f(\alpha_i) \forall i \in I$
5.  $\sigma_i \xleftarrow{\$} \{0, 1\}^k \forall i \notin I$
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  $\sigma_i$  honestly for  $i \in I$ , and output random  $\sigma_i$  for  $i \notin I$ . To argue that this is computational indistinguishable from  $H_2$ , first notice that for a fixed (known) bit  $r$ ,

$$\{f(B^{-1}(r))\} \text{IND} \{f(B^{-1}(\bar{r}))\} \quad (8.1)$$

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

$$\{f(B^{-1}(r))\} \text{IND} \mathcal{U}_k \quad (8.2)$$

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.  $\alpha_i \xleftarrow{\$} B^{-1}(r_i) \forall i \in I$
4.  $\sigma_i = f(\alpha_i) \forall i \in I$
5.  $\sigma_i \xleftarrow{\$} \{0, 1\}^k \forall i \notin I$
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. □

□

### 8.8.3 NIZK in the Hidden-Bit Model for Graph Hamiltonian

**Definition 8.11.** 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  $x_1, \dots, x_n \in V$  such that they are all distinct vertices, and  $\forall i \in \{1, \dots, n-1\} : (x_i, x_{i+1}) \in E$ ,  $(x_n, x_1) \in E$ .

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 8.12.** 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.

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.

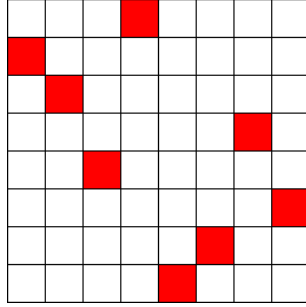


Figure 8.1: Cycle matrix.

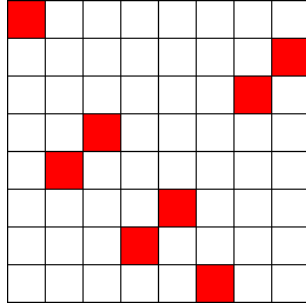


Figure 8.2: Permutation matrix.

**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  $\ell$  bits that the prover can read. The prover should be able to produce a proof  $\phi$  and choose a set  $I \subseteq \{1, 2, \dots, \ell\}$  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  $|V| = n$ . Note that the content of  $G$  is public. 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  $x_1, \dots, x_n$  in  $G$ , he can find a function  $\phi : V \rightarrow \{1, 2, \dots, n\}$  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  $\phi$  and also reveal entries of  $M_c$  corresponding to  $\phi(e)$  where  $e \notin E$ , so  $e \in V \times V \setminus E$ . 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  $M_c$ .

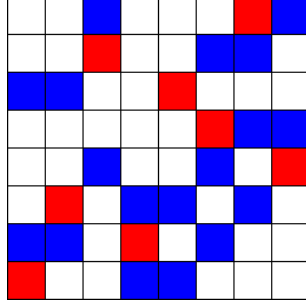


Figure 8.3: Graph matrix that includes a Hamiltonian cycle. Edges are blue/red and the cycle is red. White cells are non-edges.

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  $\phi$  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  $\phi$ .

*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  $\phi$  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  $\phi : V \rightarrow \{1, 2, \dots, n\}$  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  $\ell$  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 8.6 and 8.7.

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

$$1 - \left(1 - \frac{1}{3n^3}\right)^{n^4} \approx 1 - e^{-\frac{n}{3}} = 1 - \text{negl}(n).$$

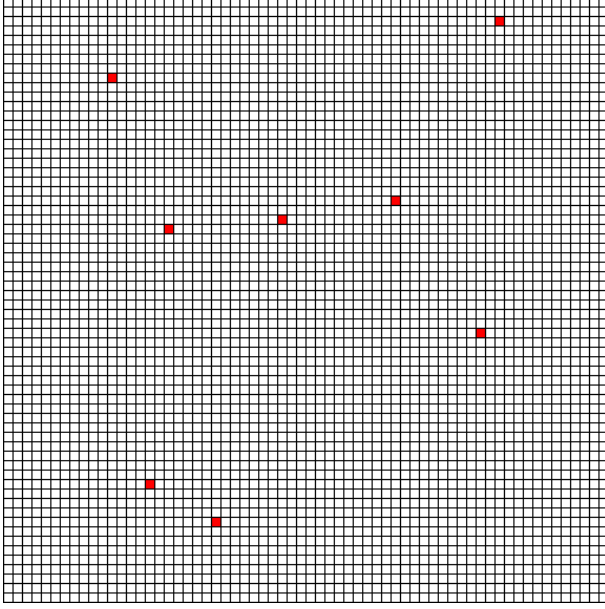


Figure 8.4: Matrix  $M$  which is  $n^2 \times n^2$  for  $n = 8$ .

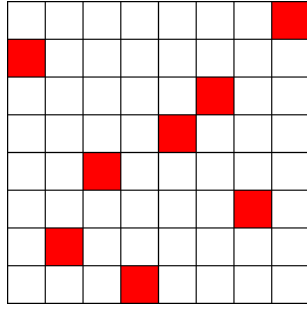


Figure 8.5: Matrix  $M_c$  which is  $n \times n$  for  $n = 8$ . The construction worked, because  $M_c$  is a cycle matrix.

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 construction 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  $\phi$  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  $\phi$  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  $1 - e^{-\frac{n}{3}} = 1 - \text{negl}(n)$ .

*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.  $\square$

**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

$$x \in L \iff f(x) \in L^*.$$

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.  $\square$

**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.  $\square$

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  $\ell$  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 8.6 and 8.7.

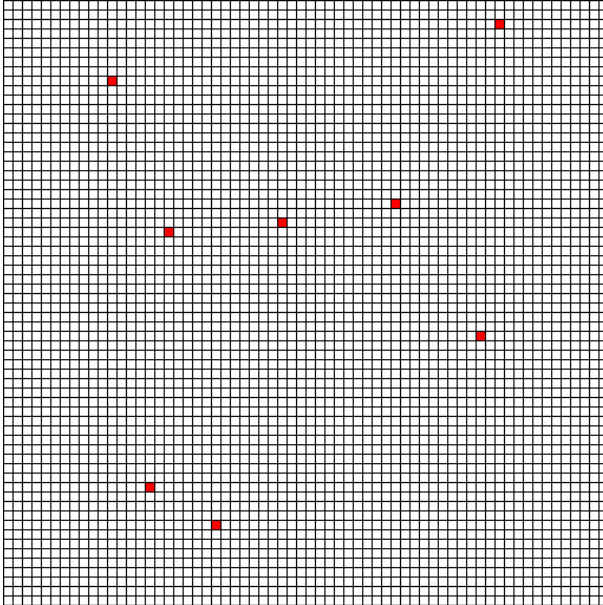


Figure 8.6: Matrix  $M$  which is  $n^2 \times n^2$  for  $n = 8$ .

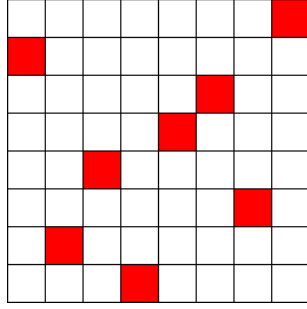


Figure 8.7: Matrix  $M_c$  which is  $n \times n$  for  $n = 8$ . The construction worked, because  $M_c$  is a cycle matrix.

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

$$1 - \left(1 - \frac{1}{3n^3}\right)^{n^4} \approx 1 - e^{-\frac{n}{3}} = 1 - \text{negl}(n).$$

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  $\phi$  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  $\phi$  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  $1 - e^{-\frac{n}{3}} = 1 - \text{negl}(n)$ .

*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.  $\square$



**Theorem 8.7.** *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

$$x \in L \iff f(x) \in L^*.$$

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.  $\square$

**Theorem 8.8.** *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.  $\square$

## 8.9 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.*

$$L = \left\{ ((G_0, G_1), (G'_0, G'_1)) \mid G_0 \simeq G_1 \vee G'_0 \simeq G'_1 \right\}$$

**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:

$$\text{View}_{V^*}(P(x, w_1) \leftrightarrow V^*(x, z)) \simeq_c \text{View}_{V^*}(P(x, w_2) \leftrightarrow V^*(x, z)).$$

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:

$$\left| \Pr \left[ \begin{array}{l} \sigma \leftarrow K(1^\kappa), \\ (\{x_i, w_i\}_{i \in [q]}, \text{state}) \leftarrow A_1(\sigma), \\ \text{such that } \forall i \in [q], (x_i, w_i) \in R \\ \forall i \in [q], \pi_i \leftarrow P(\sigma, x_i, w_i); \\ A_2(\text{state}, \{\pi_i\}_{i \in [q]}) = 1 \end{array} \right] - \Pr \left[ \begin{array}{l} (\sigma, \tau) \leftarrow \mathcal{S}_1(1^\kappa), \\ (\{x_i, w_i\}_{i \in [q]}, \text{state}) \leftarrow A_1(\sigma), \\ \text{such that } \forall i \in [q], (x_i, w_i) \in R \\ \forall i \in [q], \pi_i \leftarrow \mathcal{S}_2(\sigma, x_i, \tau); \\ A_2(\text{state}, \{\pi_i\}_{i \in [q]}) = 1 \end{array} \right] \right| \leq \text{negl}(\kappa).$$

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  $2\kappa$  bit string. To prove  $x \in L$  the prover  $P'$  proves that  $\exists(w, s)$  such that either  $(x, w) \in R$  or  $y = g(s)$ .

*Answer* Construction:

$$\begin{aligned} & \sigma \leftarrow K(1^\kappa) \\ K'(1^n) : & y \leftarrow \{0, 1\}^{2\kappa} \\ & \text{Output } \sigma' = (\sigma, y) \end{aligned}$$

$$\begin{aligned} P'(\sigma' = (\sigma, y), x, w) : & \pi \leftarrow P(\sigma, (x, y), w) \text{ if } x \in L \text{ or } \exists s \text{ s.t. } g(s) = y \\ & \text{Output } \pi \end{aligned}$$

$$V'(\sigma', x, \pi) : \text{Output } V(\sigma, x, \pi)$$

The completeness of this systems follows from the completeness of  $(K, P, V)$ . The soundness is based on the PRG  $g$ . Since  $y$  is a randomly chosen string, for almost all possible choice of  $y$ , it is not in the range of  $g$ , and thus  $x \in L$  with overwhelming probability. Therefore, the soundness property also follows from that of  $(K, P, V)$

To prove zero-knowledge property, we construct the simulator  $S_1, S_2$  as follows:

$$\begin{aligned} & \sigma \leftarrow K(1^\kappa) \\ S_1(1^\kappa) : & \quad s \leftarrow 0, 1^n, y' = g(s) \quad , \quad S_2(\sigma' = (\sigma, y'), x, \tau = s) : \quad \text{Output } P(\sigma, (x, y'), s) \\ & \quad \text{Output } ((\sigma, y'), s) \end{aligned}$$

We show that the proofs from  $S_2$  are indistinguishable from those from  $P'$  via hybrid argument. Consider following sequence of the proofs  $(\{\pi_i, i \in [q]\})$ .

$$\begin{array}{lll} \sigma \leftarrow K(1^n) & \sigma \leftarrow K(1^n) & \sigma \leftarrow K(1^n) \\ \mathcal{H}_0 : y \leftarrow \{0, 1\}^{2n} & \mathcal{H}_1 : s \leftarrow \{0, 1\}^n, y = g(s) & \mathcal{H}_2 : s \leftarrow \{0, 1\}^n, y = g(s) \\ \{\pi_i \leftarrow P(\sigma, (x_i, y), w_i)\} & \{\pi_i \leftarrow P(\sigma, (x_i, y), w_i)\} & \pi_1 \leftarrow P(\sigma, (x_i, y), s) \\ & & \{\pi_i \leftarrow P(\sigma, (x_i, y), w_i) \mid i \in [1, q]\} \end{array}$$

Since  $g$  is PRF, it can be shown that  $\mathcal{H}_0 \cong \mathcal{H}_1$  using Sunglass Lemma. Next, consider an intermediate hybrid  $\mathcal{H}_{1.5}$  where everything is the same as  $\mathcal{H}_1$ , but the generation of  $\pi_1$  is simulated by  $S_1, S_2$ .  $\mathcal{H}_1$  and  $\mathcal{H}_2$  are independently indistinguishable from  $\mathcal{H}_{1.5}$ ,  $\mathcal{H}_1 \cong \mathcal{H}_{1.5} \cong \mathcal{H}_2$ . This process can be extended the argument to  $\mathcal{H}_3, \mathcal{H}_4, \dots$  where the next  $\pi$ s are generated with  $s$  in the same way. Therefore, by the hybrid argument,  $\mathcal{H}_1$  is indistinguishable from a sequence where all proofs are generated with  $s$ . Thus,  $S_1, S_2$  simulates the proofs by  $P'$ .

### 8.9.1 Naor-Yung Construction

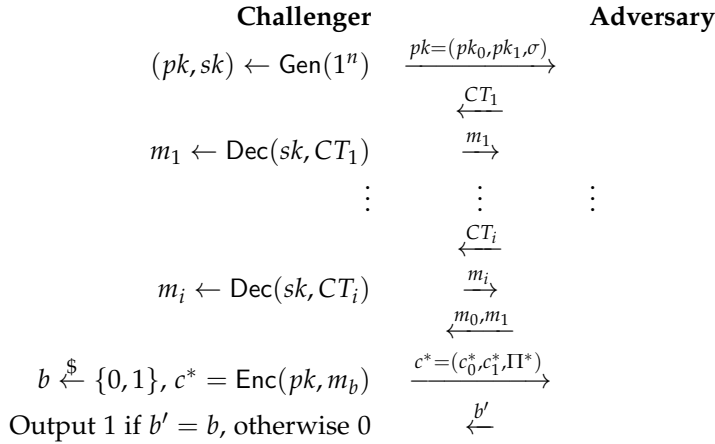
The Naor-Yung construction relies on a semantically-secure public-key encryption scheme  $(\text{Gen}, \text{Enc}, \text{Dec})$  and an adaptively-secure NIZK proof system  $(K, P, V)$  to construct a public-key encryption scheme  $(\text{Gen}_{\text{cca1}}, \text{Enc}_{\text{cca1}}, \text{Dec}_{\text{cca1}})$  that is CCA1 secure. The scheme is defined as follows:

- $\text{Gen}_{\text{cca1}}(1^n) :$ 
  1.  $(pk_0, sk_0) \leftarrow \text{Gen}(1^n)$
  2.  $(pk_1, sk_1) \leftarrow \text{Gen}(1^n)$

3.  $\sigma \leftarrow K(1^n)$
  4. Output  $pk = (pk_0, pk_1, \sigma)$ ,  $sk = sk_0$
- $\text{Enc}_{\text{cca1}}(pk = (pk_0, pk_1, \sigma), m) :$ 
    1.  $\{c_b \leftarrow \text{Enc}(pk_b, m; r_b)\}_{b \in \{0,1\}}$
    2.  $\Pi \leftarrow P(\sigma, (c_0, c_1), (m, r_0, r_1))$
    3. Output  $(c_0, c_1, \Pi)$
  - $\text{Dec}_{\text{cca1}}(sk = sk_0, CT = (c_0, c_1, \Pi)) :$ 
    1. If  $V(\sigma, (c_0, c_1), \Pi) = 1$ : output  $\text{Dec}(sk_0, c_0)$
    2. Else: output  $\perp$

**Theorem 8.9.** Assuming  $(\text{Gen}, \text{Enc}, \text{Dec})$  is a semantically-secure encryption scheme and  $(K, P, V)$  is an adaptively-secure NIZK proof system, then  $(\text{Gen}_{\text{cca1}}, \text{Enc}_{\text{cca1}}, \text{Dec}_{\text{cca1}})$  is secure against non-adaptive chosen-ciphertext attacks.

*Proof.* Recall the definition of CCA1 security game:



Starting from this Game 0, we can derive Game 1 to 4, so that it is clear that at Game 4, the adversary does not receive any information about  $b$  from  $c^*$ , and hence it is impossible to correctly guess  $b$  with a non-negligible advantage. The intuition here is to replace  $c_0^*, c_1^*, \Pi^*$  step by step. An important lemma that we make use of in the construction is:

**Lemma 8.1.** For  $\forall CT = (c_0, c_1, \Pi)$ , we have that if  $V(\sigma, (c_0, c_1), \Pi) = 1$ ,  $\text{Dec}(sk_0, c_0) = \text{Dec}(sk_1, c_1)$ .

The lemma is a direct result of the soundness guarantee of the NICK proof system. Now we are ready to construct the games:

- Game 1: instead of  $(P, V)$ , its simulator is used to provide the proof  $\Pi^*$ . Specifically, we have  $(\sigma, \tau) \leftarrow S_1(1^n)$  and  $\Pi^* \leftarrow S_2(\tau, (c_0^*, c_1^*))$ . Now  $\Pi^*$  does not contain information about  $b$ .
- Game 2: note that only  $sk_0$  is used to answer all the decryption queries by the adversary, as a result, we can set  $c_1^* \leftarrow \text{Enc}(pk_1, 0^{|m_1|})$ . This works because of the indistinguishability of the public-key encryption scheme. Now  $c_1^*$  does not contain information about  $b$ .
- Game 3: using Lemma 8.1, we can use  $sk_1$  instead of  $sk_0$  to answer all the decryption queries by the adversary.
- Game 4: now, since only  $sk_1$  is used to answer all the decryption queries by the adversary, we can set  $c_0^* \leftarrow \text{Enc}(pk_0, 0^{|m_0|})$ . This works because of the indistinguishability of the public-key encryption scheme. Now  $c_0^*$  does not contain information about  $b$  either.

## 8.10 zkSNARKs

In this section, we will overview the fundamentals of zkSNARKs, or Zero-Knowledge Succinct Non-interactive ARguments of Knowledge.

### 8.10.1 Preliminaries

Before diving into zkSNARKs, let's understand the basic framework we're working with. In cryptography, we often deal with statements of the form "I know a secret value that satisfies some property." For instance, we might want to prove that we know the private key corresponding to a public key, or that we know a solution to a Sudoku puzzle, or that we know a valid password for an account.

To formalize these statements, we use what's called a binary relation  $R$ . This relation takes two inputs: the public statement  $x$  (like a Sudoku puzzle) and the secret witness  $w$  (like the solution).  $R$  is an efficiently computable binary relation that outputs 1 if the witness  $w$  is valid for statement  $x$ , and 0 otherwise.

For an NP language  $\mathcal{L}$ , we can say that  $x \in \mathcal{L}$  if and only if there exists a witness  $w$  such that  $R(x, w) = 1$  ( $R$  being the corresponding to  $\mathcal{L}$  relation). Conversely,  $x \notin \mathcal{L}$  if and only if there does not exist any witness  $w$  such that  $R(x, w) = 1$ . A crucial property is that while finding a valid witness  $w$  may be computationally hard (like solving a Sudoku puzzle), verifying the relation  $R(x, w) = 1$  is always efficient (like checking if a Sudoku solution is valid). This verification can be done in polynomial time.

This framework allows us to express a wide variety of practical problems where we want to prove knowledge of a solution without

revealing the solution itself, which is exactly what zkSNARKs will help us achieve.

### 8.10.2 Properties of zkSNARKs

We now introduce the properties of zkSNARKs. We seek to use zk-SNARKs to prove that  $x \in L \iff R(x, w) = 1$ . Informally, if a prover has  $(x, w)$ , we must send a proof  $\pi$  such that the verifier, who has the instance  $x$ , can efficiently check that  $R(x, w) = 1$ . For the purposes of this lecture, we are focusing on non-interactive proof systems. This means that there is no back-and-forth communication between the prover and the verifier, and the verifier can verify the prover's statement in one shot.

Two of the properties of zkSNARKs are common to all proof systems: Correctness ensures that if the statement is true, the prover should always be able to convince the verifier. Soundness ensures that no cheating prover can convince the verifier about an invalid statement. However, a correct and sound proof system is not particularly interesting. Indeed, we could simply achieve this by sending the witness  $w$  from the prover to the verifier. We now discuss two properties that make zkSNARKs interesting: Succinctness and Zero-Knowledge. Succinctness requires that the proof  $\pi$  sent by the prover is significantly smaller than the witness  $w$ . Specifically, the proof size  $|\pi|$  must be bounded by  $\text{poly}(\lambda, \log(|w|))$ , where  $\lambda$  is the security parameter. This means the proof grows only (at most) polylogarithmically with the witness size. Zero-knowledge ensures that the proof should not reveal any information about the witness  $w$  beyond what can be deduced from the statement being proven. This property is what differentiates zkSNARKs from SNARKs.

The succinctness properties make SNARKs incredibly relevant, even for practical applications where we do not care about hiding the witness  $w$ . This is because the proof size is exponentially more efficient than directly sending the witness  $w$  from the prover to the verifier.

*Succinctness example.* Let's consider a practical example to illustrate the power of SNARKs' succinctness property. Suppose we have a 1 TB hard disk and want to prove to a verifier that  $\text{Hash}(\text{hard disk}) = x$  for some known hash value  $x$ . We have two options: Without SNARKs, we would need to send the entire 1 TB hard disk to the verifier, who then computes the hash themselves. With SNARKs, we can generate a succinct proof  $\pi$  (for example 1 KB) that proves knowledge of the hard disk contents whose hash equals  $x$ . Even in scenarios where we do not need to hide the hard disk contents (i.e.,

zero-knowledge is not required), the SNARK approach is dramatically more efficient in terms of communication complexity: sending a 1 KB proof versus transferring 1 TB of data. This lecture will primarily focus on achieving succinctness, as typically adding zero-knowledge properties is a relatively straightforward extension once the basic SNARK construction is understood.

*How to build SNARKs?* To construct SNARKs, we will follow these key steps. First, we need to find a “SNARK-friendly” representation model for NP languages. The example we will use is square span programs. Next, we must show that this model can capture all NP relations, using boolean circuits as an example. Then, we construct the SNARK using cryptographic techniques, specifically employing bilinear groups in our example construction. After that, we prove soundness. Finally, we add zero-knowledge properties. We will study the DFGK14 SNARK, which conceptually is in the SNARK family of the widely known Groth16 zkSNARK. DFGK14 which is conceptually simpler but employs the same fundamental cryptographic ideas.

### 8.10.3 Square Span Programs

We begin by introducing Square Span Programs (SSPs), which provide a “SNARK-friendly” representation for NP languages.

**Definition 8.13** (Square Span Program). *A square span program  $Q$  over a field  $\mathbb{F}$  consists of a size parameter  $m \in \mathbb{N}$ , a degree parameter  $d \in \mathbb{N}$ , and a set of polynomials  $\{v_0(x), v_1(x), \dots, v_m(x), t(x)\}$ . Each  $v_i(x)$  is a polynomial over  $\mathbb{F}$  of degree at most  $d$ , and  $t(x)$  is a polynomial over  $\mathbb{F}$  of degree exactly  $d$ .*

**Definition 8.14** (SSP Acceptance). *We say that a square span program  $Q$  accepts an input  $(a_1, \dots, a_\ell) \in \mathbb{F}^\ell$  if and only if there exist values  $a_{\ell+1}, \dots, a_m \in \mathbb{F}$  such that  $t(x)$  divides  $(v_0(x) + \sum_{i=1}^m a_i v_i(x))^2 - 1$ . In other words, there exists a quotient polynomial  $h(x)$  such that  $(v_0(x) + \sum_{i=1}^m a_i v_i(x))^2 - 1 = h(x)t(x)$ .*

The values  $a_1, \dots, a_\ell$  represent the input to our computation, while  $a_{\ell+1}, \dots, a_m$  serve as auxiliary values (similar to a witness in an NP relation). As we will see, this algebraic structure is particularly well-suited for constructing SNARKs. A key property of SSPs is their expressiveness: we can transform any boolean circuit into an equivalent square span program. This transformation will be our next focus.

## 8.10.4 From Boolean Circuits to SSPs

Consider the boolean circuit in Figure 8.8. We will transform this circuit into an SSP.

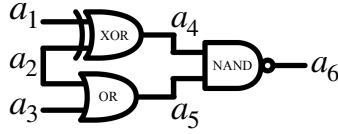


Figure 8.8: Boolean circuit example for SSP transformation

First, we express each gate type as a linear constraint. For a XOR gate, the output  $a_4$  of inputs  $a_1, a_2$  must satisfy  $a_1 + a_2 + a_4 \in \{0, 2\}$ . For an OR gate, the output  $a_5$  of inputs  $a_2, a_3$  must satisfy  $(1 - a_2) + (1 - a_3) - 2(1 - a_5) \in \{0, 1\}$ . For a NAND gate, the output  $a_6$  of inputs  $a_4, a_5$  must satisfy  $a_4 + a_5 - 2(1 - a_6) \in \{0, 1\}$ .

For the circuit to be satisfiable, all gate constraints must be satisfied, the output should be 1 or equivalently the constraint  $3(1 - a_6) \in \{0, 2\}$  must hold, and all boolean value constraints  $a_i \in \{0, 1\}$  for all  $i \in \{1, \dots, 6\}$  must be met. To standardize these constraints, we multiply all constraints involving  $\{0, 1\}$  by 2 to normalize ranges and combine constraints into a matrix form for  $a_1, \dots, a_6$ . Finally, we seek to unify the constraints into a larger matrix that encompasses all the constraints in the whole circuit, where each column represents a single boolean algebra constraint. The resulting constraint matrix  $M$  is below, with columns representing the XOR, OR, NAND, and output constraints, respectively:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & 0 & 2 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 4 & 2 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 4 & -3 & 0 & 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$

There is also a constant vector  $\vec{\delta}$  associated with these constraints:

$$\vec{\delta} = [0 \ 0 \ -4 \ 3 \ | \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$$

All constraints must evaluate to elements in  $\{0, 2\}^{10}$ .

*Generalizing to Arbitrary Circuits.* Let's now see how we can transform any arbitrary boolean circuit (with fan-in 2, fan-out 1 gates) into a Square Span Program. Consider a circuit with  $m$  wires and  $n$  gates. Our first task is to construct a matrix that captures all the constraints of our circuit. For each gate  $k$  in our circuit, we create a column vector  $\vec{v}_k = (v_{1k}, v_{2k}, \dots, v_{mk})^T$  that encodes the gate's constraints. These



constraints ensure the gate operates correctly—just as we saw with our XOR, OR, and NAND gates in the previous example.

After encoding all  $n$  gates, we add a special column for the output constraint, which takes the form  $(0, \dots, 0, -3)^T$ . The  $-3$  here is somewhat arbitrary; any field element different from 2 would work. We then augment our matrix with a diagonal matrix  $D$  where each diagonal entry is 2. This diagonal matrix serves to enforce that all our variables are boolean, a crucial requirement for circuit satisfaction.

To complete our constraint system, we need a constant vector  $\vec{\delta} = (\delta_1, \dots, \delta_n, 3, 0, \dots, 0)$  where each  $\delta_i$  represents the constant term for gate  $i$ . These constants are chosen from the set  $\{0, 2\}$ , with the exception of the output constraint's constant which is 3.

Let's now see how we can transform any arbitrary boolean circuit into a Square Span Program. Consider a circuit with  $m$  wires and  $n$  gates. The complete constraint system can be written as:

$$\begin{pmatrix} a_1 & a_2 & \cdots & a_m \end{pmatrix} \begin{pmatrix} v_{11} & v_{12} & \cdots & v_{1n} & 0 & 2 & 0 & \cdots & 0 \\ v_{21} & v_{22} & \cdots & v_{2n} & 0 & 0 & 2 & \cdots & 0 \\ v_{31} & v_{32} & \cdots & v_{3n} & 0 & 0 & 0 & \ddots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ v_{m1} & v_{m2} & \cdots & v_{mn} & -3 & 0 & 0 & \cdots & 2 \end{pmatrix} + \begin{pmatrix} \delta_1 & \delta_2 & \cdots & \delta_n & 3 & 0 & \cdots & 0 \end{pmatrix}$$

where  $(a_1, \dots, a_m)$  are the wire values (variables we solve for), the first  $n$  columns  $(v_{ij})$  represent the gate constraints, column  $n + 1$  is the output constraint  $(0, \dots, 0, -3)^T$ , the next  $m$  columns form the diagonal matrix  $D$  with 2's on the diagonal, and the constant vector  $(\delta_1, \dots, \delta_m)$  contains  $\delta_i \in \{0, 2\}$  for  $i \leq n$  (gate constraints),  $\delta_{n+1} = 3$  (output constraint), and  $\delta_i = 0$  for  $i > n + 1$  (boolean constraints).

Next, we convert these discrete constraints into a polynomial system. We choose distinct field elements  $x_1, \dots, x_d$  (where  $d = n + 1 + m$  is our total number of constraints) and use polynomial interpolation to create our SSP. For each row  $i$  of our matrix, we construct a polynomial  $v_i(x)$  that evaluates to the  $(i, j)$  entry when evaluated at point  $x_j$ . Similarly, we create a polynomial  $v_0(x)$  that interpolates our constant vector  $\vec{\delta}$ .

The target polynomial  $t(x)$  is defined as the product of all linear terms:

$$t(x) = \prod_{j=1}^d (x - x_j)$$

This polynomial is crucial because it “zeros out” exactly at our constraint points. The beauty of this construction is that it transforms our circuit satisfaction problem into an elegant polynomial divisibil-

ity question: the circuit is satisfiable if and only if there exist values  $(a_1, \dots, a_m)$  such that:

$$\left( v_0(x) + \sum_{i=1}^m a_i v_i(x) \right)^2 - 1 = h(x)t(x)$$

for some polynomial  $h(x)$ .

After converting to a polynomial system, our constraint matrix becomes:

$$\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{pmatrix}^T \begin{pmatrix} v_1(x_1) & v_1(x_2) & \cdots & v_1(x_n) & 0 & v_1(x_{n+2}) & 0 & \cdots & 0 \\ v_2(x_1) & v_2(x_2) & \cdots & v_2(x_n) & 0 & 0 & v_2(x_{n+2}) & \cdots & 0 \\ v_3(x_1) & v_3(x_2) & \cdots & v_3(x_n) & 0 & 0 & 0 & \ddots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ v_m(x_1) & v_m(x_2) & \cdots & v_m(x_n) & v_m(x_{n+1}) & 0 & 0 & \cdots & v_m(x_d) \end{pmatrix} + \begin{pmatrix} v_0(x_1) \\ v_0(x_2) \\ \vdots \\ v_0(x_n) \\ v_0(x_{n+1}) \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}^T$$

where:

- Each  $v_i(x_j)$  evaluates to the corresponding matrix entry at point  $x_j$
- The diagonal entries evaluate to 2 at their respective points
- The output constraint column evaluates to  $(0, \dots, 0, -3)$  at point  $x_{n+1}$
- $v_0(x)$  interpolates the constant vector  $(\delta_1, \dots, \delta_n, 3, 0, \dots, 0)$

For any boolean circuit  $C$ , we can construct an SSP instance  $(v_0(x), \dots, v_m(x), t(x))$  such that  $C(x_1, \dots, x_\ell) = 1$  if and only if there exist values  $(a_{\ell+1}, \dots, a_m)$  making the above polynomial division possible.

The correctness of this transformation follows from our construction: each gate's constraints are captured at distinct evaluation points, the boolean nature of our variables is enforced by the diagonal matrix, and the polynomial division condition.

Now, for notational convenience, we define  $v'_0(x) = v_0(x) - 1$ .

At each evaluation point  $x_j$ , our circuit constraints require:

$$\left( \sum_{i=1}^m a_i v_i(x_j) + v'_0(x_j) \right)^2 = 1$$

This set of point-wise constraints can be unified into a single polynomial constraint:

$$\left( \sum_{i=1}^m a_i v_i(x) + v'_0(x) \right)^2 - 1 \equiv 0 \pmod{t(x)}$$

where  $t(x) = \prod_{j=1}^d (x - x_j)$  is our target polynomial.

Equivalently, there must exist some polynomial  $h(x)$  such that:

$$\left( \sum_{i=1}^m a_i v_i(x) + v'_0(x) \right)^2 - 1 = h(x)t(x)$$

This is precisely the SSP satisfiability condition. Thus, we have shown that circuit satisfiability is equivalent to the existence of coefficients  $a_i$  that satisfy this polynomial divisibility condition.

#### 8.10.5 From SSPs to SNARKs

We'll use bilinear groups to construct our SNARK. Our construction begins with a bilinear group setup:

$$(\mathbb{F}_p, G_1, G_2, G_T, e, g_1, g_2)$$

The core idea of our construction leverages the pairing operation  $e$  to verify polynomial equations in the exponent, roughly:

$$e(g_1^{[v_0(x) + \sum_{i=1}^m a_i v_i(x)]}, g_2^{[v_0(x) + \sum_{i=1}^m a_i v_i(x)]}) = e(g_1^{h(x)}, g_2^{t(x)}) \cdot e(g_1, g_2)$$

We cannot directly argue about polynomial in the exponents;  $x$  is a formal variable not a concrete value. For this we will sample a uniformly random point  $\tau$  and check the polynomial equation in the exponent on  $\tau$ . We will keep  $\tau$  secret from the prover thus, intuitively, from their point of view random.

The common reference string (CRS) forms the foundation of our construction. It consists of powers of a secret value  $\tau$ :

$$\text{CRS} = (bg, g_1^\tau, g_1^{\tau^2}, \dots, g_1^{\tau^d}, g_2^\tau, g_2^{\tau^2}, \dots, g_2^{\tau^d}, g_1^{\beta v_{\ell+1}(\tau)}, \dots, g_1^{\beta v_m(\tau)}, h_2, h_2^\beta)$$

In our protocol, we distinguish between two types of values:

- Statement:  $(a_1, \dots, a_\ell)$  - the public inputs
- Witness:  $(a_{\ell+1}, \dots, a_m)$  - the private values

For proof generation, the prover computes four crucial group elements:

$$\begin{aligned} \pi_1 &= g_1^{\sum_{i=\ell+1}^m a_i v_i(\tau)} \\ \pi_2 &= g_2^{\sum_{i=\ell+1}^m a_i v_i(\tau)} \\ \pi_3 &= g_1^{h(z)} \\ \pi_4 &= g_1^{\beta \sum_{i=1}^m a_i v_i(\tau)} \end{aligned}$$

The verification process consists of two steps. First, the verifier computes intermediate values:

$$\begin{aligned}\pi'_1 &= g_1^{v_0(z) + \sum_{i=1}^{\ell} a_i v_i(\tau)} \cdot \pi_1 \\ \pi'_2 &= g_2^{v_0(z) + \sum_{i=1}^{\ell} a_i v_i(\tau)} \cdot \pi_2\end{aligned}$$

Then, the verifier performs three critical pairing checks:

$$\begin{aligned}e(\pi'_1, \pi'_2) &= e(\pi_3, g_2^{t(\tau)}) \cdot e(g_1, g_2) \\ e(\pi_1, g_2) &= e(g_1, \pi_2) \\ e(\pi_1, h_2^\beta) &= e(\pi_4, h_2)\end{aligned}$$

These pairing equations serve distinct purposes in ensuring the proof's validity. The first check confirms that the SSP is satisfied by verifying the polynomial equation in the exponent. The second check ensures consistency between the prover's elements in groups  $G_1$  and  $G_2$ . The final check prevents malformed group elements by verifying they were properly constructed using the CRS values.

#### 8.10.6 Soundness

Due to the non-interactive nature of our protocol, we cannot rely on black-box extraction for our security proofs. Instead, we require a non-black-box extractor with access to the prover's code, which is acceptable within our security model.

The soundness guarantees of SNARKs, including our SSP SNARK construction, differ from traditional cryptographic protocols. They necessarily rely on non-falsifiable assumptions or the random oracle model (ROM).

Non-falsifiable assumptions represent a unique class of cryptographic assumptions. The most widely used non-falsifiable assumptions are the knowledge-assumptions, informally the simplest form of these assumptions state that given  $(g, g^\alpha)$  no adversary can output a pair  $(g^x, (g^x)^\alpha)$  without actually “knowing” the exponent  $x$ . The term “non-falsifiable” stems from the fact that these assumptions make claims about what an adversary must know, rather than just what it can or cannot compute.

Our first attempt at formalizing soundness states that for every probabilistic polynomial-time (PPT) adversary  $\mathcal{A}$ , there exists a PPT extractor  $\mathcal{E}_{\mathcal{A}}$  such that:

$$\Pr[(v_1, v_2) \leftarrow \mathcal{A}(g, g^\alpha, z), x \leftarrow \mathcal{E}_{\mathcal{A}}(g, g^\alpha, z) : v_2 = v_1^\alpha \wedge v_1 \neq g^x] = \text{negl}(\lambda)$$

This probability involves three key components:

- $z$  represents auxiliary information available to the adversary

- $\lambda$  denotes the security parameter
- $\text{negl}(\lambda)$  represents a negligible function

Recent research has revealed limitations in this initial formulation. Specifically, certain obfuscated programs, when provided as auxiliary input  $z$ , enable adversaries to produce valid pairs  $(v_1, v_2)$  while preventing the extractor  $\mathcal{E}_{\mathcal{A}}$  from determining  $x$  due to the complexity of reverse-engineering the obfuscated program.

*Knowledge Soundness Definition* To address these limitations, we've developed a more robust formulation for our SNARK construction that assumes only 'benign' auxiliary information. For all relations  $R \in \mathcal{R}$  and non-uniform adversaries  $\mathcal{A}$ , there exists a non-uniform PPT extractor  $\mathcal{E}_{\mathcal{A}}$  such that for all benign  $z \in \mathcal{Z}$ :

$$\Pr[(v_1, v_2); c_0, \dots, c_d] \leftarrow (\mathcal{A} \parallel \mathcal{E}_{\mathcal{A}})(bg, g_1, g_2, \dots, g_1^{\tau^d}, g_2^{\tau^d}, z) : \\ e(v_1, g_2) = e(g_1, v_2) \wedge v_1 \neq g_1^{\sum_{i=0}^d c_i \tau^i}] = \text{negl}(\lambda)$$

This refined definition captures several crucial aspects: the adversary and extractor share access to the exact same inputs and random coins, the extractor must produce coefficients  $c_i$  explaining the adversary's output, the auxiliary information must be "benign" (excluding problematic cases like obfuscated programs), and the probability of adversarial success without the extractor finding a valid witness must be negligible.

While this non-falsifiable assumption is stronger than traditional cryptographic assumptions, it appears to be necessary for constructing efficient SNARKs using current techniques.

*Knowledge Assumption* We now formalize the knowledge assumption d-PKE (power knowledge of exponent).

For every non-uniform PPT adversary  $\mathcal{A}$  there exists a non-uniform PPT extractor  $\mathcal{E}_{\mathcal{A}}$  such that for every "benign"  $z$ :

$$\Pr[\mathcal{A} \parallel \mathcal{E}_{\mathcal{A}}(bg, g_1^{\tau}, g_1^{\tau^2}, \dots, g_1^{\tau^d}, g_2^{\tau}, g_2^{\tau^2}, \dots, g_2^{\tau^d}, z) \\ \rightarrow (v_1, v_2; c_0, \dots, c_d) : e(v_1, g_2) = e(g_1, v_2) \wedge v_1 \neq g_1^{\sum_{i=0}^d c_i \tau^i}] = \text{negl}(\lambda)$$

This is basically saying that if the adversary can output a valid pair  $(v_1, v_2)$  then the extractor can output a polynomial with coefficients  $c_0, \dots, c_d$  such that  $v_1 = g_1^{\sum_{i=0}^d c_i \tau^i}$ . Observe this assumption is very close to what we actually construct in the SNARK.

We now show a sketch of the knowledge soundness proof.

Assuming a knowledge-sound adversary  $\mathcal{A}$ , we are going to construct  $\mathcal{A}'$  as follows:

$$\mathcal{A}'(bg, \{g_1^{\tau^i}\}_{i=0}^d, \{g_2^{\tau^i}\}_{i=0}^d, z' = (z || \{g_1^{\beta v_i(\tau)}\}_{i=\ell+1}^m, h_1, h_2^\beta))$$

We give the rest of the CRS of the SNARK to the adversary inside of the auxiliary input  $z'$ . Then, this adversary runs the adversary of the knowledge soundness assumption as follows:

$$\mathcal{A}(R, z, crs) \rightarrow (\pi_1, \pi_2, \pi_3, \pi_4, (a_1, \dots, a_\ell))$$

such that the three pairing verification equation of the SNARK hold, and outputs  $v_1 = \pi_1 \cdot g_1^{v_0(\tau) + \sum_{i=1}^\ell a_i v_i(\tau)}$ ,  $v_2 = \pi_2 \cdot g_2^{v_0(\tau) + \sum_{i=1}^\ell a_i v_i(\tau)}$  where  $e(v_1, g_2) = e(g_1, v_2)$ .

There is an extractor  $\mathcal{E}'_{\mathcal{A}}$  that can output  $c_0, \dots, c_d$  such that  $v_1 = g_1^{\sum_{i=0}^d c_i \tau^i}$ .

We now construct the extractor  $\mathcal{E}_{\mathcal{A}}$  for knowledge soundness (using the the extractor the knowledge assumption  $\mathcal{E}_{\mathcal{A}'}$ ). The extractor is given  $v_1 = g_1^{f(\tau)}$  where  $f(x) = \sum_{i=0}^d c_i x^i$ .

In order for this extractor to be able to extract the actual witness of the relation, we need two things:

1.  $(f(x))^2 - 1$  is divisible by  $t(x)$ , which can be reduced to the  $d$ -Target Group Strong Diffie-Hellman assumption.
2.  $f_{wit}(x) := f(x) - v_0(x) - \sum_{i=1}^\ell a_i v_i(x)$  is in the span of  $\{v_i\}_{i=\ell+1}^m$  (equivalently there exist  $a_{\ell+1}, \dots, a_m$  such that  $f_{wit}(x) = \sum_{i=\ell+1}^m a_i v_i(x)$ ), which can be reduced to the  $d$ -Power Diffie-Hellman assumption.

The reductions are based on the fact that the pairing verification equations of the SNARK hold.

Since the two conditions are satisfied the extractor can output  $a_{\ell+1}, \dots, a_m$  as a valid witness.

## 8.11 Recursive Proofs of Knowledge

In this lecture, we discuss recursive zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs). We assume familiarity with zkSNARKs.

### 8.11.1 Introduction

Succinct non-interactive arguments of knowledge are short certificates that attest to the correct execution of a computation without revealing any secret inputs. Today, zero-knowledge proofs are being used to secure *billions* of dollars worth of assets <sup>6</sup>. Zero-knowledge proofs enable a new class of secure applications with enhanced integrity and privacy guarantees such as verifiable databases <sup>7</sup>, private

<sup>6</sup>; and

<sup>7</sup>; ; and

voting protocols <sup>8</sup>, anonymous credentials <sup>9</sup>, and private cryptocurrencies <sup>10</sup>.

<sup>8</sup>  
<sup>9</sup>; and  
<sup>10</sup>, and

More technically, SNARKs (for circuit-satisfiability) allow a prover to demonstrate that it knows a secret witness  $w$  such that for prescribed circuit  $F$  and prescribed input and output pair  $(x, y)$  that  $F(x, w) = y$ . However, today, we will be interested in proving *recursive* computation. Without loss of generality, we are interested in proving *tail* recursion, that is, we want to prove (unbounded) recursive applications of a function  $F$ . Unbounded recursion in general allows us to implement more complex programming patterns such as *for* and *while*, which are not bounded ahead of time. This allows us to prove stateful computations with dynamic control flow. In practice, proving recursion allows us to recursively prove blockchain updates, verifiable delay functions, and even a universal machine, where each recursive step is a single cycle of a CPU.

Historically, the best known approach to design a proof system for recursive applications of a function  $F$  was to unroll the entire execution  $F \circ F \circ \dots \circ F$  into a monolithic arithmetic circuit, and then use a standard proof system with succinct proofs for circuit satisfiability. Unfortunately, this would necessarily mean that the prover's space complexity would scale with the *entire trace* of the computation. Moreover, in the setting of preprocessed arguments of knowledge (where the prover and verifier would process the circuit into a succinct key to use for multiple inputs) this fixed the recursion-depth ahead of time, which often does not reflect practice.

The first breakthrough was due to Valiant <sup>11</sup> in 2012, who proposed incrementally verifiable computation (IVC), which reflected the recursive structure of the computation into the proof itself: Given a succinct proof  $\pi_i$  attesting to  $i$  steps of computation, the prover can write a succinct proof  $\pi_{i+1}$  that attests to  $i + 1$  steps by proving the correct execution of an arithmetic circuit that runs the latest step of computation, and checks  $\pi_i$  (using the proof system's verifier). This avoids having to fix the recursion depth ahead of time, while ensuring that the prover's space complexity only scales with a single step of execution.

<sup>11</sup>

Although undoubtedly elegant, Valiant's technique introduces a subtle issue: Proofs of knowledge must satisfy a stronger notion of soundness known as *knowledge-soundness*. Essentially, a proof system is considered knowledge-sound if, for any successful prover with some secret input to the computation, there exists a corresponding *extractor* that, with at most polynomial overhead, can retrieve this secret input given access to the "source code" of the prover. This extractor-based definition becomes problematic in the recursive setting: Recursive proofs require *recursive extraction* in which the extrac-

tor for step  $i$  plays the successful prover for the extractor at step  $i - 1$ . This incurs a polynomial blowup in the extractor for each successive recursive step. In particular, this results in a final extractor that runs in exponential-time with respect to the recursion-depth, which disqualifies it as a valid extractor. To account for this issue, Valiant's original technique (and modern techniques) can only provably guarantee logarithmic-depth recursion in standard models.

### 8.11.2 Preliminaries

We operate in the *preprocessing model*, which means that a trusted party will be responsible for generating a prover and verifier key.



**Definition 8.15** (Incrementally Verifiable Computation). *An incrementally verifiable computation (IVC) scheme is defined by PPT algorithms  $(\mathcal{G}, \mathcal{P}, \mathcal{V})$  and deterministic  $\mathcal{K}$  denoting the generator, the prover, the verifier, and the encoder respectively, with the following interface*

- $\mathcal{G}(1^\lambda, N) \rightarrow \mathbf{p}$ : on input security parameter  $\lambda$  and size bounds  $N$ , samples public parameters  $\mathbf{p}$ .
- $\mathcal{K}(\mathbf{p}, F) \rightarrow (\mathbf{pk}, \mathbf{vk})$ : on input public parameters  $\mathbf{p}$ , and polynomial-time function  $F$ , deterministically produces a prover key  $\mathbf{pk}$  and a verifier key  $\mathbf{vk}$ .
- $\mathcal{P}(\mathbf{pk}, (i, z_0, z_i), \omega_i, \pi_i) \rightarrow \pi_{i+1}$ : on input a prover key  $\mathbf{pk}$ , a counter  $i$ , an initial input  $z_0$ , a claimed output after  $i$  iterations  $z_i$ , a non-deterministic advice  $\omega_i$ , and an IVC proof  $\pi_i$  attesting to  $z_i$ , produces a new proof  $\pi_{i+1}$  attesting to  $z_{i+1} = F(z_i, \omega_i)$ .
- $\mathcal{V}(\mathbf{vk}, (i, z_0, z_i), \pi_i) \rightarrow \{0, 1\}$ : on input a verifier key  $\mathbf{vk}$ , a counter  $i$ , an initial input  $z_0$ , a claimed output after  $i$  iterations  $z_i$ , and an IVC proof  $\pi_i$  attesting to  $z_i$ , outputs 1 if  $\pi_i$  is accepting, and 0 otherwise.

An IVC scheme  $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$  satisfies the following requirements.

1. *Perfect Completeness: For any PPT adversary  $\mathcal{A}$*

$$\Pr \left[ \mathcal{V}(\mathbf{vk}, (i+1, z_0, z_{i+1}), \pi_{i+1}) = 1 \mid \begin{array}{l} \mathbf{pp} \leftarrow \mathcal{G}(1^\lambda, N), \\ (F, (i, z_0, z_i), (\omega_i, \pi_i)) \leftarrow \mathcal{A}(\mathbf{pp}), \\ (\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{K}(\mathbf{p}, F), \\ z_{i+1} \leftarrow F(z_i, \omega_i), \\ \mathcal{V}(\mathbf{vk}, i, z_0, z_i, \pi_i) = 1, \\ \pi_{i+1} \leftarrow \mathcal{P}(\mathbf{pk}, (i, z_0, z_i), (\omega_i, \pi_i)) \end{array} \right] = 1$$

where  $F$  is a polynomial-time computable function represented as an arithmetic circuit.

2. *Knowledge Soundness: Consider constant  $n \in \mathbb{N}$ .*

*For all expected polynomial-time adversaries  $\mathcal{P}^*$  there exists an expected polynomial-time extractor  $\mathcal{E}$  such that*

$$\Pr_{\mathbf{r}} \left[ \begin{array}{l} z_n = z \text{ where} \\ z_{i+1} \leftarrow F(z_i, \omega_i) \\ \forall i \in \{0, \dots, n-1\} \end{array} \mid \begin{array}{l} \mathbf{pp} \leftarrow \mathcal{G}(1^\lambda, N), \\ (F, (z_0, z_i), \Pi) \leftarrow \mathcal{P}^*(\mathbf{pp}, \mathbf{r}), \\ (\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{K}(\mathbf{p}, F), \\ \mathcal{V}(\mathbf{vk}, (n, z_0, z), \Pi) = 1, \\ (\omega_0, \dots, \omega_{n-1}) \leftarrow \mathcal{E}(\mathbf{p}, \mathbf{r}) \end{array} \right] \approx 1$$

where  $\mathbf{r}$  denotes an arbitrarily long random tape.

Moreover,  $F$  is a polynomial-time computable function represented as an arithmetic circuit.

3. *Succinctness: The size of an IVC proof  $\pi$  is independent of the number of iterations  $i$ .*

**Definition 8.16** (Non-Interactive Argument of Knowledge). Consider a relation  $\mathcal{R}$  over public parameters, structure, instance, and witness tuples. A non-interactive argument of knowledge for  $\mathcal{R}$  consists of PPT algorithms  $(\mathcal{G}, \mathcal{P}, \mathcal{V})$  and deterministic  $\mathcal{K}$ , denoting the generator, the prover, the verifier and the encoder respectively with the following interface.

- $\mathcal{G}(1^\lambda, N) \rightarrow \mathbf{p}$ : On input security parameter  $\lambda$ , and length parameter  $N$  samples public parameters  $\mathbf{p}$ .
- $\mathcal{K}(\mathbf{p}, \mathbf{s}) \rightarrow (\mathbf{pk}, \mathbf{vk})$ : On input structure  $\mathbf{s}$ , representing common structure among instances, outputs the prover key  $\mathbf{pk}$  and verifier key  $\mathbf{vk}$ .
- $\mathcal{P}(\mathbf{pk}, u, w) \rightarrow \pi$ : On input instance  $u$  and witness  $w$ , outputs a proof  $\pi$  proving that  $(\mathbf{p}, \mathbf{s}, u, w) \in \mathcal{R}$ .
- $\mathcal{V}(\mathbf{vk}, u, \pi) \rightarrow \{0, 1\}$ : Checks instance  $u$  given proof  $\pi$ .

An argument of knowledge satisfies completeness if for any PPT adversary  $\mathcal{A}$ ,

$$\Pr \left[ \mathcal{V}(\mathbf{vk}, u, \pi) = 1 \mid \begin{array}{l} \mathbf{p} \leftarrow \mathcal{G}(1^\lambda, N), \\ (\mathbf{s}, (u, w)) \leftarrow \mathcal{A}(\mathbf{p}), \\ (\mathbf{p}, \mathbf{s}, u, w) \in \mathcal{R}, \\ (\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{K}(\mathbf{p}, \mathbf{s}), \\ \pi \leftarrow \mathcal{P}(\mathbf{pk}, u, w) \end{array} \right] = 1.$$

An argument of knowledge satisfies knowledge soundness if for all PPT adversaries  $\mathcal{P}^*$  there exists a PPT extractor  $\mathcal{E}$  such that for all randomness  $r$

$$\Pr \left[ (\mathbf{p}, \mathbf{s}, u, w) \in \mathcal{R} \mid \begin{array}{l} \mathbf{p} \leftarrow \mathcal{G}(1^\lambda, N), \\ (\mathbf{s}, u, \pi) \leftarrow \mathcal{P}^*(\mathbf{p}, r), \\ (\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{K}(\mathbf{p}, \mathbf{s}), \\ \mathcal{V}(\mathbf{vk}, u, \pi) = 1, \\ w \leftarrow \mathcal{E}(\mathbf{p}, r) \end{array} \right] \approx 1.$$

**Definition 8.17** (Succinctness). A non-interactive argument system is succinct if the size of the proof  $\pi$  is polylogarithmic in the size of the witness  $w$ .

**Definition 8.18** (Commitment Scheme). *A commitment scheme is defined by polynomial-time algorithm  $\text{Gen} : \mathbb{N}^2 \rightarrow P$  that produces public parameters given the security parameter and size parameter, a deterministic polynomial-time algorithm  $\text{Commit} : P \times M \times R \rightarrow C$  that produces a commitment in  $C$  given a public parameters, message, and randomness tuple such that binding holds. That is, for any PPT adversary  $\mathcal{A}$ , given  $p \leftarrow \text{Gen}(\lambda, n)$ , and given  $((m_1, r_1), (m_2, r_2)) \leftarrow \mathcal{A}(p)$  we have that*

$$\Pr[(m_1, r_1) \neq (m_2, r_2) \wedge \text{Commit}(p, m_1, r_1) = \text{Commit}(p, m_2, r_2)] \approx 0.$$

*The commitment scheme is deterministic if  $\text{Commit}$  does not use its randomness.*

**Definition 8.19** (Circuit Satisfiability). *We define the circuit satisfiability relation  $\text{CSAT}$  over structure, instance, witness tuples as follows.*

$$\text{CSAT} = \left\{ (C, (x, y), w) \mid C(x, w) = y \right\}.$$



## 8.11.3 Construction

**Construction 8.1** (Incrementally Verifiable Computation). *Given a succinct commitment scheme  $(\text{Gen}, \text{Commit})$  and a succinct non-interactive argument of knowledge SNARK for circuit-satisfiability we construct an IVC scheme as follows.*

*Consider an arithmetic circuit  $F$  that takes non-deterministic input. We begin by defining an augmented circuit  $F'$  as follows, where all input arguments are taken as non-deterministic advice.*

$F'(x_i, \omega_i, \pi_i)$ :

1. Parse  $(\text{vk}_{\text{SNARK}}, i, z_0, z_i) \leftarrow x_i$ .
2. If  $i = 0$ :
  - (a) Check that  $z_0 = z_i$ .
3. Otherwise:
  - (a) Check that  $\text{SNARK}.\mathcal{V}(\text{vk}_{\text{SNARK}}, x_i, \pi_i) = 1$ .
4. Output  $x_{i+1} \leftarrow (\text{vk}_{\text{SNARK}}, i + 1, z_0, F(z_i, \omega_i))$ .

*Given the augmented circuit  $F'$ , we define  $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$  as follows.*

$\mathcal{G}(\lambda, N)$ :

1. Output  $p \leftarrow \text{SNARK}.\mathcal{G}(\lambda, N)$ .

$\mathcal{K}(p, F)$ :

1. Compute  $(\text{pk}_{\text{SNARK}}, \text{vk}_{\text{SNARK}}) \leftarrow \text{SNARK}.\mathcal{K}(p, F')$ .
2. Output  $\text{pk} \leftarrow (F', \text{pk}_{\text{SNARK}}, \text{vk}_{\text{SNARK}})$  and  $\text{vk} \leftarrow \text{vk}_{\text{SNARK}}$ .

$\mathcal{P}(\text{pk}, (i, z_0, z_i), (\omega_i, \pi_i))$ :

1. Parse  $(F', \text{pk}_{\text{SNARK}}, \text{vk}_{\text{SNARK}}) \leftarrow \text{pk}$ .
2. Compute  $x_{i+1} \leftarrow F'(\text{vk}_{\text{SNARK}}, (i, z_0, z_i), \omega_i, \pi_i)$ .
3. Let  $x_i \leftarrow (\text{vk}_{\text{SNARK}}, i, z_0, z_i)$ .
4. Output  $\pi_{i+1} \leftarrow \text{SNARK}.\mathcal{P}(\text{pk}_{\text{SNARK}}, (\perp, x_{i+1}), (x_i, \omega_i, \pi_i))$ .

$\mathcal{V}(\text{vk}, (i, z_0, z_i), \pi_i)$ :

1. If  $i = 0$ : Check that  $z_i = z_0$ .
2. Otherwise:
  - (a) Parse  $\text{vk}_{\text{SNARK}} \leftarrow \text{vk}$ .
  - (b) Let  $x_i \leftarrow (\text{vk}_{\text{SNARK}}, i, z_0, z_i)$ .
  - (c) Check that  $\text{SNARK}.\mathcal{V}(\text{vk}_{\text{SNARK}}, (\perp, x_i), \pi_i) = 1$ .

**Lemma 8.2** (Completeness). *Construction 8.1 is complete.*

*Proof.* Consider arbitrary PPT adversary  $\mathcal{A}$ . Suppose  $p \leftarrow \text{Gen}(1^\lambda, N)$ . Suppose that

$$(F, (z_0, z_i, i), \pi_i) \leftarrow \mathcal{A}(p).$$

Suppose that for  $(pk, vk) \leftarrow \mathcal{K}(p, F)$  we have that

$$\mathcal{V}(vk, (z_0, z_i, i), \pi_i) = 1. \quad (8.3)$$

Then, given

$$z_{i+1} \leftarrow F(z_i, \omega_i)$$

and

$$\pi_{i+1} \leftarrow \mathcal{P}(pk, (z_0, z_i, i), (\omega_i, \pi_i))$$

we must show that

$$\mathcal{V}(vk, (z_0, z_{i+1}, N), \pi_{i+1}) = 1 \quad (8.4)$$

with probability 1.

Indeed, consider the base case where  $i = 0$ . Then, by Precondition 8.3, by the verifier's check in the base case (Step 1) we have that  $z_0 = z_i$ . Therefore,  $\mathcal{P}$  can successfully compute  $x_{i+1}$  (Step 3), because the base case check of  $F'$  (Step 2a) passes. Then, by construction of  $F'$  (Step 4) we have that

$$x_{i+1} = (vk_{\text{SNARK}}, i + 1, z_0, F(z_i, \omega_i))$$

Moreover, by the completeness of SNARK, we have that  $\pi_{i+1}$  generated by  $\mathcal{P}$  (Step 4) is indeed satisfying. Therefore, both the checks of  $\mathcal{V}$  in Steps 2b and 2c are passing. As such, we have that postcondition 8.4 holds.

Suppose instead that  $i \geq 1$ . by Precondition 8.3, by the verifier's check in the general case we have that

$$\text{SNARK}.\mathcal{V}(vk_{\text{SNARK}}, (\perp, x_i), \pi_i) = 1$$

for  $x_i = (vk_{\text{SNARK}}, i, z_0, z_i)$ . Then,  $\mathcal{P}$  can successfully compute  $x_{i+1}$  (Step 3), as the SNARK verifier check in  $F'$  (Step 3a) holds. Once again, by construction of  $F'$  (Step 4) we have that

$$x_{i+1} = (vk_{\text{SNARK}}, i + 1, z_0, F(z_i, \omega_i))$$

Moreover, by the completeness of SNARK, we have that  $\pi_{i+1}$  generated by  $\mathcal{P}$  (Step 4) is indeed satisfying. Therefore, both the checks of  $\mathcal{V}$  in Steps 2b and 2c are passing. As such, we have that postcondition 8.4 holds.  $\square$

**Lemma 8.3** (Knowledge-Soundness). *Construction 8.1 is knowledge-sound.*

*Proof.* Let  $n$  be a global constant. Consider a deterministic expected polynomial-time adversary  $\mathcal{P}^*$ . Let  $p \leftarrow \text{Gen}(1^\lambda, N)$ . Suppose on input  $p$  and randomness  $r$ ,  $\mathcal{P}^*$  outputs polynomial-time function  $F$ , instance  $(z_0, z)$ , and IVC proof  $\pi$ . Let  $(pk, vk) \leftarrow \mathcal{K}(p, F)$ . Suppose that

$$\mathcal{V}(vk, (z_0, z, n), \pi) = 1.$$

We must construct an expected polynomial-time extractor  $\mathcal{E}$  that, with input  $(p, r)$ , outputs  $(\omega_0, \dots, \omega_{n-1})$  such that by computing

$$z_{i+1} \leftarrow F(z_i, \omega_i)$$

we have that  $z_n = z$  with probability  $1 - \text{negl}(n)\lambda$ .

We show inductively that we can construct an expected polynomial-time extractor  $\mathcal{E}_i$  that outputs  $((z_i, \dots, z_{n-1}), (\omega_i, \dots, \omega_{n-1}), \pi_i)$  such that for all  $j \in \{i+1, \dots, n\}$ ,

$$z_j = F(z_{j-1}, \omega_{j-1})$$

and

$$\mathcal{V}(vk, z_0, z_i, \pi_i) = 1 \tag{8.5}$$

for  $z_n = z$  with probability  $1 - \text{negl}(n)\lambda$ . Then, because when  $i = 0$ ,  $\mathcal{V}$  checks that  $z_0 = z_i$  the values  $(\omega_0, \dots, \omega_{n-1})$  retrieved by  $\mathcal{E} = \mathcal{E}_0$  are such that computing  $z_{i+1} = F(z_i, \omega_i)$  for all  $i \geq 1$  gives  $z_n = z$ .

At a high level, to construct an extractor  $\mathcal{E}_{i-1}$ , we first assume the existence of  $\mathcal{E}_i$  that satisfies the inductive hypothesis. We then use  $\mathcal{E}_i$  to construct an adversary for the underlying succinct non-interactive argument, which we denote as  $\tilde{\mathcal{P}}_{i-1}$ . This in turn guarantees an extractor for the underlying non-interactive argument, which we denote as  $\tilde{\mathcal{E}}_{i-1}$ . We then use  $\tilde{\mathcal{E}}_{i-1}$  to construct  $\mathcal{E}_{i-1}$  that satisfies the inductive hypothesis.

In the base case, let  $\mathcal{E}_n(p, r)$  output  $(\perp, \perp, \pi_n)$  where  $\pi_n$  is the output of  $\mathcal{P}^*(p, r)$ . By the premise, we have that  $\pi_n$  is satisfying. As such,  $\mathcal{E}_n$  succeeds with probability  $1 - \text{negl}(n)\lambda$  in expected polynomial-time.

For  $i \geq 1$ , suppose we can construct an expected polynomial-time extractor  $\mathcal{E}_i$  that outputs  $((z_i, \dots, z_{n-1}), (\omega_i, \dots, \omega_{n-1}))$ , and  $\pi_i$  that satisfies the inductive hypothesis. To construct an extractor  $\mathcal{E}_{i-1}$ , we first construct an adversary  $\tilde{\mathcal{P}}_{i-1}$  for the underlying SNARK as follows.

$\tilde{\mathcal{P}}_{i-1}(\mathbf{p}, r)$ :

1. Let  $(F, z_0) \leftarrow \mathcal{P}^*(\mathbf{p}, r)$
2. Let  $((z_i, \dots, z_{n-1}), (\omega_i, \dots, \omega_{n-1}), \pi_i) \leftarrow \mathcal{E}_i(\mathbf{p}, r)$ .
3. Let  $\text{vk}_{\text{SNARK}} \leftarrow \text{SNARK}.\mathcal{K}(\mathbf{p}, F)$ .
4. Let  $\mathbf{x}_i \leftarrow (\text{vk}_{\text{SNARK}}, (i, z_0, z_i))$ .
5. Output  $(F', (\perp, \mathbf{x}_i), \pi_i)$ .

We now analyze the success probability of  $\tilde{\mathcal{P}}_{i-1}$ . By the inductive hypothesis, we have that

$$\mathcal{V}(\text{vk}, z_0, z_i, \pi_i) = 1,$$

where  $\pi_i \leftarrow \mathcal{E}_i(\mathbf{p}, r)$  with probability  $1 - \text{negl}(n)\lambda$ . Therefore, by the the verifier's checks we have that

$$\text{SNARK}.\mathcal{V}(\text{vk}_{\text{SNARK}}, (\perp, \mathbf{x}_i), \pi_i) = 1$$

for  $\mathbf{x}_i = (\text{vk}_{\text{SNARK}}, (i, z_0, z_i))$  and  $\text{vk}_{\text{SNARK}} \leftarrow \text{SNARK}.\mathcal{K}(\mathbf{p}, F')$ . Therefore,  $\tilde{\mathcal{P}}_{i-1}$  succeeds in producing a satisfying proof  $\pi_i$  for structure  $F'$  and instance  $\mathbf{x}_i$  with probability  $1 - \text{negl}(n)\lambda$ .

Then, by the knowledge soundness of SNARK there exists an expected-polynomial-time extractor  $\tilde{\mathcal{E}}_{i-1}$  that outputs  $(\mathbf{x}_{i-1}, \omega_{i-1}, \pi_{i-1})$  such that  $\mathbf{x}_i = F'(\mathbf{x}_{i-1}, \omega_{i-1}, \pi_{i-1})$  with probability  $1 - \text{negl}(n)\lambda$ .

Given  $\tilde{\mathcal{E}}_{i-1}$ , we construct an expected polynomial time  $\mathcal{E}_{i-1}$  as follows.

$\mathcal{E}_{i-1}(\mathbf{p}, r)$ :

1. Run  $\tilde{\mathcal{P}}_{i-1}(\mathbf{p}, r)$  to parse
 
$$((z_i, \dots, z_{n-1}), (\omega_i, \dots, \omega_{n-1}), \pi_i)$$
 from its internal state.
2. Compute  $(\mathbf{x}_{i-1}, \omega_{i-1}, \pi_{i-1}) \leftarrow \tilde{\mathcal{E}}_{i-1}(\mathbf{p}, r)$ .
3. Parse  $(\text{vk}_{\text{SNARK}}, (i-1, z_0, z_{i-1})) \leftarrow \mathbf{x}_{i-1}$
4. Output  $((z_{i-1}, \dots, z_{n-1}), (\omega_{i-1}, \dots, \omega_{n-1}), \pi_{i-1})$ .

We now reason about the success probability of  $\mathcal{E}_{i-1}$ . We first reason that the output  $(z_{i-1}, \dots, z_{n-1})$ , and  $(\omega_{i-1}, \dots, \omega_{n-1})$  are valid. By the inductive hypothesis, we already have that for all  $j \in \{i+1, \dots, n\}$ ,

$$z_j = F(z_{j-1}, \omega_{j-1})$$



with probability  $1 - \text{negl}(n)\lambda$ . Moreover, by the success probability of  $\mathcal{E}_{i-1}$ , we have that

$$z_i = F(z_{i-1}, \omega_{i-1})$$

and that

$$\text{SNARK}.\mathcal{V}(\text{vk}_{\text{SNARK}}, x_{i-1}, \pi_{i-1}) = 1$$

where  $x_{i-1} = (\text{vk}_{\text{SNARK}}, (i-1, z_0, z_i))$  with probability  $1 - \text{negl}(n)\lambda$ .

Therefore we have that  $\mathcal{E}_{i-1}$  succeeds with probability  $1 - \text{negl}(n)\lambda$  satisfying the inductive hypothesis.  $\square$



## 9

# Secure Computation

### 9.1 Introduction

Secure multiparty computation (or MPC) considers the problem of enabling mutually distrusting parties to compute a joint public function on their private inputs without revealing any extra information about these inputs beyond what is leaked by the output of the computation. For instance, there could be three parties  $P_1, P_2, P_3$  with private inputs  $x_1, x_2, x_3$  respectively, and they want to compute a public function  $f$  without revealing anything beyond  $f(x_1, x_2, x_3)$ . Note that this notion generalizes zero knowledge proofs, where the prover's input is the statement-witness pair  $(x, w)$ , the verifier's input is the statement  $x$ , and the public function  $f$  is the verification algorithm for the corresponding relation which outputs 1 if  $w$  is a valid witness w.r.t.  $x$ .

This setting is well motivated, and captures many different applications. Now, we look at some applications of MPC which will help build intuition and also highlight the challenge in defining security for MPC.

*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.

*Yao's Millionaires' Problem:* Two millionaires want to know who is richer without revealing their actual wealth. The function they want to compute is  $f(x_1, x_2) = x_1 > x_2$ .

*Private Contact Discovery:* Signal employs MPC to identify contacts from your phone who are also on Signal, all without exposing your contact list to Signal. Here the function  $f$  is an intersection function over the user's contact list and Signal's member list. This

specific application of MPC is commonly known as private set intersection (PSI) and has been the subject of extensive research.

*Password Breaches:* Suppose your browser needs to verify whether your password has been compromised, but without disclosing it to a service that maintains a database of leaked passwords. This is also an instance of PSI.

*Cryptographic Wallets:* Cryptographic wallets typically split their keys into multiple shards that are then given to multiple parties (e.g., the user's device and the wallet provider) so that compromising any one party does not reveal the keys. In principle, whenever these keys are needed, the client can get the shards from each party and perform the operation. However, the client becomes the single point of compromise for that period and that defeats the point of sharding. MPC enables the parties holding the shards to perform any operations on the keys without reconstructing it on a single device.

*Searchable Encryption:* Searchable encryption schemes allow clients to store their data with a server, and subsequently grant servers tokens to conduct specific searches.

Note that in all these applications, the output revealed to the parties itself can reveal a lot of information about the private inputs of the parties. For instance, in the voting example, if only one user votes, their input is trivially revealed. Or, in the case of the millionaires, if  $x_1 = 10^6 + 1$  and  $f(x_1, x_2) = 1$ , then the first millionaire knows that  $x_2 = 10^6$ . Similarly, most searchable encryption 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 of information that is leaked?

From these examples we see that defining security of MPC is tricky because it is imperfect in that some leakage is inherent in the computation. What we want to capture is that no party should learn anything about the private inputs of other parties beyond what is revealed by the output of the computation. To formalize this notion, we adopt the **real/ideal world paradigm**.

## 9.2 Real/Ideal World Paradigm

Suppose there are  $n$  parties and each party  $P_i$  has a private input  $x_i$ . They want to compute a public function (represented by circuit  $C$ ) on their inputs. The goal is to do this securely: even if some parties are corrupted, no party  $P_i$  should learn anything beyond  $y_i$  where  $(y_1, \dots, y_n) = C(x_1, \dots, x_n)$ .

*Real World.* In the real world, the  $n$  parties interact with each other and participate in a protocol  $\Pi$  to compute  $C$ . This protocol can involve multiple rounds of interaction. The real world adversary  $\mathcal{A}$  can corrupt a subset of the parties. The interaction is summarized in Figure 9.1.

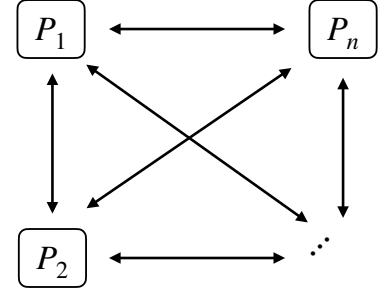


Figure 9.1: Real world interaction.

*Ideal World.* In the ideal world, an angel or trusted third party  $\mathcal{F}_C$  (parameterized by  $C$ ) helps in the computation of  $C$ . The computation in this world is secure by design: each party  $P_i$  sends its input  $x_i$  to  $\mathcal{F}_C$  and receives the output  $y_i$  of the computation  $(y_1, \dots, y_n) = C(x_1, \dots, x_n)$ . In this world, an ideal world adversary  $\mathcal{S}$  controls the parties corrupted by  $\mathcal{A}$  in the real world, and as such, is responsible for sending their inputs to  $\mathcal{F}_C$ . The interaction is summarized in Figure 9.2.

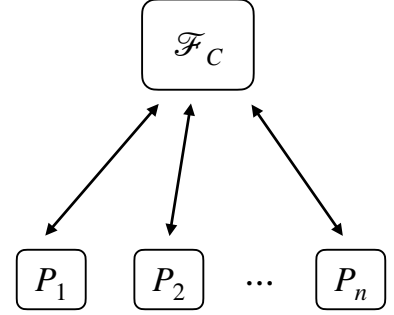


Figure 9.2: Ideal world interaction.

*Notation.* Let  $\lambda$  denote the security parameter. Let  $\text{Real}_{\Pi}^{\mathcal{A}}(1^\lambda, x_1, \dots, x_n, z)$  denote the joint distribution of the honest party outputs as well as the view of the corrupted parties (randomness, inputs, outputs, all messages seen during the protocol) in the real world, where  $z$  is an auxiliary inputs available to  $\mathcal{A}$ . Let  $\text{Ideal}_{\mathcal{F}_C}^{\mathcal{S}}(1^\lambda, x_1, \dots, x_n, z)$  denote the joint distribution of the honest party outputs in the ideal world and the protocol transcript (or view) output by ideal adversary  $\mathcal{S}$  given inputs and outputs of corrupted parties.

*Security Definition.* A protocol  $\Pi$  securely realizes  $C$  if there exists a PPT ideal adversary  $\mathcal{S}^{\mathcal{A}(\cdot)}$  (with oracle access to the real world adversary) such that  $\forall$  non-uniform PPT adversaries  $\mathcal{A}$  in the real world,  $\forall$  private inputs  $x_1, \dots, x_n$ , and  $\forall$  auxiliary inputs  $z \in \{0, 1\}^*$ , we have:

$$\text{Real}_{\Pi}^{\mathcal{A}}(1^\lambda, x_1, \dots, x_n, z) \approx_c \text{Ideal}_{\mathcal{F}_C}^{\mathcal{S}}(1^\lambda, x_1, \dots, x_n, z)$$

*Setting.* So far, we have brushed over some important details of the setting. Below we state these details explicitly:

1. **Assumption:** The protocol could rely on cryptographic assumptions and be secure against computationally bounded adversaries, or it could be statistically secure and protect against unbounded adversaries.
2. **Setup:** The parties in the protocol could have access to a common reference string (CRS) that is generated by the trusted party in the ideal world.
3. **Communication Channel:** The setting could assume private peer-to-peer (P2P) channels among the parties, a broadcast channel,

a combination of both, or insecure channels which don't have privacy or integrity. In the statistical setting, we assume secure P2P channels because they require computational assumptions to set up.

4. **Corruption Type:** We consider primarily two types of adversaries:

- *Semi-honest adversaries:* Corrupted parties follow the protocol execution  $\Pi$  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  $\Pi$ .

5. **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 and the state of that honest party corrupted is given to the adversary.

6. **Fairness:** The protocol we consider could be “fair”, i.e., if one party gets their output, then all parties get their output, or they could be unfair where a corrupted party can abort the computation after learning the output before the honest parties can learn it. It is important to model this weakness of the protocol in the ideal functionality  $\mathcal{F}_C$ . This is done by having the ideal functionality ask the ideal adversary if the honest parties should receive an output. If adversary disapproves, the ideal functionality sends the special abort output  $\perp$  to the honest parties.

7. **Corruption Bound:** The setting places some upper bound  $t$  on the total parties  $n$  that can be corrupted by the adversary (there is nothing to protect when all parties are corrupted). Some well-known corruption settings are  $t < n$ ,  $t < n/2$ ,  $t < n/3$ , and  $t \leq \log n$ .

8. **Standalone vs Concurrent 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 A Simple and Efficient MPC Protocol

This protocol is highly efficient and considers a very simple setting:  $n = 3$  parties, corruption bound  $t = 1$ , unbounded and semi-honest adversary, statistical security, unfairness, and secure P2P channels.

**(2,3)-secret sharing scheme:** A  $(n, t)$ -sharing splits a secret such that any  $t$  shares are enough to reconstruct the secret, and any set of less than  $t$  shares do not reveal anything about the secret. Let  $k \in \mathbb{Z}_p$  be a secret. A  $(3, 3)$ -sharing is easy to construct: sample  $k_1, k_2$  uniformly at random from  $\mathbb{Z}_p$  and set  $k_3 = k - k_1 - k_2 \bmod p$ . Given  $(3, 3)$ -shares, we can construct a  $(2, 3)$ -sharing by considering a replicated secret sharing scheme: each party  $P_i$  gets shares  $k_i, k_{i+1 \bmod 3}$ . It is easy to see that the shares held by any two parties are sufficient to reconstruct the secret.

**Protocol:** The protocol proceeds in the following phases:

*Input Sharing Phase:* Each party  $P_i$  shares their input  $x_i$  using the  $(2, 3)$ -sharing scheme, and sends the corresponding shares to the other parties. For instance,  $P_1$  shares  $x_1$  as  $(x_{1,1}, x_{1,2}, x_{1,3})$ , keeps  $(x_{1,1}, x_{1,2})$ , and sends  $(x_{1,2}, x_{1,3})$  to  $P_2$  and  $(x_{1,3}, x_{1,1})$  to  $P_3$ .

*Computation Phase:* The computation phase maintains the invariant that given  $(2, 3)$ -shares of the input to a gate (addition or multiplication), the computation produces  $(2, 3)$ -shares of the output of the gate. That way, the parties can keep compute the gates all the way to the output of the circuit  $C$ .

**Addition Gate:** Addition gates are simple: given  $(2, 3)$ -shares of  $\alpha$  and  $\beta$ , the parties can compute  $(2, 3)$ -shares of  $\alpha + \beta$  by simply summing up their shares locally. For instance, party  $P_1$  can compute its shares of  $\alpha + \beta$  by computing  $(\alpha_1 + \beta_1, \alpha_2 + \beta_2)$ .

**Multiplication Gate:** Multiplication gates are more involved. The product  $\gamma = \alpha \cdot \beta$  can be represented in the form of shares as follows:  $\gamma = \alpha \cdot \beta = (\alpha_1 + \alpha_2 + \alpha_3) \cdot (\beta_1 + \beta_2 + \beta_3)$ . There are nine terms in this product, and note that all parties together can compute every term given their shares of  $\alpha$  and  $\beta$ . Thus, assuming each party computes the sum of three mutually exclusive terms (let's call it  $\gamma_i$  for party  $P_i$ ), they can locally compute  $(3, 3)$ -shares of  $\gamma$ . To go from  $(3, 3)$ -shares to  $(2, 3)$ -shares, each party  $P_i$  can send  $\gamma_i$  to party  $P_{i-1 \bmod 3}$ .

We are not done yet. These shares represent partial sums and can reveal information about the secret. To prevent this, each party needs to additionally mask  $\gamma_i$  before sending it. In particular,  $P_i$  does the following: (i) mask  $\gamma_i$  by adding a random value  $r_i$  to it before sending it to  $P_{i-1 \bmod 3}$ , (ii) subtract  $r_i$  from  $\gamma_{i+1 \bmod 3}$  re-

ceived from  $P_{i+1 \bmod 3}$ , and (iii) additionally send  $r_i$  to  $P_{i+1 \bmod 3}$ , who also subtracts  $r_i$  from  $\gamma_{i+1 \bmod 3}$ .

*Output Reconstruction Phase:* Each party simply publishes its shares of the output of the circuit  $C$ .

**Theorem 9.1.** *The protocol described above securely realizes the circuit  $C$  in the presence of a static semi-honest adversary that corrupts at most one party.*

*Proof.* Let  $P_i$  be the corrupted party without loss of generality. Consider the following ideal world adversary  $\mathcal{S}$  that simulates the real world adversary  $\mathcal{A}$ :

- *Input Sharing Phase:*  $\mathcal{S}$  knows the input  $x_i$  of corrupted party  $P_i$ . It sends it to  $\mathcal{F}_C$  and receives the output  $y_i = C(x_1, x_2, x_3)$ . To simulate adversary's view perfectly during the input sharing phase,  $\mathcal{S}$  simply sends uniformly random strings on behalf of the honest parties.
- *Computation Phase:* The view of corrupted party only consists of the messages transferred during the evaluation of multiplication gates. Note that the message received by a party is uniformly random due to the random masking, and thus,  $\mathcal{S}$  can again simulate the view perfectly by simply including uniformly random messages.
- *Output Reconstruction Phase:* The view of corrupted party during output reconstruction phase is the shares output by the honest parties. Note that  $\mathcal{S}$  knows the randomness of corrupted party  $P_i$ , its inputs as well as all the messages it saw during the protocol. Thus, it knows the exact shares held by  $P_i$ . Given the output  $y_i$  for the corrupted party received from  $\mathcal{F}_C$  and the shares held by  $P_i$ ,  $\mathcal{S}$  can infer what the shares output by honest parties should be. This completes the simulation.

It is easy to see that this transcript is identical to the real world transcript, and thus the protocol is secure.  $\square$

#### 9.4 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  $\perp$ .



### 9.4.1 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 receiver 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.  $\square$

**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  $x_1, \dots, x_{3n} \in \{0, 1\}$ . Each bit is placed into its own a Rabin oblivious transfer box. The  $i$ th box then reveals either  $x_i$  or else  $\perp$  to the receiver. Let

$$S := \{i \in \{1, \dots, 3n\} : \text{the receiver knows } x_i\}.$$

The receiver picks two sets  $I_0, I_1 \subseteq \{1, \dots, 3n\}$  such that  $\#I_0 = \#I_1 = n$ ,  $I_c \subseteq S$  and  $I_{1-c} \subseteq \{1, \dots, 3n\} \setminus S$ . 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.  $\square$

We will see below that length-preserving one-way trapdoor permutations can be used to realize 1-out-of-2 oblivious transfer.

**Theorem 9.2.** *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  $y_{1-c}$ .  $\square$

#### 9.4.2 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  $S_i \leftarrow \{0, 1\}, i \in \{1, \dots, 5\}$ .

2.  $P_1$  computes

$$\alpha_0 = S_0 \oplus S_2 \oplus m_0$$

$$\alpha_1 = S_0 \oplus S_3 \oplus m_1$$

$$\alpha_2 = S_1 \oplus S_4 \oplus m_2$$

$$\alpha_3 = S_1 \oplus S_5 \oplus m_3$$

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.

## 9.5 Yao's Garbled Circuit

### 9.6 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.

#### 9.6.1 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.

#### 9.6.2 (Garble, 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 (Garble, 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.

$$(\tilde{C}, \{\ell_{i,b}\}_{i \in [n], b \in \{0,1\}}) \leftarrow \text{Garble}(1^k, C)$$

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.

$$C(x) \leftarrow \text{Eval}(\tilde{C}, \{\ell_{i,x_i}\}_{i \in [n]})$$

*Correctness* Correctness is as usual, if you garble honestly, evaluation should produce the correct result.

$$\forall C, x \Pr[C(x) = \text{Eval}(\tilde{C}, \{\ell_{i,x_i}\}), (\tilde{C}, \{\ell_{i,b}\}) = \text{Garble}(1^k, C)] = 1 - \text{neg}(k)$$

*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.

$$\begin{aligned} \exists \text{Sim} : \forall C, x \\ (\tilde{C}, \{\ell_{i,x_i}\}_{i \in [n]}) &\simeq \text{Sim}(1^k, C, C(x)) \text{ where} \\ (\tilde{C}, \{\ell_{i,b}\}_{i \in [n], b \in \{0,1\}}) &\leftarrow \text{Garble}(1^k, C) \end{aligned}$$

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.

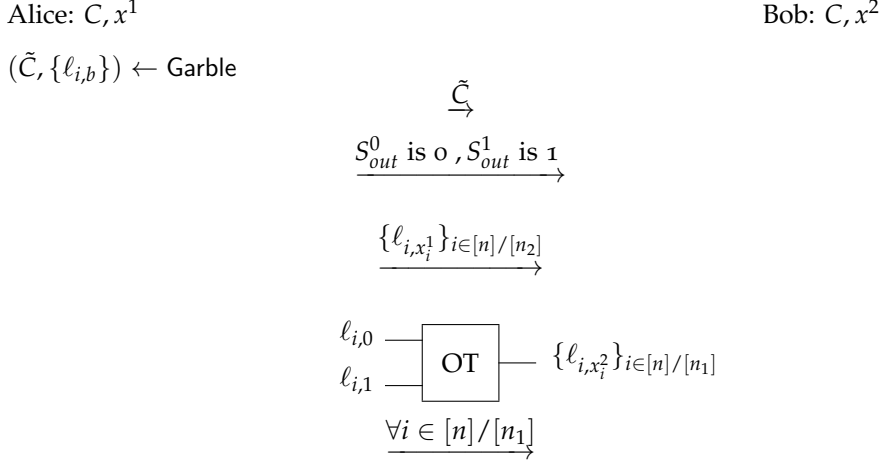
### 9.7 Use for Semi-honest two party secure communication

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 9.3.

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.

Figure 9.3: Messages in Yao's Garbled Circuit



### 9.7.1 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:

$W$  = the set of wires in the circuit

$G$  = 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

$$\forall w \in W \quad S_w^0, S_w^1 \leftarrow \{0, 1\}^k.$$

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,  $S_{w_i}^1$ . 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  $w_3$ ,

$$\begin{aligned} e_g^{00} &= \text{Enc}_{S_{w_1}^0}(\text{Enc}_{S_{w_2}^0}(S_{w_3}^1, 0^k)) \\ e_g^{01} &= \text{Enc}_{S_{w_1}^0}(\text{Enc}_{S_{w_2}^1}(S_{w_3}^1, 0^k)) \\ e_g^{10} &= \text{Enc}_{S_{w_1}^1}(\text{Enc}_{S_{w_2}^0}(S_{w_3}^1, 0^k)) \\ e_g^{11} &= \text{Enc}_{S_{w_1}^1}(\text{Enc}_{S_{w_2}^1}(S_{w_3}^0, 0^k)). \end{aligned}$$

We add  $k$  zeros at the end.

*Final Output* For the final output wire,  $S_{out}$ , we can just give out their values,

$$\begin{aligned} S_{out}^0 &\text{ corresponds to } 0 \\ S_{out}^1 &\text{ corresponds to } 1. \end{aligned}$$

$\tilde{C}$  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  $S_{w_3}$  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.

$$\exists i \in \{a, b, c, d\} : \text{Dec}_{S_{w_2}}(\text{Dec}_{S_{w_1}}(e_g^i)) = S_{w_3}, 0^k$$

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 in the ideal world as well.

### 9.8 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.

### 9.9 Malicious attacker instead 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

$$\begin{aligned} c_1 &= com(x_1; w_1) \\ d_1 &= com(r_1; \phi_1) \end{aligned}$$

Party 2 produces

$$c_2 = com(x_2; w_2)$$

$$d_2 = \text{com}(r_2; \phi_2)$$

We have the following guarantee:  $\exists x_i, r_i, w_i, \phi_i$  such that  $c_i = \text{com}(x_i; w_i) \wedge d_i = \text{com}(r_i; \phi_i) \wedge t = \pi(i, \text{transcript}, x_i, r_i)$ , 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:

$$\begin{array}{ccc} d_1 = \text{com}(s_1; \phi_1) & & d_2 = \text{com}(s_2; \phi_2) \\ \xrightarrow{\quad} & & \xleftarrow{\quad} \\ s_2' & & s_1' \\ \xrightarrow{\quad} & & \xleftarrow{\quad} \end{array}$$

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.

### 9.9.1 Zero-knowledge proof of knowledge (ZK-PoK)

**Definition 9.1 (ZK-PoK).** *Zero-knowledge proof of knowledge (ZK-PoK) is a zero-knowledge proof system  $(P, V)$  with the property proof of knowledge with knowledge error  $\kappa$ :*

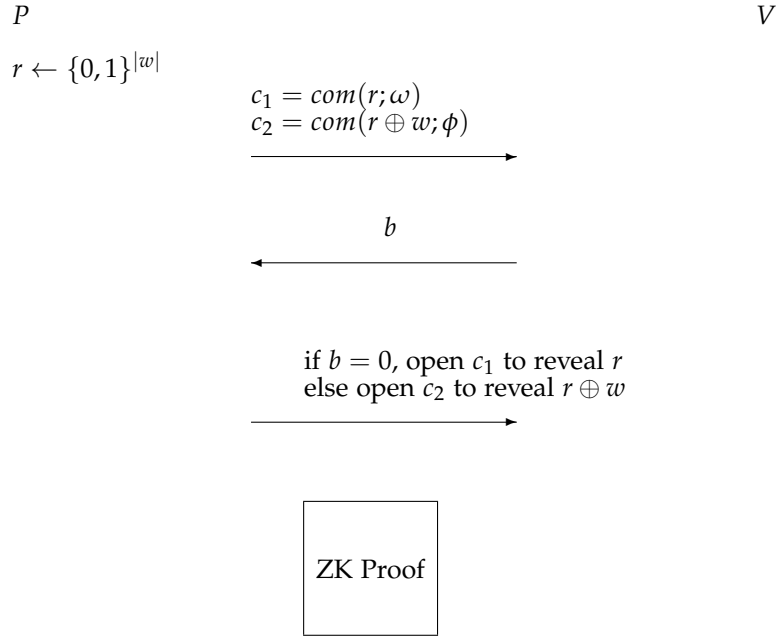
*$\exists$  a PPT  $E$  (knowledge extractor) such that  $\forall x \in L$  and  $\forall P^*$  (possibly unbounded), it holds that if  $\Pr[\text{Out}_V(P^*(x, w) \leftrightarrow V(x))] > \kappa(x)$ , then*

$$\Pr[E^{P^*}(x) \in R(x)] \geq \Pr[\text{Out}_V(P^* \leftrightarrow V(x))] = 1] - \kappa(x).$$

*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:





The last ZK proof proves that  $\exists r, w, \omega, \phi$  such that  $(x, w) \in R$  and  $c_1 = \text{com}(r; \omega)$ ,  $c_2 = \text{com}(r \oplus w; \phi)$ .

### 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.*



# Obfuscation

## 10.1 VBB Obfuscation

**Definition 10.1** (Obfuscator of circuits under VBB). *O is an obfuscator of circuits if*

1. *Correctness*:  $\forall c, x, O(c)(x) = c(x)$ .
2. *Efficiency*:  $\forall c, |O(c)| \leq \text{poly}(|c|)$ .
3. *VBB*:  $\forall A, A$  is PPT bounded,  $\exists S$  (also PPT) s.t.  $\forall c$ ,

$$|\Pr[A(O(c)) = 1] - \Pr[S^c(1^{|c|}) = 1]| \leq \text{negl}(|c|).$$

One can easily construct a PKE scheme based on any SKE scheme  $(\text{Gen}', \text{Enc}', \text{Dec}')$  using the VBB obfuscation:

- $\text{Gen}(1^n)$ :  $\text{sk} \leftarrow \text{Gen}'(1^n)$ , output  $\text{pk} = O(P_{\text{sk}})$ , where  $P_{\text{sk}}(m; r) = \text{Enc}'_{\text{sk}}(m; r)$ .
- $\text{Enc}(\text{pk}, m)$ : Output  $O(P_{\text{sk}})(m; r)$ .
- $\text{Dec}(\text{sk}, c)$ : Output  $\text{Dec}'_{\text{sk}}(c)$ .

One could also construct an IBE scheme with a digital signature scheme  $(G, S, V)$  using the VBB obfuscation:

- $\text{Setup}(1^n)$ :  $(\text{pk}_s, \text{sk}_s) \leftarrow G(1^n)$ , output  $\text{mpk} = \text{pk}_s$ ,  $\text{msk} = \text{sk}_s$ .
- $\text{KeyGen}(\text{msk}, \text{id})$ : Output  $\text{sk}_{\text{id}} = S(\text{msk}, \text{id})$ .
- $\text{Enc}(\text{mpk}, \text{id}, m)$ : Output  $O(P_{\text{mpk}, \text{id}, m})$ , where  $P_{\text{mpk}, \text{id}, m}(\sigma) = \begin{cases} m, & \text{if } V(\text{mpk}, \text{id}, \sigma) = 1, \\ \perp, & \text{o/w} \end{cases}$ .
- $\text{Dec}(\text{sk}_{\text{id}}, c)$ : Output  $c(\text{sk}_{\text{id}})$ .

## 10.2 Indistinguishability Obfuscation

**Definition 10.2** (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  $x$ ,  $C(x) = iO(C(x))$ .*
- **Polynomial slowdown.** *For every circuit  $C \in \mathcal{C}_\kappa$ ,  $|iO(C)| \leq p(|C|)$  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  $iO(C_1) \stackrel{c}{\simeq} iO(C_2)$ . More precisely, there is a negligible function  $\nu(k)$  such that for any (possibly non-uniform) PPT  $A$ ,*

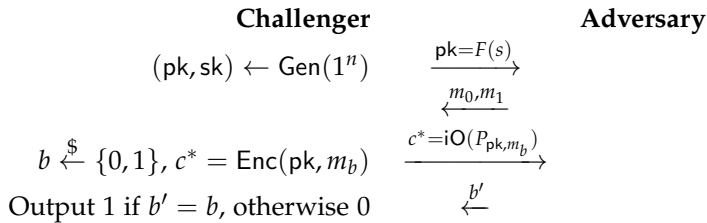
$$|\Pr[A(iO(C_1)) = 1] - \Pr[A(iO(C_2)) = 1]| \leq \nu(k).$$

Note that the circuit size requirement for the indistinguishability property is not really restrictive, as one can always pad circuits to have the same size. Moreover, consider circuit  $c$ , the *best possible obfuscated BPO*  $BPO(c)$ , and  $c'$  which is just padding  $c$  to the same size of  $BPO(c)$ . Computationally bounded adversaries cannot distinguish between  $iO(c')$  and  $iO(BPO(c))$ .

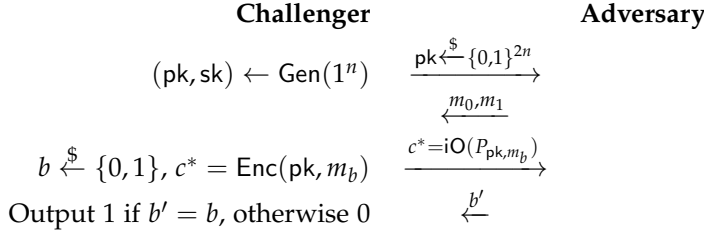
One could construct a PKE scheme based on a length doubling PRG  $F : \{0, 1\}^n \rightarrow \{0, 1\}^{2n}$ , using the  $iO$  obfuscation:

- $\text{Gen}(1^n)$ :  $s \xleftarrow{\$} \{0, 1\}^n$ ,  $\text{sk} = s$ ,  $\text{pk} = F(s)$
- $\text{Enc}(\text{pk}, m)$ : Output  $iO(P_{\text{pk}, m})$ , where  $P_{\text{pk}, m}(s) = \begin{cases} m, & \text{if } F(s) = \text{pk}, \\ \perp, & \text{o/w} \end{cases}$ .
- $\text{Dec}(\text{sk}, c)$ : Output  $c(\text{sk})$ .

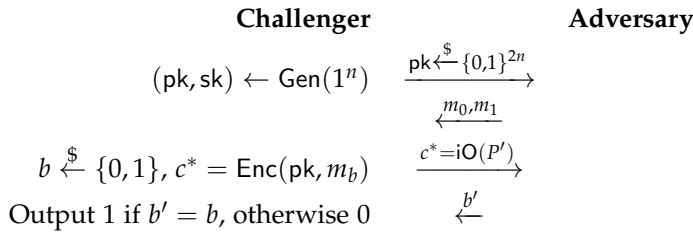
One can prove that this constructed PKE scheme is CPA secure using a hybrid argument. Recall the definition of CPA security:



The first hybrid replaces  $\text{pk}$  with a random string:



This hybrid is clearly indistinguishable from the original due to the property of PRG. The next hybrid replaces the ciphertext with an obfuscation of a different program:



where  $P'$  always outputs  $\perp$ . Clearly, the ciphertext does not depend on the message  $m_b$ . Therefore, as long as we can prove that this hybrid is indistinguishable from the previous one, we are done with the proof. First, as mentioned earlier, we can pad  $P'$  to have the same size as  $P_{pk, m}$ . Then, observe that, since we sample  $pk$  uniformly, with an overwhelming probability, there does not exist any  $s$  such that  $F(s) = pk$ . As a result,  $P_{pk, m}$  also returns  $\perp$  for every input  $s$  and is thus functionally equal to  $P'$ . Therefore, following the indistinguishability property of  $\text{iO}$ , these two hybrids are indistinguishable.

### 10.3 $\text{iO}$ for Polynomial-sized Circuits

**Definition 10.3** (Indistinguishability Obfuscator for  $\text{NC}^1$ ). *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  $\text{iO}_{\text{NC}^1}$  is an indistinguishability obfuscator for circuit class  $\text{NC}^1$  if it is an indistinguishability obfuscator for  $\mathcal{C}_\kappa$ .*

Given an indistinguishability obfuscator  $\text{iO}_{\text{NC}^1}$  for circuit class  $\text{NC}^1$ , we shall demonstrate how to achieve an indistinguishability obfuscator  $\text{iO}$  for all polynomial-sized circuits. The amplification relies on fully homomorphic encryption (FHE).

**Definition 10.4** (Homomorphic Encryption). *A homomorphic encryption scheme is a tuple of PPT algorithms  $(\text{Gen}, \text{Enc}, \text{Dec}, \text{Eval})$  as follows:*

- $(\text{Gen}, \text{Enc}, \text{Dec})$  is a semantically-secure public-key encryption scheme.
- $\text{Eval}(\text{pk}, C, e)$  takes public key  $\text{pk}$ , an arithmetic circuit  $C$ , and ciphertext  $e = \text{Enc}(\text{pk}, x)$  of some circuit input  $x$ , and outputs  $\text{Enc}(\text{pk}, C(x))$ .

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  $\text{sk} = a$  and  $\text{pk} = g^a$ . For ciphertexts  $\text{Enc}(\text{pk}, m_1) = (g^{r_1}, g^{ar_1} \cdot m_1)$  and  $\text{Enc}(\text{pk}, m_2) = (g^{r_2}, g^{ar_2} \cdot m_2)$ , observe that

$$\text{Enc}(\text{pk}, m_1) \cdot \text{Enc}(\text{pk}, m_2) = (g^{r_1+r_2}, g^{a(r_1+r_2)} \cdot m_1 \cdot m_2) = \text{Enc}(\text{pk}, m_1 \cdot m_2)$$

Note that this scheme becomes additively homomorphic by encrypting  $g^m$  instead of  $m$ .

**Definition 10.5** (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  $\text{Eval}(\cdot, \cdot, \cdot)$  is at most polynomial in the security parameter  $\kappa$ .*

### 10.3.1 Construction

Let  $(\text{Gen}, \text{Enc}, \text{Dec}, \text{Eval})$  be a fully homomorphic encryption scheme. We require that  $\text{Dec}$  be realizable by a circuit in  $\text{NC}^1$ . The obfuscation procedure accepts a security parameter  $\kappa$  and a circuit  $C$  whose size is at most polynomial in  $\kappa$ .

1. Generate  $(\text{pk}_1, \text{sk}_1) \leftarrow \text{Gen}(1^\kappa)$  and  $(\text{pk}_2, \text{sk}_2) \leftarrow \text{Gen}(1^\kappa)$ .
2. Encrypt  $C$ , encoded in canonical form, as  $e_1 \leftarrow \text{Enc}(\text{pk}_1, C)$  and  $e_2 \leftarrow \text{Enc}(\text{pk}_2, C)$ .
3. Output an obfuscation  $\sigma = (\text{iO}_{\text{NC}^1}(P), \text{pk}_1, \text{pk}_2, e_1, e_2)$  of program  $P_{\text{pk}_1, \text{pk}_2, \text{sk}_1, e_1, e_2}$  as described below.

The evaluation procedure accepts the obfuscation  $\sigma$  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  $f_1 \leftarrow \text{Eval}(U_x, e_1)$  and  $f_2 \leftarrow \text{Eval}(U_x, e_2)$ , respectively.
2. Denote by  $\omega_1$  and  $\omega_2$  the set of all wires in  $R_1$  and  $R_2$ , respectively. Compute  $\pi_1 : \omega_1 \rightarrow \{0, 1\}$  and  $\pi_2 : \omega_2 \rightarrow \{0, 1\}$ , 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  $P_{pk_1, pk_2, sk_1, e_1, e_2}(x, f_1, \pi_1, f_2, \pi_2)$ .

Program  $P_{pk_1, pk_2, sk_1, e_1, e_2}$  has  $pk_1$ ,  $pk_2$ ,  $sk_1$ ,  $e_1$ , and  $e_2$  embedded.

1. Check whether  $R_1(x) = f_1 \wedge R_2(x) = f_2$ .  $\pi_1$  and  $\pi_2$  enable this check in logarithmic depth.
2. If the check succeeds, output  $\text{Dec}(sk_1, f_1)$ ; 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  $iO_{NC^1}(P_{pk_1, pk_2, sk_1, e_1, e_2})$ .

### 10.3.2 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  $iO(C_1)$ . Recall that  $e_1 = \text{Enc}(pk_1, C_1)$ ,  $e_2 = \text{Enc}(pk_2, C_1)$ , and  $\sigma = (iO_{NC^1}(P_{pk_1, pk_2, sk_1, e_1, e_2}), \dots)$ .

$H_1$  : We instead generate  $e_2 = \text{Enc}(pk_2, C_2)$ , relying on the semantic security of the underlying fully homomorphic encryption scheme.

$H_2$  : We alter program  $P_{pk_1, pk_2, sk_2, e_1, e_2}$  such that it instead embeds  $sk_2$  and outputs  $\text{Dec}(sk_2, f_2)$ . The output of the obfuscation procedure becomes  $\sigma = (iO_{NC^1}(P_{pk_1, pk_2, sk_2, e_1, e_2}), \dots)$ ; we rely on the properties of functional equivalence and indistinguishability of  $iO_{NC^1}$ .

$H_3$  : We generate  $e_1 = \text{Enc}(pk_1, C_1)$  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  $P_{pk_1, pk_2, sk_1, e_1, e_2}$  and arrive at an honest execution of  $iO(C_1)$ .

□

## 10.4 Identity-Based Encryption

Another use of indistinguishability obfuscation is to realize identity-based encryption (IBE).

**Definition 10.6** (Identity-Based Encryption). *An identity-based encryption scheme is a tuple of PPT algorithms (Setup, KeyGen, Enc, Dec) as follows:*

- $\text{Setup}(1^\kappa)$  generates and outputs a master public/private key pair  $(\text{mpk}, \text{msk})$ .
- $\text{KeyGen}(\text{msk}, \text{id})$  derives and outputs a secret key  $\text{sk}_{\text{id}}$  for identity  $\text{id}$ .
- $\text{Enc}(\text{mpk}, \text{id}, m)$  encrypts message  $m$  under identity  $\text{id}$  and outputs the ciphertext.
- $\text{Dec}(\text{sk}_{\text{id}}, c)$  decrypts ciphertext  $c$  and outputs the corresponding message if  $c$  is a valid encryption under identity  $\text{id}$ , or  $\perp$  otherwise.

We combine an indistinguishability obfuscator  $\text{iO}$  with a digital signature scheme  $(\text{Gen}, \text{Sign}, \text{Verify})$ .

- Let  $\text{Setup} \equiv \text{Gen}$  and  $\text{KeyGen} \equiv \text{Sign}$ .
- $\text{Enc}$  outputs  $\text{iO}(P_m)$ , where  $P_m$  is a program that outputs (embedded) message  $m$  if input  $\text{sk}$  is a secret key for the given  $\text{id}$ , or  $\perp$  otherwise.
- $\text{Dec}$  outputs the result of  $c(\text{sk}_{\text{id}})$ .

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  $\text{Com}(\cdot; r)$  the commitment algorithm of a non-interactive commitment scheme with explicit random coin  $r$ .

- Let  $\sigma$  be a common random string.  $\text{Setup}(1^\kappa)$  outputs  $(\text{mpk} = (\sigma, c_1, c_2), \text{msk} = r_1)$ , where  $c_1 = \text{Com}(0; r_1)$  and  $c_2 = \text{Com}(0^{|\text{id}|}; r_2)$ .
- $\text{KeyGen}(\text{msk}, \text{id})$  produces a proof  $\pi = P(\sigma, x_{\text{id}}, s)$  for the following language  $L$ :  $x \in L$  if there exists  $s$  such that

$$\underbrace{c_1 = \text{Com}(0; s)}_{\text{Type I witness}} \vee \underbrace{(c_2 = \text{Com}(\text{id}^*; s) \wedge \text{id}^* \neq \text{id})}_{\text{Type II witness}}$$

- Let  $P_{\text{id}, m}$  be a program which outputs  $m$  if  $V(\sigma, x_{\text{id}}, \pi_{\text{id}}) = 1$  or outputs  $\perp$  otherwise.  
 $\text{Enc}(\text{mpk}, \text{id}, m)$  outputs  $\text{iO}(P_{\text{id}, m})$ .

We briefly sketch the hybrid argument:

$H_0$  : This corresponds to an honest execution as described above.

$H_1$  : We let  $c_2 = \text{Com}(\text{id}^*; r_2)$ , relying on the hiding property of the commitment scheme.



$H_2$  : We switch to the Type II witness using  $\pi_{id_i} \forall i \in [q]$ , corresponding to the queries issued by the adversary during the first phase of the selective-identity security game.

$H_3$  : We let  $c_1 = \text{Com}(1; r_1)$ .

### 10.5 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 (Setup, Sign, Verify).

$(vk, sk) \leftarrow \text{Setup}(1^k)$ :

$sk$  = 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

$\sigma \leftarrow \text{Sign}(sk, m)$ : Output  $F_k(m)$ .

$\text{Verify}(vk, m, \sigma)$ : Output  $P_k(m, \sigma)$ .

Our security requirements will be that the adversary does not win the following game negligibly:

Challenger

$(vk, sk) = \text{Setup}(1^k)$  and

picks random  $m$

Adversary

$$\begin{array}{c} P_{k,m} \\ \xrightarrow{\quad} \\ \alpha, m^* \\ \xleftarrow{\quad} \end{array}$$

Adversary wins game if  $\text{Verify}(vk, m^*, \sigma) = 1$

To prove the security of this system, we use a hybrid argument.  $H_0$  is as above.

$H_1$ : Adjust  $vk$  so that  $vk = iO(P_{k,m,\alpha})$  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  $\alpha$  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  $\alpha$  it relies on some  $\beta$  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  $\beta$ .

## 10.6 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.

$(pk, sk) \leftarrow Gen(1^k)$

$sk$  = key of puncturable function and the seed of the PRF  $F_k$

$pk = iO(P_k)$  where  $P_k$  is the program:

$P_k(m, r)$ :

$t = PRG(r)$

Output  $c = (t, F_k(t) \oplus m)$

$Enc(pk, m)$ : Sample  $r$  and output  $(pk(m, r))$ .

$Dec(sk = k, c = (c_1, c_2))$ : Output  $F_k(sk, c_1) \oplus c_2$ .

Our security requirements will be that the adversary does wins the following game negligibly:

Challenger

$(pk, sk) = Gen(1^k)$  and

Randomly sample  $b$  from  $\{0, 1\}$  and

$c^* = Enc(pk, b)$  and

Adversary

$\xrightarrow{P_{k,c^*}}$   
 $\xleftarrow{b^*}$

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  $pk = iO(P_{k,\alpha,t})$  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  $\alpha$  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.

### 10.7 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.

$H_{final} = H_{2^n}$ :  $iO(pk, c_2)$

$H_1 \cdots H_i$ : 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  $\alpha$  with a random  $\beta$  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  $c_2$ .

