Lecture Notes

on

Introduction to Cryptography

Course 15356/15856, Fall 2020

Vipul Goyal
CMU

Acknowledgment

These lecture notes are largely based on scribe notes of the students who took CMU's "Introduction to Cryptography" by Professor Vipul Goyal in 2018 and 2019. The notes were later edited by the teaching assistant Elisaweta Masserova. We thank all of the students who contributed to these notes:

Danniel Yang
Zhiyang He
Maxwell Johnson
Jaron Chen
Ziyi Zhong
Jacob Neumann
Shaoyan Li
Yi Zhou
Yan Shen
Hanjun Li
Jin Yan
Ke Wang
Eipe Koshy
Walter Tan
Kartik Chitturi
Chuta Sano
Elisaweta Masserova
Macallan Cruff
Huzaifa Abbasi
Jie Zhou
Edouard Dufour Sans
Logan Snow
Justin Cheng
Sahil Hasan
Rebecca Stevens
Hariank Muthakana
Bhuvan Agrawal
Raghav Behl
Nikhil Vanjani
Arushi Bandi
Yifan Song

If your name is not on the list (and you believe it should be) - please let us know.

The notes are not free of typos by any means and not all the chapters have been carefully edited. If you notice anything that should be changed, please get in touch with us.

Vipul Goyal: goyal [at] cs.cmu.edu
Elisaweta Masserova: elisawem [at] cs.cmu.edu

Contents

Introduction

Cryptography studies techniques aimed at securing communication in the presence of adversaries. While encryption is probably the most prominent example of a cryptographic problem, modern cryptography is much more than that. In this class, we will learn about pseudorandom number generators, digital signatures, zero-knowledge proofs, multi-party computation, to name just a few examples.

While the techniques used in current systems are relatively modern, the problems tackled by cryptography have been around for thousands of years. In particular, the problem of encryption was especially appealing to the humankind. Indeed, until recent times "cryptography" was essentially a synonym for "encryption". The goal of encryption is to allow parties to communicate securely in the presence of an adversary. Say Alice wants to send a secret message to Bob. Can she do it so that the eavesdropper Eve who is intercepting communication between Alice and Bob does not get any useful information about the secret?

Throughout history, many encryption schemes have been proposed. They have been mainly used in military and warfare. The ones that span from the ancient Roman Republic to around 50 years ago are considered to be *classical* encryption schemes, whereas the later ones we call *modern* encryption schemes. The classical encryption schemes typically did not provide any sort of provable security guarantees, they have been used just because people thought they work (until they were broken). Modern encryption schemes try to take a different approach by *proving* the security of the schemer relying on some *assumptions*. In this chapter we will look at some classical encryption schemes. We will start by looking at the *symmetric* encryption schemes. In constrast to *asymmetric*, or *public-key* encryption, the encryption- and decryption keys of the symmetric schemes are the same.

1.1 Classical Symmetric Encryption Schemes

Definition 1. A *symmetric key encryption scheme* $SE = (Gen, Enc, Dec)$ is defined by the following three algorithms:

- $k \leftarrow Gen(\kappa)$. The *key generation* algorithm Gen takes as input a security parameter κ [1] and generates a secret key k. The security parameter κ determines the length of the key. Typically, the longer the key, the more secure is the scheme.
- $c \leftarrow Enc(k, m)$. The *encryption* algorithm Enc takes as input a key k and a message m, and outputs a ciphertext c.

[1] Typically we use 1^κ as a security parameter to make sure that Gen runs in time polynomial in the size of the input. For simplicity, we will just write κ.

- $m = Dec(k, c)$. The *decryption* algorithm Dec takes as input a key k and a ciphertext c, and outputs a plaintext message m.

Correctness. The scheme is said to be correct, if for all security parameters κ and all messages m the following holds:

$$Pr[m = Dec(k, c) : k \leftarrow Gen(\kappa), c \leftarrow Enc(k, m)] = 1$$

Remark 1. In the notation above we first specify the statement ($m = Dec(k, c)$), and then, after the colon, we specify the condition ($k \leftarrow Gen(\kappa), c \leftarrow Enc(k, m)$). We will use this notation throughout this manuscript.

We will now take a look at one of the oldest known classical symmetric encryption schemes: the Caesar cipher.

1.1.1 Caesar Cipher

Caesar cipher was used by Julius Caesar in the ancient Roman Republic more than 2000 years ago. It was used to encrypt military information passed around in the army. Here's what Suetonius, a Roman historian, records in *Life of Julius Caesar*:

> If he had anything confidential to say, he wrote it in cipher, that is, by so changing the order of the letters of the alphabet, that not a word could be made out. If anyone wishes to decipher these, and get at their meaning, he must substitute the fourth letter of the alphabet, namely D, for A, and so with the others.

The Caesar cipher is a specific instance of *substitution* ciphers which we will define later. Because of the cipher's historical origin, we will assume that the message consists of letters, although it can be extended to a larger message space.

Formally, the Caesar cipher is defined as follows:

- $k \leftarrow Gen(\cdot)$. The key generation algorithm outputs the key k, a value from 0 to 25.
- $c \leftarrow Enc(k, m)$. The encryption algorithm substitutes each letter of the message m by a letter k positions further in the alphabet.
- $m = Dec(k, c)$. The decryption algorithm substitutes each letter of the ciphertext c by a letter k positions earlier in the alphabet.

Example. For key $k = 1$ and message $m =$ "$attack$", we get:

- $Enc(1, \text{“}attack\text{”}) =$ "$buubdl$"
- $Dec(1, \text{“}buubdl\text{”}) =$ "$attack$"

Remark 2. One way to break the cipher is to test all possible keys (there are only 26 of them) in a brute-force attack. Whichever results in a message that is not gibberish produces probably the right message. Today's computers make conducting this attack very fast, it is hard to say how effective the cipher was in Caeser's time.

1.1.2 Substitution Ciphers

In substitution ciphers the key is some mapping P of the alphabet. To produce a ciphertext, each letter of the message is substituted by another letter as defined by the mapping P. Note that Caesar cipher is indeed a special case of a substitution cipher.

More formally, substitution ciphers are defined by the following three algorithms:

- $P \leftarrow Gen(\cdot)$. The key generation algorithm outputs a uniformly generated random permutation (mapping) P of English letters from a to z.
- $c \leftarrow Enc(P, m)$. The encryption algorithm substitutes each letter λ of the message m with $P(\lambda)$.
- $m \leftarrow Enc(P, c)$. The decryption algorithm substitutes each letter λ of the ciphertext c with $P^{-1}(\lambda)$, where P^{-1} is the inverse permutation of P.

Note that in the English alphabet there are 26! possible permutations. Thus, a way to break this scheme is to try all these 26! permutations (brute force attack). However, there are much better ways to break the scheme:

1. **Frequency Analysis:** Some letters are more frequent than others. For example, if you notice that the most frequent letter in the ciphertext is 'd', there is a high probability that it was mapped from 'e'. Note that for this attack to work, the adversary needs to have access to a ciphertext that is long enough.
2. **Bigrams:** Some pairs of letters are more frequent than others. Say we know (e.g. from the frequence analysis) that 't' maps to 'd'. Then, if in the ciphertext 'e' follows 'd' very frequently, there is a high probability that it was mapped from 'h'.

Remark 3. Sometimes you do not need to be able to decrypt the complete ciphertext. Say you know that the plaintext message is either "attack" or "defend". Then, decrypting even a single letter is enough.

1.1.3 Vigenère Cipher

Vigenère cipher is an extension of Caesar Cipher named after French cryptographer Blaise de Vigenère. Its main idea is to use a key *sequence*, instead of a single key as in the Caeser cipher. More formally:

- $s \leftarrow Gen(\kappa)$. The key generation algorithm outputs a uniformly generated random string s of length κ.
- $c \leftarrow Enc(s, m)$. The encryption algorithm repeats the string s until it has the same length as the message m. Let m_i be the letter at position i of m, and s_i is the letter at position i of s. Then, each letter m_i is substituted by the letter k_i positions further in the alphabet, where k_i corresponds to the position of s_i in the alphabet.
- $m \leftarrow Enc(s, c)$. The decryption algorithm repeats the string s until it has the same length as the message m. Then, each letter c_i is substituted by the letter k_i positions earlier in the alphabet, where k_i corresponds to the position of s_i in the alphabet.

Example.

Plaintext:	`DEFENCE`
Key:	`COVERCO`
Ciphertext:	`FSAIEES`

Looking at what some famous mathematicians and historical figures have to say about Vigenère Cipher reveals that the concept of security has changed significantly over the years. Author and mathematician Lewis Carroll, known for Alice in the Wonderland as

well as his mathematical works, proclaimed that the cipher was "unbreakable". The Scientific American magazine called it "impossible of translation". However, one of the founding fathers of computer science, Charles Babbage, managed to break it, but did not publish his findings[2].

Remark 4. One way to break the scheme is the following: Start by guessing the key length l. Then, arrange the letters of the ciphertext in a matrix as follows:

$$\begin{array}{cccc} 1 & l+1 & 2l+1 & \dots \\ 2 & l+2 & 2l+2 & \dots \\ 3 & l+3 & 2l+3 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{array}$$

and analyse each row using the frequency analysis. Note that it is possible since the letters of a row were encrypted using a single letter key, just as in the Caesar cipher.

1.2 PERFECTLY SECURE ENCRYPTION

The classical ciphers we looked at previously all tried to provide some notion of security, but have all been broken. This suggests that designing a secure encryption scheme is hard. However, even *defining* security is entirely non-trivial. To see why, we take a look at some intuitive attempts at defining it:

- **Attempt 1:** $Pr[A(c) = m : k \leftarrow Gen(\cdot), c \leftarrow Enc(k, m)] = 0$, where A is some adversary. Now, note that even if all the adversary does it to just randomly guess the key, it has a small probability of being correct. Thus, unfortunately, this definition does not work.
- **Attempt 2:** $Pr[A(c) = m : k \leftarrow Gen(\cdot), c \leftarrow Enc(k, m)] = \alpha$, where α is some fixed very small number (say, $\alpha = \frac{1}{2^{100}}$). Recall the "attack"-"defend" example. If the adversary knows that there are only a few possible options for the plaintext message, it will be able to guess it with a probability higher than α.
- The scheme hides every character perfectly, meaning the probability of guessing each character is $\leq \frac{1}{26}$. However, since in an english text some letters are more frequent than others, the adversary will be able to guess some characters with higher probabilities.

The problem is that we are trying to assign some fixed quantity to the probability of guessing the right message, even though for different messages or message spaces the "right" probability can be completely different. Thus, we should isolate the definition of security from any specific message space. Intuitively, having some knowledge about the message space is fine, as long as the ciphertext does not provide more information about the message than the adversary had *before* seeing the ciphertext. Thus, we essentially want the ciphertext to be independent of the message.

[2]Note that if the key length of the Vigenère cipher is equal to the text length, and the key is used only once, the cipher is indeed unbreakable.

Definition 2. (Gen, Enc, Dec) is a perfectly secure encryption scheme if and only if for all pairs of messages (m_1, m_2) and all ciphertexts c the following holds:

$$Pr[c = Enc(k_1, m_1) : k_1 \leftarrow Gen(\cdot)] = Pr[c = Enc(k_2, m_2) : k_2 \leftarrow Gen(\cdot)]$$

Remark 5. Thus, no matter what message you start with, the distribution of the resulting ciphertext is the same as for all other messages in the given message space.

1.2.1 One-Time Pad

We now discuss a scheme that achieves the above definition of perfect security - *One Time Pad*. This scheme is essentially a generalization of the Vigenère Cipher, where the length of the key is the same as the length of the message and the key is used only once. In the following, for simplicity, we assume that we are working with a binary alphabet. Let $\oplus$ denote the "exclusive or" of two binary strings. Formally, One Time Pad is defined as follows:

- $s \leftarrow Gen(\kappa)$. The key generation algorithm outputs a uniformly random binary string of length κ.
- $(s \oplus m) \leftarrow Enc(s, m)$. The encryption algorithm outputs an exclusive or of the key and the message string. This key must only be used once.
- $(s \oplus c) \leftarrow Dec(s, c)$. The decryption algorithm outputs an exclusive or of the key and the ciphertext string.

Theorem 1. One-time pad is a perfect symmetric key encryption scheme.

Proof. For all pairs of messages (m_1, m_2) and for all c, where $|m_1| = |m_2| = |c| =: n$, the following holds:

$$Pr[c = Enc(s, m_1) : s \leftarrow Gen(n)] = Pr[c = Enc(s, m_2) : s \leftarrow Gen(n)] = \frac{1}{2^n},$$

since for each c and m_i, where $i \in \{1, 2\}$, there exists exactly one s such that $c = Enc(s, m_i)$: $s = m_i \oplus c$.

□

Unfortunately, one-time pad has a number of very obvious limitations: what if the message that we want to encrypt is very long? It would require a transmission of a very long key. What if we want to encrypt not a single, but multiple messages? Since the key can only be used once, we would need to transmit multiple keys. These issues prevent us from using one-time pad in practice.

1.3 SHANNON'S THEOREM

In 1948, Claude Shannon, a wartime coding theory researcher for the US government, invented coding theory with his memorandum *A Mathematical Theory of Communication*, and presented his Noisy-Channel Coding Theorem, also known as Shannons's Theorem.

Theorem 2. (Shannon's Theorem) For all perfectly secure symmetric key encryption schemes with message space M and key space K, it holds that $|K| \geq |M|$, where $|K|$ (resp. $|M|$) denotes the size of the key (resp. message) space.

Proof. Assume there exists a perfectly secure encryption scheme (Gen, Enc, Dec) for which $|K| < |M|$. Pick any ciphertext c such that there exists a message m'' that encrypts to c using some key from K. Then, decrypt c using every key from K to recover a set of messages S. Note that S is of size at most $|K|$, because we get at most one unique message per decryption. Since $|K| < |M|$ by the assumption, there exists a message $m' \in M$ such that $m' \notin S$. Thus,

$$Pr[Enc(k, m') = c : k \leftarrow Gen(\cdot)] = 0$$

At the same time, some message m'' must encrypt to c, which contradicts the assumption that the scheme is perfectly secure, since

$$0 = Pr[c = Enc(k_1, m') : k_1 \leftarrow Gen(\cdot)] \neq Pr[c = Enc(k_2, m'') : k_2 \leftarrow Gen(\cdot)] > 0$$

□

Remark 6. Note that for the proof of this theorem we assumed we could decrypt c with every key from K, which can be hard to do in practice for a large enough key space. So, in fact, there could be an encryption scheme that has a smaller key space and is “secure" with respect to some other definition.

One-Way Functions

In the previous chapter, we explored several classical ciphers that attempt to hide information in messages. However, only one of them (One-Time Pad) is perfectly secure. We saw Shannon's Theorem, which claims that it is impossible to have a perfectly secure symmetric key scheme with key space smaller than message space. These results imply that perfect security might be too hard to achieve in many scenarios. However, it may be possible to define a weaker notion of security that might still be sufficient in many use cases. In this chapter, we will discuss security against "reasonable" adversaries.

Intuitively, modern cryptography is based on the assumption that the adversary's computational power is limited (in some "reasonable" way, which we will define later). With respect to encryption, this means that secure schemes should ensure that it is computationally "hard" for the adversary to decrypt a message, while allowing for efficient enc- and decryption for users that have the secret key. To construct such schemes we will require primitives that posses useful computational hardness properties.

In this chapter we will introduce one such fundamental cryptographic primitive - one-way functions, which are "easy" to compute, but "hard" to invert. Before we can dive in, we need to consider a few helpful definitions.

2.1 Polynomial Time and Probabilistic Polynomial Time Algorithms

We now explain what we meant by "reasonable" in the previous paragraph. Typically, modern cryptographic algorithms consider *probabilistic* adversaries that run in *polynomial* time:

Definition 3 (PT)**.** An algorithm A, which runs on input x, is called *polynomial time (PT)* if there exists a polynomial $P(n)$ such that for all large enough $n = |x|$, the number of steps in the computation of $A(x)$ is bounded above by $P(|x|)$.

Definition 4 (PPT)**.** An algorithm A_r, which runs on input x and uses randomness r, is called *probabilistic polynomial time (PPT)* if there exists a polynomial $P(n)$ such that for all large enough $n = |x|$, the number of steps in the computation of $A_r(x)$ is bounded above by $P(|x|)$.

Definition 5 (EPPT)**.** An algorithm A_r, which runs on input x and uses randomness r, is called *expected probabilistic polynomial time (EPPT)* if there exists a polynomial $P(n)$ such that for all large enough $n = |x|$, the **expected** number of steps in the computation of $A_r(x)$ is bounded above by $P(|x|)$. Generally, all theorems and proofs will hold for EPPT algorithms as well.

Example. Consider the following algorithm $\mathcal{A}$:

1 :	At each step, toss a coin.
2 :	If it turns out head, return 1.
3 :	Otherwise, toss again.

Note that this algorithm does not necessarily halt in polynomial time. In fact, it might run for arbitrarily many steps. However, in expectation, the number of steps taken by this algorithm is 2. Therefore, this is a PPT algorithm.

2.2 Negligible and Noticeable Functions

The next definition that we will encounter quite frequently are *negligible* and *noticeable* functions:

Definition 6. A function $\nu(n)$ is called *negligible* if for every polynomial $P(n)$, for all large enough n, $\nu(n) < \frac{1}{P(n)}$.

We typically denote negligible functions as $\mathrm{negl}(n)$. As an example, $\nu(n) = \frac{1}{2^n}$ is a negligible function since the exponential function 2^n grows faster than any polynomial $p(n)$. In general, we want the probability of an adversary breaking our system to be negligible.

Definition 7. A function $f(n)$ is called *noticeable* if there exists two polynomials, $P(n)$ and $Q(n)$, such that for all n large enough, $\frac{1}{P(n)} \leq f(n) \leq Q(n)$.

We typically denote noticeable functions as $\mathrm{notice}(n)$. The following facts about negligible and noticeable functions are easy to derive:

- $f(n) = \mathrm{poly}_1(n) \cdot \mathrm{poly}_2(n)$ is a polynomial.
- $f(n) = \mathrm{notice}_1(n) \cdot \mathrm{notice}_2(n)$ is noticeable.
- $f(n) = \mathrm{notice}(n) \cdot \mathrm{negl}(n)$ is negligible.
- $f(n) = \mathrm{negl}_1(n) \cdot \mathrm{negl}_2(n)$ is negligible.
- $f(n) = \mathrm{notice}_1(n) + \mathrm{notice}_2(n)$ is noticeable.
- $f(n) = \mathrm{notice}(n) + \mathrm{negl}(n)$ is noticeable.
- $f(n) = \mathrm{negl}_1(n) + \mathrm{negl}_2(n)$ is negligible.
- $f(n) = c \cdot \mathrm{notice}(n)$ is noticeable for constant c.
- $f(n) = c \cdot \mathrm{negl}(n)$ is negligible for constant c.

2.3 One-way Functions

2.3.1 Strong One-way Functions

We now present an important definition for this lecture: one-way Functions (OWF). Intuitively, we would like a function $f(x)$ such that computing f is "easy" for all x, but inverting $f(x)$ is "hard" for a randomly chosen x. Here "easy" and "hard" refers to computability in polynomial time. This leads us to the following definition:

Definition 8 (OWF)**.** A function $f : \{0,1\}^n \to \{0,1\}^m$ is an *one-way function* if the following holds:

1. f runs in polynomial time. Equivalently, the computation of $f(x)$ is polynomial time for all x.
2. For all PPT adversaries A, there exists a negligible function $\mathrm{negl}(n)$ such that for all large enough n, we have

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

Remark 7. Here $f(x)$ is called the image of x, and x is called the preimage of $f(x)$. We can easily extend this definition to $f : D \to R$ for any domain D and range R. Moreover, we can either view the domain D as a set from which the input is sampled randomly, or as a distribution.

There is an important subtlety to this definition: in the second condition, we require that the probability of any adversary finding *any* preimage is negligible. It might be tempting to change the second condition to:

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

However, this modified definition is not very useful. To see that, consider the function f such that $f(x) = 0$ for all x. Then f is an OWF under this definition, but it is intuitively useless for cryptographic purposes. Therefore, we strengthen condition 2 to the above definition.

2.3.2 Variants

Technically, the definition provided above refers to *strong* one-way functions. Additionally, there exist *weak* one-way functions, that can be quite useful as well:

Definition 9 (Weak OWF)**.** A function $f : \{0,1\}^n \to \{0,1\}^m$ is a *weak one-way function* if the following holds:

1. f runs in polynomial time. Equivalently, the computation of $f(x)$ is polynomial time for all x.
2. For all PPT adversaries A, there exists a noticeable function $\mathrm{notice}(n)$ such that for all large enough n, we have

$$\Pr[f(x) = f(x') : x \xleftarrow{\$} \{0,1\}^n, x' \xleftarrow{\$} A(f(x))] \leq 1 - \mathrm{notice}(n).$$

Intuitively, a weak one-way function should be hard to invert on some noticeable fraction of inputs.

Definition 10. A function $f : D \to R$ is an injective one-way function if f is a one-way function, and f is injective. Injective one-way functions are also called one-to-one one-way functions.

Remark 8. An injective function (also known as one-to-one function) is a function that maps distinct elements of its domain to distinct elements of its range.

Definition 11. A function $f : D \to R$ is a *one-way permutation (OWP)* if f is an injective one-way function, and f is a permutation (i.e. $D = R$).

2.3.3 Limitations

Unfortunately, one-way functions only satisfy a weak sense of security. They only guarantee that the input to function f is not leaked entirely, but it is still possible that a substantial amount of information is leaked. In fact, the following statement is true:

Proposition 1. Given an one-way function $f : \{0,1\}^n \to \{0,1\}^n$, we can build another one-way function $g : \{0,1\}^{2n} \to \{0,1\}^{2n}$, such that g leaks half of its input.

Proof. Let $\|$ denote concatenation. Consider the following definition of g: given input $x_1\|x_2$, where $|x_1| = |x_2| = n$, output $f(x_1)\|x_2$. It suffices for us to argue that g is an one-way function.

Assume for the sake of contradiction that there is an adversary A that inverts g with a noticeable probability. We construct an adversary B that inverts f with noticeable probability. Consider the following algorithm:

1 : Given $f(x)$, construct $y = f(x)\|t$, where t is a random number from $\{0,1\}^n$.
2 : Run $A(y)$.
3 : If A halts without failing, return the first n bits of $A(y)$'s output.
4 : Otherwise, return FAIL.

Now it's easy to see that B succeeds as long as A succeeds. Since A has noticeable probability of success, B has noticeable probability of success, which implies f is not an one-way function, leading to a contradiction. Therefore g is an one-way function. □

We also have the following statement about the existence of one-way functions:

Theorem 3. If $\mathbb{P} = \mathbb{NP}$, then one-way functions do not exist.

Proof. Here we present a sketch of the proof. Assume for the sake of contradiction that there exists a one-way function $f : D \to R$. Then, since f can be computed in polynomial time, inverting f is in $\mathbb{NP}$ because given a preimage, it takes polynomial time to check its validity. Assuming $\mathbb{P} = \mathbb{NP}$, finding a preimage for f could be computed in polynomial time, which means f is not a one-way function, leading to a contradiction. □

2.4 Constructing One-way Functions

We will now try to provide concrete constructions of one-way functions. In particular, we will assume the hardness of certain problems, i.e. they cannot be solved in PPT, and construct one-way functions based on such assumptions. The first such example would be the hardness of factorization.

2.4.1 Hardness of Factorization and Chebyshev's Theorem

Assumption 1 (Factoring Assumption)**.** Define $P_n = \{p : p \text{ is prime and } p \leq 2^n\}$ as the set of all n-bit prime numbers. Then for all PPT algorithms A:

$$\Pr[A(N) = (p,q) : p \xleftarrow{\$} P_n, q \xleftarrow{\$} P_n, N = p \cdot q] \leq \text{negl}(n).$$

Theorem 4 (Chebyshev's Theorem)**.** Let p be a n-bit number chosen uniformly at random. Then $\Pr[p \text{ is prime}] \geq \frac{1}{2n}$.

2.4.2 OWF Based on Factorization

We now propose two one-way function constructions based on the factorization assumption. The first construction we consider is defined for a somewhat contrived input distribution, the second one for uniform inputs.

Construction 1. Consider the following distribution as the input domain D: Sample p, q from P_n uniformly at random, return $x = p\|q$. Define the function $f : D \to R$ that takes in $2n$-bit input $p\|q$ and output $p \cdot q$.

Proposition 2. f is a one-way function for D under Assumption 1.

Proof. Assume for the sake of contradiction f is not an one-way function. Then there must exist PPT adversary A that has a noticeable probability of inverting f. Using A, we'll build a PPT algorithm B that breaks the factoring assumption with noticeable probability. B takes as input a challenge N and will attempt to factor it. Consider the following definition for B:

1 :	Run $A(N)$, let its output be $x = p\|q$.
2 :	If $p \cdot q = N$, return (p, q).
3 :	Otherwise, return FAIL.

Note that here we assume N is a product of 2 primes, as in Assumption 1. It is easy to see that if A succeeds, then B succeeds as well. Therefore, we have

$$\text{notice}(n) \leq \Pr[f(x) = N : x \xleftarrow{\$} A(N)] = \Pr[B(N) = (p, q)].$$

This is a contradiction. Therefore, f is a one-way function. □

Construction 2. Let $D = \{0,1\}^{n^3}$ and $R = \{0,1\}^{2n}$. Given $x \in D$, f interprets x as a set S of n^2 n-bit integers, and uses deterministic PT primality test to find the first two 2 primes in S. If f finds p and q, then output $p \cdot q$. Otherwise, it outputs FAIL.

Proposition 3. f is an one-way function for the uniform distribution under assumption 1.

Proof. Assume for the sake of contradiction that there is a PPT adversary A that inverts f with noticeable probability. Consider the following PPT algorithm B that gets a challenge integer N as input and attempts to break the factoring assumption:

1: Run $A(N)$.

2: If it succeeds and returns $x' \in D$, then interpret x' as a set of n^2 integers and use primality test to find the first two primes in this set. If found, output the primes.

3: Else, output FAIL.

Note that we can't analyze the success probability of B using the same methods in Proposition 2, because the function f now have a probability of outputing FAIL, which means A could receive FAIL as an input. Note that if f would output FAIL with noticeable probability, then even a basic adversary that returns $(0, \cdots, 0)$ all the time could have noticeable probability of inverting f. Therefore, we prove the following claim:

Claim 1. The probability that f outputs FAIL is negligible.

Proof. Here we have

$$\begin{aligned}
\Pr[f \text{ outputs FAIL}] &= \Pr[\text{There is at most 1 prime in } S] \\
&= \Pr[\text{There is no prime in } S] + \Pr[\text{There is exactly 1 prime in } S] \\
&\leq (1 - \frac{1}{2n})^{n^2} + n^2 \cdot (1 - \frac{1}{2n})^{n^2-1} \cdot 1 \\
&\leq ((1 - \frac{1}{2n})^n)^n + 2n^2((1 - \frac{1}{2n})^n)^n \\
&\leq (e^{-1/2})^n + 2n^2(e^{-1/2})^n \\
&= e^{-n/2} + 2n^2 e^{-n/2}.
\end{aligned}$$

We used Chebyshev's theorem and the fact that $\Pr[p \text{ is prime}] \leq 1$ in the first inequality, and the fact that for $x = -\frac{1}{2}$ the sequence $(1 + \frac{x}{n})^n$ is an increasing sequence converging to e^x in the third inequality. Since the product of a polynomial and a negligible function is negligible, and the sum of two negligible funcitons is negligible, we get that the probability of f outputting FAIL is negligible.

□

Now let $\epsilon = \Pr[f(x) = \text{FAIL}]$. Assuming $x \xleftarrow{\$} D$ and $x' \xleftarrow{\$} A(f(x))$, we have:

$$\begin{aligned}
\Pr[f(x) = f(x')] &= \Pr[f(x) = f(x') | f(x) = \text{FAIL}] \cdot \epsilon + \Pr[f(x) = f(x') | f(x) \neq \text{FAIL}] \cdot (1 - \epsilon) \\
&\geq \text{notice}(n).
\end{aligned}$$

Since the probability that f outputs FAIL is negligible, and we assume that A inverts f with noticeable probability, A must succeed on non-FAIL inputs with noticeable probability. Thus, $\Pr[f(x) = f(x') | f(x) \neq \text{FAIL}]$ is noticeable. The probability that B succeeds is at least the probability of A succeeding on non-FAIL inputs multiplied by the probability of f not outputting FAIL. The probability of B's success is therefore noticeable (as a product of two noticeable functions), which is a contradiction. We conclude that f is a one-way function. □

2.4.3 Weak OWF Based on Factorization

Besides standard one-way functions, we can also construct weak one-way functions for uniform inputs based on Assumption 1 in a simpler way. Consider the following example:

Construction 3. Define the function $f : D \to R$ that takes in a uniform $2n$-bit input, interprets it as a concatenation of two n-bit integers $p \| q$, and outputs $p \cdot q$.

Proposition 4. f is a weak one-way function under Assumption 1.

Proof. From Chebyshev's Theorem, we know the probability that both p and q are primes is at least $(\frac{1}{2n})^2 = \frac{1}{4n^2}$. Now from Construction 1 and Proposition 2, if p, q are both prime, for any PPT algorithm A, $\Pr[f(x) = f(x') : x' \leftarrow A(f(x))] \leq \text{negl}(n)$. Therefore, we have

for any PPT algorithm B,

$$\begin{aligned}
&\Pr[f(x) = f(x') : x \xleftarrow{\$} D, x' \leftarrow B(f(x))] \\
&= \Pr[f(x) = f(x') : x \xleftarrow{\$} D, x' \leftarrow B(f(x)) | x = p \| q, p, q \in P_n] \Pr[p, q \in P_n] \\
&+ \Pr[f(x) = f(x') : x \xleftarrow{\$} D, x' \leftarrow B(f(x)) | x = p \| q, p \notin P_n \vee q \notin P_n] \Pr[p \notin P_n \vee q \notin P_n] \\
&\leq \Pr[f(x) = f(x') : x \xleftarrow{\$} D, x' \leftarrow B(f(x)) | x = p \| q, p, q \in P_n] + \Pr[p \notin P_n \vee q \notin P_n] \\
&\leq \mathrm{negl}(n) + 1 - \frac{1}{4n^2} \leq 1 - (\frac{1}{4n^2} - \mathrm{negl}(n)).
\end{aligned}$$

Since $\frac{1}{4n^2}$ is noticeable, we have that f is a weak one-way function. □

2.5 HARDCORE PREDICATES

As we learned in the previous sections, one-way functions guarantee only that the input x to the one-way function is not leaked *entirely*. Thus, it is entirely possible that substantial amounts of information are leaked. However, since the input is not leaked entirely, it seems likely that there must be at least one bit of the input x that is hard to guess. Thus, the question is whether we can pinpoint such bits of x. More formally: Given $f(x)$ where $f : D \to R$ is an one-way function, is there a predicate h such that $h(x)$ is hard to predict? This leads to the following definition:

Definition 12 (Hardcore Predicate)**.** A predicate $h : \{0,1\}^n \to \{0,1\}$ is a hardcore predicate (HCP) for an one-way function $f : \{0,1\}^n \to \{0,1\}^m$ if

1. h runs in polynomial time.
2. For all PPT adversaries A and for all n large enough,

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

It is a natural question to ask if there exists a single hardcore predicate that works for all one-way functions f. However, if there exists such a predicate h, then we can construct a one-way function f' from a generic one-way function f such that $f'(x) = f(x) \| h(x)$. It is easy to check that f' is a one-way function, and h is not a hardcore predicate for f'. Therefore, we have to be less ambitious when we're constructing hardcore predicates. Here the problem was that f could depend upon h. In the following, we will choose h at random after f is chosen to make sure it is independent of f. This leads us to the next theorem:

Definition 13 (Inner Product)**.** Given $x, r \in \{0,1\}^n$, the inner product of x and r is $\langle r, x \rangle = \sum_i r_i x_i \mod 2$.

Theorem 5 (Goldreich-Levin Theorem)**.** Given a one-way function $f : \{0,1\}^n \to \{0,1\}^m$, sample r at random from $\{0,1\}^n$. Then $h_r(x) = \langle r, x \rangle$ is a hardcore predicate for f with probability $1 - \mathrm{negl}(n)$.

Proof Idea: Here we only present the simplest case of the proof. Assume there exists an adversary A that predicts $h_r(x)$ with probability 1 for all $r \in \{0,1\}^n$. Let $e_i \in \{0,1\}^n$ be the vector with 1 in the i-th entry and 0 in all other entries for all $i \in \{1, 2, \cdots, n\}$. Then we can use A to predict $h_{e_i}(x)$ for all e_i. Since $h_{e_i}(x) = \langle e_i, x\rangle = \sum_{j=1}^{n}(e_i)_j x_j \mod 2 = x_i$, computing all $h_{e_i}(x)$ gives us x bit by bit. This contradicts the assumption that f is a one-way function. Proof of the other cases, where A predicts $h_r(x)$ with probability less than 1 but better than $\frac{1}{2}$ are generalization of this idea and are more complex.

2.5.1 Weak One-to-One One-Way Functions

Here we show an example of a weak one-to-one one-way function. We will use what we call a *signal string*, which is a string used to tag a value with some extra information.

We often concatenate signal strings with output values to provide more information about the values. Signal strings are useful for creating one-to-one functions.

Construction 4. We construct a function f which takes as input a string x in $\{0,1\}^{2n}$ and proceeds as follows:

Let $x = x_1||x_2,$ for $x_1, x_2 \in \{0,1\}^n$.
Case 1: If x_1 and x_2 are primes, output $x_1 \cdot x_2$||signal string
Case 2: Else, output $x_1||x_2$||signal string

Our signal string specifies whether $x_1 > x_2$ and which case of f created the output. It's not important to specify how we encode our signal string, but the encoding schema could range from a compact binary representation (e.g. "10" to specify $x_1 > x_2$ and case 1) to a verbose encoding (e.g. "$x_1 > x_2$ and we took case 1", encoded to binary using ASCII). Note that the signal string does not specify the values of x_1 or x_2.

Proof. We now show that f is a weak one-way function. By Chebyshev's Theorem, we know that there is a noticeable probability of x_1 and x_2 both being prime (and the probability of f returning $x_1 \cdot x_2||$ signal string is noticeable as well). Say some PPT adversary A is able to invert conditioned on x_1, x_2 both being prime with some non-negligible probability. In this case, we can construct a PPT adversary B breaking the factoring assumption as follows: B takes as input an integer N, chooses a bit b at random and runs A on input "$N||x_1 > x_2$ and case 1" if $b = 1$, and on input "$N||x_1 \leq x_2$ and case 1' if $b = 0$. Then, B parses A's output as $x_1||x_2$ and outputs x_1, x_2. If A succeeds with probability δ, B succeeds with probability δ as well. Thus, if A has a non-negligible success probability, then B breaks factoring assumption with a non-negligible probability as well, which is a contradiction. Thus, we have shown that there is a noticeable fraction of inputs that are hard to invert and so f is a weak one-way function.

f is one-to-one because the input is unambiguously defined (though possibly hard to compute) given the output. Fundamental theorem of arithmetic states that every number has unique prime factors. Hence in case 1, $N = x_1 \cdot x_2$ forces the input to be either $x_1||x_2$ or $x_2||x_1$. However now the signal string tells us which one of these inputs is the correct one. In case 2, the output contains the input in clear. The signal string also tells us which case we are in. □

Remark 9. Note that f is not a strong one-way function. Regardless of the value of n,

$$Pr[x_1 \text{ or } x_2 \text{ is even}] = \frac{3}{4}$$

This is a high probability and, in this case, we are in case 2 and thus the output contains the input $x_1||x_2$.

2.6 YAO'S HARDNESS AMPLIFICATION* (OPTIONAL MATERIAL)

In this section, we show how to create a strong one-way function given a weak one-way function.

Say we have a weak one-way function f with associated hardness $1 - q$. Define a function F as follows:

$$F(x_1, x_2, ..., x_N) = f(x_1)||f(x_2)||...||f(x_N), \text{ for } N = \frac{n}{q}$$

Theorem 6 (Yao's Hardness Amplification)**.** F is a strong one-way function.

Intuition:

F takes as input a long string interpreted as many strings of length n. F runs f on each of these smaller strings and concatenates the results together. For an adversary to invert F, they must invert each of $f(x_1), f(x_2), ..., f(x_N)$. If even one of those is hard to invert, F will be hard to invert.

Proof. Assume for the sake of contradiction that there exists a PPT adversary A which can invert F with noticeable probability p.

We will construct a PPT adversary B which inverts f with probability greater than q, which will contradict the assumption that f is a weak one-way function with associated hardness q.

Idea: Define B as follows:

1 :	Choose $i \xleftarrow{\$} [1, N]$.
2 :	For $j \neq i$, generate x_j at random.
3 :	Invoke A($f(x_1)||...||f(x_{i-1})||y||f(x_{i+1})||...||f(x_N)$).
4 :	On A's response, interpret its result as $x'_1, ..., x'_i, ..., x'_N$, and return x'_i.

The probability of B inverting f is noticeable, because A has a noticeable probability of inverting F. However, a noticeable probability of inverting f is not sufficient; we must construct an adversary that inverts B with probability greater than q.

Attempt: "Just try again"

Repeat k times: Put y in a random position, choose the other x_j's at random, and invoke A, as described in the definition of B above.

A succeeds with probability p, so the probability of failing to invert $f(x)$ should be $(1 - p)^k$, right?

No! Since y is in every input, each invocation of A is not independent. It is possible that y is simply hard to invert, so repeated attempts will not increase the chances of inverting y under f.

Final attempt:

As in our previous attempt, B will rerandomize and reinvoke A. This time, it will repeat this process $\frac{n^2}{pq}$ times. We will show that this is sufficient for B to invert f with probability greater than q.

First, we must introduce a definition:

Definition 14. BAD = $\{x | Pr_{\text{coins of B}}[\text{B inverts } f(x) \text{ in one iteration}] \leq \frac{pq}{2n}\}$

Claim 2. If $x \notin BAD$, B inverts $y = f(x)$ with probability $\geq 1 - negl(n)$.

Proof.

$$\begin{aligned}
\Pr[\text{B not inverting } f(x),\, x \notin BAD \text{ in one iteration}] &\leq \left(1 - \frac{pq}{2n}\right) \\
\Pr[\text{B not inverting } f(x),\, x \notin BAD \text{ in } \frac{n^2}{pq} \text{ iterations}] &\leq \left(1 - \frac{pq}{2n}\right)^{\frac{n^2}{pq}} \\
&\leq \left(1 - \frac{pq}{2n}\right)^{\left(\frac{1}{\frac{pq}{2n}}\right)^{\frac{n}{2}}} \\
&\leq \left(\frac{1}{e}\right)^{\frac{n}{2}} \\
&\leq negl(n)
\end{aligned}$$

□

Claim 3. The fraction of BAD inputs is less than $\frac{q}{2}$, i.e. $Pr_{\text{over x}}[x \in BAD] < \frac{q}{2}$.

Proof. Assume for the sake of contradiction that $Pr_{\text{over x}}[x \in BAD] \geq \frac{q}{2}$.

$$\begin{aligned}
Pr[\text{A succeeds in inverting } F(x_1, x_2, ..., x_N)] = p = & Pr[\text{A succeeds } | \forall i : x_i \notin BAD] \cdot Pr[\forall i : x_i \notin BAD] + \\
& Pr[\text{A succeeds } | \exists i : x_i \in BAD] \cdot Pr[\exists i : x_i \in BAD] \\
\leq & 1 \cdot \left(1 - \frac{q}{2}\right)^N + \big(N \cdot Pr[\text{A succeeds} | x_i \in BAD] \\
& \cdot Pr[\exists i : x_i \in BAD]\big) \\
\leq & \left(1 - \frac{q}{2}\right)^N + \frac{n}{q} \cdot \frac{pq}{2n} \cdot 1 \\
p \leq & negl(n) + \frac{p}{2} \\
p \leq & 2 \cdot negl(n) \leq negl(n)
\end{aligned}$$

This is a contradiction to the definition of A, because we know $p \geq notice(n)$. □

So, we get that

$$Pr[B \text{ fails }] = Pr[B \text{ fails} | x \in BAD] Pr[x \in BAD] + Pr[B \text{ fails} | x \notin BAD] Pr[x \notin BAD] < 1 \cdot \frac{q}{2} + negl(n) \cdot 1$$

and thus Pr[B succeeds] $> 1 - \frac{q}{2} - negl(n) > 1 - q$. This contradicts the assumption that f is a weak one-way function which can only be inverted with probability $\leq 1 - q$. Therefore, F is a strong one-way function. □

Pseudorandomness

Randomness plays an important role for various schemes in cryptography, as any scheme requires a uniformly sampled key. Schemes such as encrypting a session key in an SSL connection or encrypting a hard drive require randomness. Randomness can come from various input sources, such as mouse movements, key presses, or the temperature of the computer. However, these sources can only generate a limited amount of randomness, and this amount is often insufficient for most schemes in cryptography. As intuition suggests, the longer a key is, the harder it is for an adversary to guess the key, so if there is insufficient randomness, the key is no longer secure.

Because good randomness can be hard to come by, we need to find a workaround for generating keys purely through randomness. This forces us to ask the question: "Is it possible to stretch a small random string into a longer random string?" Let us consider a small random string $s \xleftarrow{\$} \{0,1\}^n$. Is there an algorithm $A(s)$ that deterministically produces another truly random string t such that $t \in \{0,1\}^m$ and $m > n$?

Intuitively, even if we consider the simple case where $m = n + 1$, we know that this is not possible, as we will only be able to generate 2^n unique outputs as opposed to the 2^{n+1} unique outputs that a truly random string would have. Pseudorandomness can help us overcome the bottleneck created by the lack of randomness. Intuitively, pseudorandomness is used to generate objects that *appear* random despite containing only some small amount of randomness.

3.1 Pseudorandom Generators (Informal)

Informally, a pseudorandom generator (PRG) is a function $G : \{0,1\}^n \to \{0,1\}^m$, where $m > n$, the input consists of n random bits and the output "looks random" to any probabilistic polynomial time (PPT) algorithm. More formally, "looking random" means that no PPT adversary can "distinguish" between a string generated by G and a truly random string. Before we formalize the definition of pseudorandomness and pseudorandom generators, we need to define some vital concepts.

3.1.1 Definitions

Definition 15 (Uniform Distribution)**.** Distribution X is called a uniform distribution over an n-bit string if for every $s \in \{0,1\}^n$ the following holds:

$$\Pr[x = s : x \xleftarrow{\$} X] = \frac{1}{2^n}$$

Definition 16 (Identical Distributions)**.** Distributions X_0 and X_1 are said to be identical distributions if for every s the following holds:

$$\Pr[x = s : x \xleftarrow{\$} X_0] = \Pr[x = s : x \xleftarrow{\$} X_1]$$

Definition 17 (Distribution Ensembles)**.** A sequence $\{X^n\}_{n\in N}$ is called a distribution ensemble (or distribution family) if for each $n \in N$, X^n is a distribution over $\{0,1\}^n$.

For example, X^n can be the distribution of all uniform n-bit strings.

3.2 COMPUTATIONAL INDISTINGUISHABILITY

Before discussing pseudorandomness, it is necessary to discuss the notion of computational indistinguishability. The term "computationally indistinguishablity" formalizes the notion of two distributions looking "identical to a PPT adversary."

First Attempt. The distribution ensembles $\{X_0^n\}_n$ and $\{X_1^n\}_n$ are "identical to a PPT adversary" if for all s the following holds:

$$|\Pr[x = s : x \xleftarrow{\$} X_0^n] - \Pr[x = s : x \xleftarrow{\$} X_1^n]| \leq negl(n)$$

Remark 10. Now we can see why the distribution ensembles (as opposed to distributions) are important: they give us the parameter n used in asymptotic notions like noticeable and negligible. In the following, we will simply represent them as X_0^n and X_1^n (as opposed to $\{X_0^n\}$ and $\{X_1^n\}$)

Unfortunately, this definition is flawed. Let us consider the following distribution ensembles:

- $\{X_0^n\}_n$ represents all even numbers of length n.
- $\{X_1^n\}_n$ represents all odd numbers of length n.

Let us consider the case where s is even. We know that the total number of elements in both ensembles is 2^n, and thus, the number of elements in each ensemble is 2^{n-1}.

$$\Pr[x = s : x \xleftarrow{\$} X_0^n] = \frac{1}{2^{n-1}}$$

$$\Pr[x = s : x \xleftarrow{\$} X_1^n] = 0$$

Substituting in the calculations from above:

$$|\Pr[x = s : x \xleftarrow{\$} X_0^n] - \Pr[x = s : x \xleftarrow{\$} X_1^n]| = \frac{1}{2^{n-1}} \leq negl(n)$$

$\frac{1}{2^{n-1}}$ is a negligible function, which by the definition above means $\{X_0^n\}_n$ and $\{X_1^n\}_n$ are indistinguishable. However, this definitely should not be the case, as the two distributions are totally different and can easily be distinguished. Thus, this definition is flawed, so we will need to find an alternate and correct definition.

Definition 18 (Computational Indistinguishability)**.** Two distribution ensembles $\{X_0^n\}_n$ and $\{X_1^n\}_n$ are *computationally indistinguishable* if for every PPT adversary A,

$$|\Pr[A(x) = 0 : x \xleftarrow{\$} X_0^n] - \Pr[A(x) = 0 : x \xleftarrow{\$} X_1^n]| \leq negl(n)$$

$A(x)$ can be thought of as a guess by A regarding which ensemble x is coming from. We require this guess to not change as we change the ensemble.

Example. Consider the following uniform distribution ensembles:

- $\{X_0^n\}_n$ represents all even strings of length n.
- $\{X_1^n\}_n$ represents all odd strings of length n.

Let us say the PPT adversary A outputs 0 if x is even and 1 if x is odd. We get:

$$\Pr[A(x) = 0 : x \xleftarrow{\$} X_0^n] = 1$$

$$\Pr[A(x) = 0 : x \xleftarrow{\$} X_1^n] = 0$$

Using the definition of computational indistinguishability,

$$|\Pr[A(x) = 0 : x \xleftarrow{\$} X_0^n] - \Pr[A(x) = 0 : x \xleftarrow{\$} X_1^n]| = 1 \geq negl(n)$$

Thus, $\{X_0^n\}_n$ and $\{X_1^n\}_n$ do not satisfy the definition of computational indistinguishability, which we intuitively know clearly must be true.

The notion of computational indistinguishability allows us to define the computational indistinguishability advantage:

Definition 19 (Distinguishing Advantage)**.** The *distinguishing advantage* for ensembles X_0^n and X_1^n is ϵ if for all PPT A the following holds:

$$\epsilon \geq |\Pr[A(x) = 0 : x \xleftarrow{\$} X_0^n] - \Pr[A(x) = 0 : x \xleftarrow{\$} X_1^n]|$$

Now, let us examine an alternate way to determine whether two distribution ensembles are computationally indistinguishable.

Additionally, we introduce related notions of prediction advantage and prediction indistinguishability:

Definition 20 (Prediction Advantage)**.** Distribution ensembles $\{X_0^n\}_n$ and $\{X_1^n\}_n$ have *prediction advantage* ϵ if for all PPT A the following holds:

$$|\Pr[A(x) = b : b \xleftarrow{\$} \{0,1\}, x \xleftarrow{\$} X_b^n] - \frac{1}{2}| \leq \epsilon$$

Definition 21 (Prediction Indistinguishability)**.** We say that the distribution ensembles $\{X_0^n\}_n$ and $\{X_1^n\}_n$ have *prediction indistinguishability* if the prediction advantage ϵ is negligible.

We can now ask the question of how the notions of prediction indistinguishability and computational indistinguishability are related. Does having prediction indistinguisability imply that there is computational indistinguishability and vice versa?

Lemma 1. If $\{X_0^n\}_n$ and $\{X_1^n\}_n$ are computationally indistinguishable, then they also have prediction indistinguishability and vice versa.

Proof. Assume that the distribution ensembles $\{X_0^n\}_n$ and $\{X_1^n\}_n$ have prediction advantage ϵ_p. Then for all PPT A:

$$\begin{aligned}
\epsilon_p &\geq |\Pr[A(x) = b : b \xleftarrow{\$} \{0,1\}, x \xleftarrow{\$} X_b^n] - \frac{1}{2}| \\
&= |\Pr[A(x) = 0 : x \xleftarrow{\$} X_0^n]\Pr[b = 0] + \Pr[A(x) = 1 : x \xleftarrow{\$} X_1^n]\Pr[b = 1] - \frac{1}{2}| \\
&= |\frac{1}{2}\Pr[A(x) = 0 : x \xleftarrow{\$} X_0^n] + \frac{1}{2}\Pr[A(x) = 1 : x \xleftarrow{\$} X_1^n] - \frac{1}{2}| \\
&= \frac{1}{2}|\Pr[A(x) = 0 : x \xleftarrow{\$} X_0^n] + \Pr[A(x) = 1 : x \xleftarrow{\$} X_1^n] - 1| \\
&= \frac{1}{2}|\Pr[A(x) = 0 : x \xleftarrow{\$} X_0^n] + (1 - \Pr[A(x) = 0 : x \xleftarrow{\$} X_1^n]) - 1| \\
&= \frac{1}{2}|\Pr[A(x) = 0 : x \xleftarrow{\$} X_0^n] + 1 - \Pr[A(x) = 0 : x \xleftarrow{\$} X_1^n] - 1| \\
&= \frac{1}{2}|\Pr[A(x) = 0 : x \xleftarrow{\$} X_0^n] - \Pr[A(x) = 0 : x \xleftarrow{\$} X_1^n]|
\end{aligned}$$

$|\Pr[A(x) = 0 : x \xleftarrow{\$} X_0^n] - \Pr[A(x) = 0 : x \xleftarrow{\$} X_1^n]|$ is the expression for the distinguishing advantage, so we get

$$\epsilon_p \geq \frac{1}{2}\epsilon_c,$$

where ϵ_c is the distinguishing advantage. If the distribution ensembles have prediction indistinguishability, ϵ_p is negligible, and thus ϵ_c is negligible as well. The reverse direction can be shown similarly by beginning with a computational indistinguishability assumption.

Theorem 7. The computational indistinguishability advantage and the prediction advantage are within a factor of 2 of each other.

Now that we understand what computational indistinguishability entails, we can discuss some of the properties of computational indistinguishability.

3.3 Properties of Computational Indistinguishability

1. Notation: If distribution ensembles $\{X_0^n\}_n$ and $\{X_1^n\}_n$ are computationally indistinguishable, then we write
$$\{X_0^n\}_n =_c \{X_1^n\}_n$$
2. Closure: If $\{X_0^n\}_n =_c \{X_1^n\}_n$, then for every PPT algorithm A,
$$\{A(X_0^n)\}_n =_c \{A(X_1^n)\}_n$$
Proof. Let us assume that the distribution ensembles $\{X_0^n\}_n$ and $\{X_1^n\}_n$ are computationally indistinguishable and A is a PPT algorithm. Assume there exists a PPT

algorithm B that distinguishes $\{A(X_0^n)\}_n$ and $\{A(X_1^n)\}_n$ with some non-negligible advantage $p(n)$:

$$|\Pr[B(x) = 0 : x \xleftarrow{\$} A(X_0^n)] - \Pr[B(x) = 0 : x \xleftarrow{\$} A(X_1^n)]| \geq p(n)$$

Rewriting the equation, we get:

$$|\Pr[B(A(x)) = 0 : x \xleftarrow{\$} X_0^n] - \Pr[B(A(x)) = 0 : x \xleftarrow{\$} X_1^n]| \geq p(n)$$

Thus, the PPT algorithm $B(A(\cdot))$ (algorithm that works by first executing A on its input, then executing B on the output of A, and finally outputting whatever B outputs) distinguishes $\{X_0^n\}_n$ and $\{X_1^n\}_n$ with probability at least $p(n)$, which contradicts the initial assumption that the distribution ensembles are computationally indistinguishable. □

3. Transitivity: If the computational indistinguishability advantage between $\{X_n\}$ and $\{Y_n\}$ is ϵ_1, and the computational indistinguishability advantage between $\{Y_n\}$ and $\{Z_n\}$ is ϵ_2, then the computational indistinguishability advantage between $\{X_n\}$ and $\{Z_n\}$ is $\leq \epsilon_1 + \epsilon_2$.

   Proof. Using the triangle inequality,

$$\begin{aligned}
\epsilon &= |\Pr[A(x) = 0 : x \xleftarrow{\$} X_n] - \Pr[A(x) = 0 : x \xleftarrow{\$} Z_n]| \\
&= |\Pr[A(x) = 0 : x \xleftarrow{\$} X_n] - \Pr[A(x) = 0 : x \xleftarrow{\$} Y_n] + \Pr[A(x) = 0 : x \xleftarrow{\$} Y_n] - \Pr[A(x) = 0 : x \xleftarrow{\$} Z_n]| \\
&\leq |\Pr[A(x) = 0 : x \xleftarrow{\$} X_n] - \Pr[A(x) = 0 : x \xleftarrow{\$} Y_n]| + |\Pr[A(x) = 0 : x \xleftarrow{\$} Y_n] - \Pr[A(x) = 0 : x \xleftarrow{\$} Z_n]| \\
&\leq \epsilon_1 + \epsilon_2
\end{aligned}$$

With the knowledge of the properties of computational indistinguishability, we can examine a useful lemma that is used in many proofs that contain so-called hybrid arguments:

Lemma 2 (Hybrid Lemma)**.** Let $X_1, X_2, \ldots, X_m$ be distribution ensembles where $m = poly(n)$. If there exists a PPT algorithm D that has computational indistinguishability advantage ϵ with respect to X_1 and X_m, then there exists an $i \in [1, 2 \ldots, m-1]$ such that D distinguishes X_i and X_{i+1} with advantage at least $\frac{\epsilon}{m}$.

Proof. Let ϵ_i denote the computational indistinguishability advantage between X_i and X_{i+1}. Assume that $\epsilon_i \leq \frac{\epsilon}{m}\ \forall i$. Using the transitivity property,

$$\epsilon \leq \sum_{i=1}^{m-1} \epsilon_i \Rightarrow \epsilon \leq \epsilon \frac{m-1}{m}$$

This cannot be true, which means there exists an i such that $\epsilon_i \geq \frac{\epsilon}{m}$. □

Example We will now informally discuss a very intuitive hybrid argument example. Suppose you are a car manufacturer and you can either buy branded car parts or generic car parts for the car. All your lab tests show that branded and generic parts can't be distinguished. But you are worried that when you actually put together all the generic parts,

overall the car might break down or quality will go down noticeably. Can you prove that its ok to build a full car from generic parts assuming that each part in isolation is indistinguishable from branded?

Intuitively, we can define the hybrids where in each hybrid one more part (motor, then brakes, then lights etc.) is changed from branded to generic. Note that if we can distinguish one hybrid from the next, we can distinguish the branded from the generic version of the specific part by building a car that uses the given challenge part (for example, the motor) and then using the distinguisher for the two hybrids. Thus, any two consecutive hybrids are indistinguishable. Since there are only polynomially many car parts, we can conclude that the security holds by the hybrid lemma.

In the context of computational indistinguishability, one question is particularly interesting: can seeing *multiple* samples help the adversary distinguish between distributions?

Theorem 8. Given two distribution ensembles $\{X_0^n\}_n$ and $\{X_1^n\}_n$, if it holds that

$$|\Pr[A(x) = 0 : x \xleftarrow{\$} X_0^n] - \Pr[A(x) = 0 : x \xleftarrow{\$} X_1^n]| \leq \epsilon$$

for every PPT algorithm A, then it holds that

$$|\Pr[A(x_1, \cdots, x_k) = 0 : x_1, \cdots, x_k \xleftarrow{\$} X_0^n] - \Pr[A(x_1, \cdots, x_k) = 0 : x_1, \cdots, x_k \xleftarrow{\$} X_1^n]| \leq k\epsilon$$

for every PPT algorithm A as well.

Proof. We begin by defining the following distributions:

$$H_0 = x_1, \cdots, x_k \text{ where } \forall i, x_i \xleftarrow{\$} X_0^n$$
$$H_1 = x_1, \cdots, x_k \text{ where } x_1 \xleftarrow{\$} X_1^n, \text{ and } x_2, \cdots, x_k \xleftarrow{\$} X_0^n$$
$$\cdots$$
$$H_k = x_1, \cdots, x_k \text{ where } \forall i, x_i \xleftarrow{\$} X_1^n$$

Now we prove the following lemma about these distributions:

Lemma 3. For all $i \in [k-1]$ holds:

$$|\Pr[A(x) = 0 : x \xleftarrow{\$} H_i] - \Pr[A(x) = 0 : x \xleftarrow{\$} H_{i+1}]| \leq \epsilon$$

Proof. Suppose the statement does not hold. Thus, there is a PPT adversary A that is able to distinguish between some two distributions H_i, H_{i+1} with advantage greater than ϵ. Given this A, we construct a PPT adversary B on the indistinguishability of X_0^n, X_1^n. B takes input a challenge u, where u is either sampled from X_0^n or X_1^n, and works as follows:

1 : Set $x_i = u$.
2 : Set $x_1, \cdots, x_{i-1} \xleftarrow{\$} X_0^n$.
3 : Set $x_{i+1}, \cdots, x_k \xleftarrow{\$} X_1^n$.
4 : Let $U = x_1, \cdots, x_k$.
5 : Run $A(U)$
6 : Output X_0 if A outputs H_i, and output X_1 otherwise.

Note that if u comes from X_0, then U comes from H_i, and if u comes from X_1, then U comes from H_{i+1}. Therefore, B succeeds whenever A succeeds, and thus B can distinguish X_0 and X_1 with advantage at least ϵ, which leads to contradiction.

□

The theorem follows from the lemma above and the Hybrid Lemma (Lemma 2), applied to H_0, H_k. □

This theorem shows that sampling multiple samples does not help the adversary much. Because when ϵ is some negligible function $\mathrm{negl}(n)$, so is $k\epsilon$.

3.4 PSEUDORANDOM ENSEMBLES

We are now ready to return to discuss pseudorandomness and provide formal definitions. Intuitively, distribution ensemble is pseudorandom if it looks like a uniform distribution to any PPT adversary:

Definition 22 (Pseudorandom Ensembles)**.** A distribution ensemble $\{X_n\}_n$ over $\{0,1\}^{l(n)}$ is a *pseudorandom ensemble (PRE)* if

$$\{X_n\}_n =_c \{U_{l(n)}\}$$

where $U_{l(n)}$ denotes a uniform distribution over $l(n)$ bits.

3.5 PSEUDORANDOM GENERATORS

Now, for pseudorandom generators, we intuitively want to not only have the pseudorandomness property itself, but we also want the PRG to be efficiently computable and be able to extend the length of the original string:

Definition 23 (Pseudorandom Generator (PRG))**.** A *pseudorandom generator* $G : \{0,1\}^n \to \{0,1\}^{l(n)}$ is a deterministic algorithm with the following properties:

1. $G(x)$ can be computed in polynomial time (PT) for every $x \in \{0,1\}^n$
2. $l(n) > n$
3. $\{G(x) : x \xleftarrow{\$} \{0,1\}^n\} =_c \{U_{l(n)}\}$

The *stretch* of the PRG G is defined as $|G(x)| - |x|$.

3.6 CONSTRUCTION OF PSEUDO-RANDOM GENERATORS

We will discuss how pseudo-random generators can be constructed based on OWP. We have only seen 1-1 OWF so far, but we will see a construction of a OWP later in the class.

3.6.1 Construction of 1-bit stretch PRG

We start by constructing a 1-bit stretch PRG using the HCP (recall Definition 12). Suppose f is a one-way permutation of n-bit input, and h is a HCP of f. Construct $G : \{0,1\}^n \to \{0,1\}^{n+1}$ to be $G(s) = f(s) \,||\, h(s)$. Here, $l(n) = n + 1$.

Claim 4. G is a PRG.

Proof. Obviously G runs in polynomial time and $l(n) > n$. Therefore we only have to prove $\{G(s) : s \xleftarrow{\$} \{0,1\}^n\} =_c U_{n+1}$. Define the following distributions

$$H_0 = \{f(s)|| \, h(s) : s \xleftarrow{\$} \{0,1\}^n\}$$
$$H_1 = \{f(s)|| \, U_1 : s \xleftarrow{\$} \{0,1\}^n\}$$
$$H_2 = U_{n+1}$$

By definition of HCP, we have

$$H_0 =_c H_1$$

Now, consider an arbitrary $x \in \{0,1\}^n$. Since f is a one-way permutation,we have

$$\forall x \in \{0,1\}^n \ \Pr[f(s) = x : s \xleftarrow{\$} \{0,1\}^n] = \frac{1}{2^n}.$$

Since H_1 is the concatenation of f with U_1, and H_2 is uniform, we have

$$\Pr[x = x' : x' \xleftarrow{\$} H_1\,] = \Pr[x = x' : x' \xleftarrow{\$} H_2] = \frac{1}{2^{n+1}},$$

which implies

$$H_1 = H_2.$$

Therefore, by the transitivity of computational indistinguishability, we get

$$H_0 =_c H_2.$$

Hence, G is a PRG. □

3.6.2 Construction of poly-stretch PRG

Given a 1-bit stretch PRG G, we can now construct a poly-stretch PRG $G_p : \{0,1\}^n \to \{0,1\}^{l(n)}$ ($l(n) = poly(n)$) as follows:

i) Take as input n-bit string s.

ii) Invoke 1-bit stretch PRG G and do the following

$$s_0 \leftarrow s$$
$$s_1 \,||\, b_1 \leftarrow G(s_0)$$
$$s_2 \,||\, b_2 \leftarrow G(s_1)$$
$$\cdots$$
$$s_{l(n)} \,||\, b_{l(n)} \leftarrow G(s_{l(n)-1})$$

iii) Output $b1 \,||\, \cdots \,||\, b_{l(n)}$.

Claim 5. G_p is a PRG.

Proof. It is obvious that $l(n) > n$. Since G runs in polynomial time and G_p invokes G $l(n)$ times, G_p also runs in polynomial time. Therefore, we only have to prove

$$\{G_p(s) : s \xleftarrow{\$} \{0,1\}^n\} =_c \{U_{l(n)}\}$$

Define H_i ($i \in [0, l(n)]$) be the distribution generated as follows

i) Take as input n-bit string s.

ii) Do the following

$$\begin{aligned}
&s_0 \leftarrow s \\
&s_1 \,||\, b_1 \leftarrow U_{n+1} \\
&\cdots \\
&s_i \,||\, b_i \leftarrow U_{n+1} \\
&s_{i+1} \,||\, b_{i+1} \leftarrow G(s_i) \\
&\cdots \\
&s_{l(n)} \,||\, b_{l(n)} \leftarrow G(s_{l(n)-1})
\end{aligned}$$

iii) Output $b1 \,||\, \cdots \,||\, b_{l(n)}$.

First we prove the following lemma.

Lemma 4. For every $0 \le i \le l(n)$, we have $H_i =_c H_{i-1}$

Proof. Assume that the statement is not true. Then, there exists a PPT algorithm A that can distinguish H_i from H_{i-1} with some noticeable advantage δ. Then we can construct a PPT algorithm B that distinguishes G from U_{n+1} with the same advantage δ. B takes as input the challenge u, which is either from U_{n+1} or from $G(\cdot)$ applied to a random seed, and works as follows:

1 :	Generate $b_1, \cdots, b_{i-1}$ at random.
2 :	Set $s_i \,||\, b_i \leftarrow u$.
3 :	Set $s_j \,||\, b_j \leftarrow G(s_{j-1})$ for every $j > i$.
4 :	Run $A(b_1 \| \cdots \| b_{l(n)})$.
5 :	Output U_{n+1} if A outputs H_i, and output G otherwise.

Consider $b_1 \| \cdots \| b_{l(n)}$. If u is from U_{n+1}, then it comes from H_i, and if u is from G, then it comes from H_{i-1}. Therefore, B succeeds whenever A succeeds and thus B distinguishes G from U_{n+1} with a noticeable advantage, which leads to contradiction. □

Note that

$$H_0 = \{G_p(s) : s \xleftarrow{\$} \{0,1\}^n\}.$$

The claim follows from the lemma above and the Hybrid Lemma (see 2), applied to H_0 and $H_{l(n)} = U_{l(n)}$. □

3.7 APPLICATION: ONE-TIME PAD KEY GENERATION

Recall that the one-time pad (see 1.2.1) requires that the length of key equals the length of the message. So how can we encrypt a long message with only a short key? The idea is that using a PRG, we can generate a long key from the short one. Since PRGs are deterministic, we only have to transmit the short key, and the recipient of the message will be able to compute the long key using the short one by himself. Thus, now we can encrypt arbitrarily long messages using a short key. However, this approach has the following limitation: the short key can only be used once. As we will see later, pseudorandom functions will allow us to overcome this limitation.

3.8 PSEUDORANDOM FUNCTIONS

Before we discuss pseudorandom functions, we need to understand what a random function is. Intuitively, it is a function f that maps each unique input to an output chosen uniformly at random from the output domain (same inputs result in same outputs). It can be thought of as a large table that maps each input x to an output $f(x)$. If the input consists of n bits and the output of m bits, such table is of size $2^n \cdot m$ bits. Clearly, this table becomes very large very fast and we would like to have a more efficient alternative.

As we have seen, pseudorandom generators (PRGs) provide a way to generate a long string of bits – which is computationally indistinguishable from a string generated uniformly at random – from a shorter string of bits. Now we would like to construct something even more powerful – something that allows us to generate exponentially many pseudorandom bits from a short string of bits. However, there is a catch - since our construction should still run in polynomial time, there is no time to even write down such exponentially long string. Thus, we will settle for random access to this pseudorandom string:

Definition 24 (Pseudorandom function)**.** A function $F : \{0,1\}^n \times \{0,1\}^{m_1} \to \{0,1\}^{m_2}$ is called a *pseudorandom function* (PRF) if it satisfies the following conditions:

- F is deterministic and, for all $s \in \{0,1\}^n$ and all $x \in \{0,1\}^{m_1}$, $F(s,x)$ can be computed in polynomial time.
- For all PPT algorithms D,

$$\Pr[D \text{ wins the guessing game}] \leq \frac{1}{2} + \text{negl}(n)$$

where *the guessing game* is defined as follows:
We have a challenger C and a PPT adversary D.

1. C samples $s \xleftarrow{\$} \{0,1\}^n$ and a bit $b \xleftarrow{\$} \{0,1\}$.
2. For $i = 1, \ldots, q$ (where q is polynomial in n):
   a) D chooses $x_i \in \{0,1\}^{m_1}$ and sends it to C.
   b) If $b = 0$, C replies with $F(s, x_i)$. If $b = 1$, C picks $y_i \in \{0,1\}^{m_2}$ uniformly at random and sends it to D. If D has already queried x_i (i.e. $x_i = x_j$ for some $j < i$), then C responds with y_j (the same string as the last time).
3. Finally, D outputs a guess for b. If D is correct, then D wins the game.

Essentially, F is a PRF if F is deterministic and poly-time, but no PPT adversary D can, given polynomially-many queries to $F(s,\cdot)$ for a uniformly random seed s, distinguish $F(s,\cdot)$ from a truly random function $F_R : \{0,1\}^{m_1} \to \{0,1\}^{m_2}$.

We will now give an example of how to construct a PRF, given a PRG.

3.8.1 Construction

Let $g : \{0,1\}^n \to \{0,1\}^{2n}$ be a PRG, and for $s \in \{0,1\}^n$, let $g_0(s)$ denote the first n bits of $g(s)$ and $g_1(s)$ the last n bits of $g(s)$.

Define $F : \{0,1\}^n \times \{0,1\} \to \{0,1\}^n$ as follows:

$$F(s,x) = \begin{cases} g_0(s) & \text{if } x = 0 \\ g_1(s) & \text{if } x = 1 \end{cases}$$

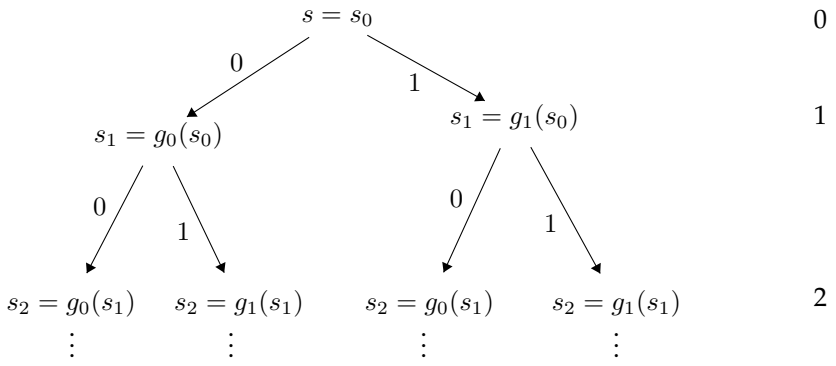
We will now use recursion to generalize this approach. Specifically, we can define $F : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ in the following way: given a seed $s \in \{0,1\}^n$ and an input $x \in \{0,1\}^m$, define a sequence of n-bit strings:

$$\begin{aligned} s_0 &= s \\ s_1 &= g_{x_1}(s_0) \\ s_2 &= g_{x_2}(s_1) \\ &\vdots \\ s_m &= g_{x_m}(s_{m-1}) \end{aligned}$$

where x_i is the i^{th} bit of x. We then define $F(s,x) = s_m$.

Note that even though this construction provides n-bit output, we can generalize it to produce a longer output simply by applying a PRG to the output.

We will now show that this construction indeed defines a PRF. Intuitively, we can think of this procedure as traversing the following binary tree:



We begin at the root node (level 0) and traverse the tree according to the bits of x: if the first bit is 0, then we go to the left (the node marked $s_1 = g_0(s_0)$); if the first bit is 1, we go to the right. We proceed in this manner until we get to the bottom of the tree (which is at level m). Note that the number of leaves is exponential and hence we cannot naively apply the hybrid argument. This is because if we have an exponential number of hybrids changing

leaves to uniform one by one, even if the distinguishing advantage is negligible at each step, it could be noticeable overall. Hence, we need a smarter approach. Since the height of the tree is $m + 1$ (which is polynomial in m), the number of queries to g is polynomial. Intuitively, we will change responses to all these queries to uniform. However, it turns out that we can't change responses one by one. This is because responses are generated using the same tree. Instead, we will conduct a hybrid over the levels of the tree.

Proposition 5. The function F defined above is a PRF.

Proof. Since g is deterministic, F is deterministic as well. Additionally, F is polynomial-time in n (we run g, which is assumed to be poly-time, m times, where m is polynomial in n). Thus, it suffices to check the second condition of the definition of PRF.

As mentioned, we will conduct a hybrid argument to show that $F(s, x)$ is computationally indistinguishable from uniform random output for every x that the adversary queries, and this will allow us to conclude that the second condition of the PRF definition is satisfied. We define the following hybrids:

H_0: Pick string s_0 on level 0 uniformly at random, and calculate all the subsequent s_i's as above.

H_i: Mark all nodes visited during the evaluation of adversary's queries, pick strings corresponding to the nodes visited by the adversary on level i uniformly at random. Note that strings corresponding to nodes up to and including level i are random, further down we calculate all s_is using the PRG as defined by the construction above. Note that we do not need to pick nodes that are above level i since they are not used in the evaluation of the queries.

$\vdots$

H_m: All strings corresponding to the nodes visited during the evaluation of adversary's queries for level 0 and up to and including level m are random.

Intuitively, when evaluating an output for some string x in hybrid H_i, we just skip the first $i-1$ levels, pick a random string for the node visited at level i and continue the evaluation as usual from there. Notice that H_m is just picking all the s_i uniformly at random. Now, it just suffices to show that the output of H_i and H_{i+1} are computationally indistinguishable.

Lemma 5. The probability in the guessing game can't change more than by some negligible amount when switching from hybrid H_i to hybrid H_{i+1}.

Proof. Notice that, if an adversary D makes queries in the $i/i+1$-th experiment, and D's challenger is using H_i, then the challenger generates all nodes up to the ith level uniformly at random, but for the nodes on the $(i + 1)$-th level, the challenger uses the PRG g, as described above. In contrast, if the challenger is using H_{i+1}, then the nodes on the $(i + 1)$-st level are generated uniformly at random as well. We can define a series of hybrids based on the number q of the queries the adversary makes as follows:

$H_i[0]$: Same as H_i.

$H_i[1]$: For the first query made by D, generate the node v visited during the evaluation of D's query on level $i + 1$ and its sibling uniformly at random, but use g for all subsequent queries (unless the node v or its sibling is visited multiple times, then return the previously generated values).

$\vdots$

$H_i[q]$: For all queries made by D, generate the nodes visited during the evaluation of D's query on level $i+1$ and its sibling uniformly at random (if a node or its sibling is visited multiple times for different queries of D, the same string is returned each time).

Notice that for $j \in \{0, \ldots, q-1\}$, $H_i[j] =_c H_i[j+1]$ holds by the security of the PRG g. Since $H_i[0] = H_i$ and $H_i[q] = H_{i+1}$ and q is polynomial, by the hybrid lemma we get that H_i and H_{i+1} are computationally indistinguishable. □

Thus, by the hybrid argument, the output of F is computationally indistinguishable from uniformly-random output, and we are done. □

3.8.2 PRF Application to Encryption

We can use pseudorandom functions to ensure secure message communication. For example, suppose Alice had a message m she wanted to securely communicate to Bob. First, they agree on a PRF F and a seed value s. Then, to securely communicate a message m, Alice does the following:

1. Generate a random x of the appropriate length.
2. Calculate $c = F(s, x) \oplus m$, and send the ciphertext (c, x) to Bob.
3. Bob calculates $F(s, x)$ and recovers $m = c \oplus F(s, x)$.

If m is longer than the length of input F accepts, then Alice can send m in several pieces, generating a fresh x each time (and each time sending it to Bob).

Remark 11. Another option is to apply a PRG to the PRF output to expand its length. This construction is analyzed in more detail in future chapters.

Fun with Composition
Part 1 – One-Way Functions

We now consider a few examples focusing on one-way functions.

Example 1 Suppose we have two OWFs f_1 and f_2. Let $g(x) = f_1(x)||f_2(x)$. Is g a OWF?

Answer: No.

Proof. Consider the following f_1 and f_2:

- Let $x = x_1||x_2$ and f be a OWF.
- Define $f_1(x) = f_1(x_1||x_2) = x_1||f(x_2)$ and $f_2(x) = f_2(x_1||x_2) = x_2||f(x_1)$.

We know that both $f_1(x)$ and $f_2(x)$ are one-way functions (see Proposition 1 for proof). Thus, we can use $f_1(x)$ and $f_2(x)$ to construct $g(x)$:

$$g(x) = g(x_1||x_2) = f_1(x_1||x_2)||f_2(x_1||x_2) = x_1||f(x_1)||x_2||f(x_2)$$

Clearly, g leaks the entirety of the input x, so it is not a OWF (an adversary can guess the input x with probability 1 by extracting x_1 and x_2 from the output).

The intuition behind why g fails the definition for a OWF is because we are given the outputs of *two* OWFs applied to x. $f_1(x)$ is hard to invert only if x is uniform. However, given $f_2(x)$, x is no longer uniform (i.e., easy to tell apart x from a random number). Similar reasoning holds for $f_2(x)$. □

Example 2 Suppose f is a OWF. Let $g(x) = f(x)||f(2x)$. Is g a OWF?

Answer: No.

Proof. Let $f'(x)$ be some OWF. Let x_1 denote the first half of the bit representation of x, and x_2 the second. Similarly, let $(\frac{x}{2})_1$ and $(\frac{x}{2})_2$ denote the first and second halves of the bit representation of the value $\frac{x}{2}$.

Now consider the following function $f(x)$:

$$f(x) = \begin{cases} (\frac{x}{2})_1||f'((\frac{x}{2})_2) & x \text{ even} \\ x_2||f'(x_1) & x \text{ odd} \end{cases}$$

Note that $f(x)$ is a OWF (proof similar to that of Proposition 1). Thus, we can use $f(x)$ to construct $g(x)$:

$$g(x) = \begin{cases} (\frac{x}{2})_1||f'((\frac{x}{2})_2)||x_1||f'(x_2) & x \text{ even} \\ x_2||f'(x_1)||x_1||f'(x_2) & x \text{ odd} \end{cases}$$

We can design a PPT adversary A against g as follows:

1. Assume x is odd (note that this happens with probability $\frac{1}{2}$)
2. Then $g(x) = f(x)||f(2x) = x_2||f'(x_1)||(\frac{2x}{2})_1||f'((\frac{2x}{2})_2) = x_2||f'(x_1)||x_1||f'(x_2)$.
3. Output $x = x_1||x_2$

Note that if the input x is indeed odd, A succeeds with probability 1. Thus, overall A succeeds with the non-negligible probability $\frac{1}{2}$. The reason why g does not have the same security as f is because an adversary against g has access to both $f(x)$ and $f(2x)$. Certain OWFs for f may leak different information given either $f(x)$ or $f(2x)$, which is sufficient to construct A to invert $g(x)$ with a non-negligible probability. In other words, given $f(x)$, the input of $f(2x)$ is not uniformly random, so it is possible that $f(2x)$ will leak information which combined with $f(x)$ can help to invert $f(x)$. □

Example 3 Suppose f is a deterministic OWF. Let $g(x||x') = f(x)||f(x')$. Is g a OWF?

Answer: Yes.

Proof. Main idea is: given $f(x')$, x is still uniform and vice versa. We will now provide a formal proof. Suppose g is not a OWF. Then there exists a PPT adversary A which inverts g with some non-negligible probability α. Using A, we can design the adversary B that inverts f with the same probability α as follows:

1. Given the challenge $f(x)$, sample $x' \xleftarrow{\$} U_n$ and let $y = f(x)||f(x')$.
2. Since x' was sampled uniformly from U_n, we can claim that $x||x'$ comes from the distribution equal to the distribution expected as input by g, and thus y comes from the same distribution as the one expected by A. Note that we were not able to take this step in Example 1 or 2 because there was no way to compute $f_2(x)$ or $f(2x)$.
3. Run $x^* = A(y)$. Since A succeeds with probability α, $g(x^*) = y$ with probability α. Let x_1 denote the first half of some string x, and x_2 the second half. If $g(x^*) = y$, then it holds that $g(x^*) = g(x_1^*||x_2^*) = f(x_1^*)||f(x_2^*) = y = f(x)||f(x')$, and thus in particular $f(x_1^*) = f(x)$. Thus, when we output x_1^* as our guess for x, we succeed with the same probability α.

Since B inverts f with non-negligible probability α and thus breaks the security of the OWF f, we have reached a contradiction and g must therefore indeed be a OWF. □

Symmetric Key Encryption

Recall that we already saw an example of a perfectly secure symmetric encryption – the one-time pad. Although the one-time pad is perfectly secure, it requires that the key has the same length as the message. In fact, Shannon's Theorem tells us that any perfectly secure symmetric key encryption scheme requires a key which is at least as long as the message. Obviously, this is not very practical. To bypass this limitation, we need to relax the security definition. We will now provide a formal definition of symmetric key encryption and its relaxed security notion:

4.1 Syntax of Symmetric Key Encryption

Definition 25. A symmetric key encryption scheme consists of three algorithms (Gen, Enc, Dec) defined below:

- $k \leftarrow Gen(\kappa)$. The PPT *key generation* algorithm Gen takes as input a security parameter κ [3] and generates a secret key k.
- $c \leftarrow Enc(k, m)$. The PPT *encryption* algorithm Enc takes as input a key k and a message m, and outputs a ciphertext c.
- $m \leftarrow Dec(k, c)$. The deterministic polynomial time *decryption* algorithm Dec takes as input a key k and a ciphertext c, and outputs a plaintext message m.

Additionally, symmetric key encryption must satisfy correctness:

Correctness. The scheme is said to be correct, if for all security parameters κ and all messages m the following holds:

$$Pr[m = Dec(k, c) : k \leftarrow Gen(\kappa), c \leftarrow Enc(k, m)] = 1$$

Recall that we had the following definition for perfect security of the encryption scheme: (Gen, Enc, Dec) is a perfectly secure encryption scheme if and only if for all pairs of messages (m_0, m_1) and all ciphertexts c the following holds:

$$Pr[c = Enc(k, m_0) : k \leftarrow Gen(\cdot)] = Pr[c = Enc(k, m_1) : k \leftarrow Gen(\cdot)]$$

Equivalently, we could have said that for all pairs of messages (m_0, m_1) it holds that $\{Enc(k, m_0) : k \leftarrow Gen(\cdot)\} = \{Enc(k, m_1) : k \leftarrow Gen(\cdot)\}$.

We now introduce the following definition for indistinguishability based security of symmetric encryption:

[3] Typically we use 1^κ as a security parameter to make sure that Gen runs in time polynomial in the size of the input. For simplicity, we will just write κ.

Definition 26 (indistinguishability based security)**.** (Gen, Enc, Dec) is an *indistinguishability based secure* symmetric encryption scheme if and only if for all pairs of messages (m_0, m_1) the following holds:

$$\{Enc(k, m_0) : k \leftarrow Gen(n)\} =_c \{Enc(k, m_1) : k \leftarrow Gen(n)\}$$

The difference between perfect security and indistinguishable security is that perfect security requires the distributions of $\{E(k, m_0) : k \leftarrow G(n)\}$ and $\{E(k, m_1) : k \leftarrow G(n)\}$ to be identical, and indistinguishable security only requires them to be computationally indistinguishable. We additionally have the following alternative notion:

Definition 27 (indistinguishability based security, alternative)**.** (Gen, Enc, Dec) is an *indistinguishability based secure* symmetric encryption scheme if and only if for all pairs of messages (m_0, m_1) and all PPT adversaries $\mathcal{A}$ the following holds:

$$|Pr[\mathcal{A}(Enc(k, m_b)) = b : k \leftarrow Gen(n), b \leftarrow \{0, 1\}] - \frac{1}{2}| \leq negl(n)$$

While the first notion of indistinguishability based security is inspired by the computational indistinguishability definition, the second one is inspired by prediction indistinguishability.

Claim 6. Both definitions of indistinguishability based security are equivalent.

Proof. The proof is very similar to the proof of equivalence of computational and prediction indistinguishability. In the following, we show one direction:

Let (Gen, Enc, Dec) be a symmetric key encryption scheme with indistinguishable security as defined in Definition 26. Then, for any n, m_0, m_1, and PPT adversary $\mathcal{A}$, define:

- $p = Pr[\mathcal{A}(Enc(k, m_0)) = 0 : k \leftarrow Gen(n)]$
- $p' = Pr[\mathcal{A}(Enc(k, m_1)) = 0 : k \leftarrow Gen(n)]$

Since $\{Enc(k, m_0) : k \leftarrow Gen(n)\} =_c \{Enc(k, m_1) : k \leftarrow Gen(n)\}$, we know that $p - p' \leq negl(n)$. Additionally, $1 - p' = Pr[\mathcal{A}(Enc(k, m_1)) = 1 : k \leftarrow Gen(n)]$. Let $q = Pr[\mathcal{A}(Enc(k, m_b)) = b : k \leftarrow Gen(n), b \leftarrow \{0, 1\}]$, which is the probability of $\mathcal{A}$ outputting the correct bit b. Then, we get the following:

$$\begin{aligned}
|q - \frac{1}{2}| &= |Pr[\mathcal{A}(Enc(k, m_0)) = 0 : k \leftarrow Gen(n)]Pr[b = 0 : b \leftarrow \{0, 1\}] \\
&\quad + Pr[\mathcal{A}(Enc(k, m_1)) = 1 : k \leftarrow Gen(n)]Pr[b = 1 : b \leftarrow \{0, 1\}] - \frac{1}{2}| \\
&= |\frac{1}{2} \cdot p + \frac{1}{2} \cdot (1 - p') - \frac{1}{2}| \leq |\frac{1}{2} \cdot p + \frac{1}{2} \cdot (1 - p + negl(n)) - \frac{1}{2}| = |1/2 \cdot negl(n)|
\end{aligned}$$

□

4.2 One-Time Encryption Using a PRG

Given a pseudo-random generator $F : \{0, 1\}^n \to \{0, 1\}^{l(n)}$, we can define a symmetric key encryption scheme with indistinguishable security as follows:

- $Gen(n)$: sample a PRG seed $k \xleftarrow{\$} \{0, 1\}^n$

- $Enc(k, m)$, $m \in \{0,1\}^{l(n)}$: Output $c = m \oplus PRG(k)$
- $Dec(k, c)$: Output $m = c \oplus PRG(k)$

It is clear that Gen, Enc, and Dec are all polynomial time algorithms, since PRG(k) is poly-time by definition, and $\oplus$ is linear.

Correctness holds because given any m and k, $Dec(k, Enc(k, m)) = Dec(k, m \oplus PRG(k)) = (m \oplus PRG(k)) \oplus PRG(k) = m$.

We now show that indistinguishable security holds:

Proof. We want to show that for all messages m_0, m_1 the following holds:

$$\{m_0 \oplus PRG(k) : k \xleftarrow{\$} \{0,1\}^n\} =_c \{m_1 \oplus PRG(k) : k \xleftarrow{\$} \{0,1\}^n\}$$

First, note that the following holds by the security of PRG:

$$\{m_0 \oplus PRG(k) : k \xleftarrow{\$} \{0,1\}^n\} =_c \{m_0 \oplus R : R \xleftarrow{\$} \{0,1\}^{l(n)}\},$$
$$\{m_1 \oplus PRG(k) : k \xleftarrow{\$} \{0,1\}^n\} =_c \{m_1 \oplus R : R \xleftarrow{\$} \{0,1\}^{l(n)}\}$$

Now, note that by perfect security of the one-time pad we have

$$\{m_0 \oplus R : R \xleftarrow{\$} \{0,1\}^{l(n)}\} = \{m_1 \oplus R : R \xleftarrow{\$} \{0,1\}^{l(n)}\}$$

Thus, we get that $\{Enc(k, m_0) : k \leftarrow Gen(n)\} =_c \{Enc(k, m_1) : k \leftarrow Gen(n)\}$. □

4.3 MULTI-MESSAGE ENCRYPTION

Note that the previous construction works only for a single message. However, it is possible to slightly modify this construction to allow for encryption of multiple messages. Informally, we store a counter ctr which denotes the sum of the lengths of all messages $m_1...m_n$ that we sent. When we want to encrypt the $(n+1)$-st message m_{n+1}, we xor m_{n+1} with bits of $PRG(k)$ starting at $ctr+1$. After we send the message, we update the counter: $ctr = ctr + |m_{n+1}|$. Note that this doesn't satisfy the usual definition since an updatable counter is needed as input to the encryption and decryption algorithms. The kind of encryption that uses such counters (or any other form of state that is updated with each use of the encryption) is known as *stateful*.

We formally define the security of a multi-message symmetric key encryption scheme as follows:

Definition 28. For every choice of vectors $(m_0^1, m_0^2...m_0^q)$ and $(m_1^1, m_1^2...m_1^q)$, where $q = poly(n)$, the following holds:

$$\{\{Enc(k, m_0^i)\}_{i=1}^q : k \leftarrow Gen(n)\} =_c \{\{Enc(k, m_1^i)\}_{i=1}^q : k \leftarrow Gen(n)\}$$

Using this definition, we will now show that a secure multi-message symmetric key encryption scheme cannot be both stateless and deterministic (encryption that is both stateless and deterministic means that if a plaintext is encrypted multiple times, the same ciphertext is generated each time).

Theorem 9. A multi-message secure symmetric key encryption scheme cannot be both deterministic and stateless.

Proof. Let (Gen, Enc, Dec) be a multi-message secure symmetric key encryption scheme that is stateless and deterministic. Consider the following two message vectors: (m_0^0, m_0^1), where $m_0^0 = m_0^1$, and (m_1^0, m_1^1), where $m_1^0 \neq m_1^1$. Let $k \leftarrow Gen(n)$ be chosen at random. Since Enc is stateless and deterministic, encryption of the vector (m_0^0, m_0^1), where $m_0^0 = m_0^1$, results in a vector (c_0^0, c_0^1) with two identical ciphertexts ($c_0^0 = c_0^1$) because inputs to Enc are identical and Enc is deterministic. In contrast, encryption of the vector (m_1^0, m_1^1), where $m_1^0 \neq m_1^1$, results in a vector (c_1^0, c_1^1) with two different ciphertexts, since otherwise the decryptor would not know whether the ciphertext corresponds to m_1^0 or m_1^1 (contradiction to the correctness property of the encryption scheme).

Thus, a PPT adversary $\mathcal{A}$ can use these two message vectors to distinguish the distributions $\{\{Enc(k, m_0^i)\}_{i=1}^q : k \leftarrow Gen(n)\}$ and $\{\{Enc(k, m_1^i)\}_{i=1}^q : k \leftarrow Gen(n)\}$ with probability 1, contradicting the security of the given encryption scheme. □

4.3.1 Multi-Message Encryption using a PRF

We have shown that stateless non-randomized encryption schemes cannot be multi-message secure. However, stateless nondeterministic encryption schemes can be multi-message secure. We will now show an example of such a scheme.

Given a pseudo-random function $F : \{0,1\}^n \times \{0,1\}^{m_1} \rightarrow \{0,1\}^{m_2}$, we constuct an encryption scheme (Gen, Enc, Dec) as follows:

- $Gen(n)$: Output $k \xleftarrow{\$} \{0,1\}^n$
- $Enc(k, m)$: Sample $r \xleftarrow{\$} \{0,1\}^{m_1}$, set $c_1 = r$, compute $c_2 = m \oplus PRF(k, r)$, and output $c = (c_1, c_2)$.
- $Dec(k, (c_1, c_2))$: Output $c_2 \oplus PRF(k, c_1)$.

First, note that Gen, Enc, and Dec are all polynomial time algorithms, since PRF and xor can be computed in polynomial time.

To show correctness, note that the following holds for all messages m and keys k:

$$(m \oplus PRF(k, r)) \oplus PRF(k, r) = m$$

We will now show that (Gen, Enc, Dec) is indistinguishablility based secure using a hybrid argument:

Proof. Let $M_0 = \{m_0^i\}_{i=1}^q$ and $M_1 = \{m_1^i\}_{i=1}^q$ be two vectors of size q, where $q = poly(n)$. Let $k \xleftarrow{\$} \{0,1\}^n$. Then, we construct the following hybrids:

H_0: $\{(r^i, m_0^i \oplus PRF(k, r^i))\}_{i=1}^q$
H_1: $\{(r^i, m_0^i \oplus RF(r^i))\}_{i=1}^q$ where RF is a truly random function
H_2: $\{(r^i, m_0^i \oplus R^i : R^i \xleftarrow{\$} \{0,1\}^n)\}_{i=1}^q$
H_3: $\{(r^i, m_1^i \oplus R^i : R^i \xleftarrow{\$} \{0,1\}^n)\}_{i=1}^q$
H_4: $\{(r^i, m_1^i \oplus RF(r^i))\}_{i=1}^q$ where RF is a truly random function
H_5: $\{(r^i, m_1^i \oplus PRF(k, r^i))\}_{i=1}^q$

First, note that $H_0 =_c H_1$ because of the security guarantee of the PRF function: if there exists an adversary $\mathcal{A}$ that distinguishes H_0 and H_1 with some non-negligible probability, then we can construct another adversary $\mathcal{B}$ who can win the guessing game in the PRF definition with probability $\frac{1}{2} + notice(n)$. Until $\mathcal{A}$ provides an output, in each round i the adversary $\mathcal{B}$ works by sampling r^i, querying r^i from its challenger and using the response R^i to construct input $(r^i, m_0^i \oplus R^i)$ to its internal adversary $\mathcal{A}$. Then, $\mathcal{B}$ outputs "PRF" if $\mathcal{A}$ outputs "H_0", and "RF" otherwise. $\mathcal{B}$ wins with the same probability as $\mathcal{A}$, and since we assume that $\mathcal{A}$ wins with some non-negligible probability, we reach a contradiction to the security of the given PRF.

Now, note that if all the r^is were different, then H_1 and H_2 would be identical distributions. However, if there was a $r^i = r^j$, then $RF(r^i)$ would be the same as $RF(r^j)$, but at the same time R^i would not be the same as R^j with a very high probability. Thus, an adversary would be able to distinguish H_1 and H_2 with very high probability if there was a vector with a repeated string.

However, the probability of some r^i being equal to some r^j for fixed indices i and j is $\frac{1}{2^{m_1}}$, so the probability of any two r_i, r_j being equal is bounded by above by $\frac{\binom{q}{2}}{2^{m_1}}$ which is negligible. Therefore, any PPT adversary can only distinguish between H_1 and H_2 with negligible probability, so $H_1 =_c H_2$.

H_2 and H_3 are identical because one-time pad is perfectly secure. H_3 is induistinguishable from H_4 by the same reasoning as used in the proof of indistinguishability of H_1 and H_2. H_4 is induistinguishable from H_5 by the same reasoning as used in the proof of indistinguishability of H_0 and H_1. By the hybrid lemma, we are done. □

Fun with Hybrids
Part 2 – Encryption Schemes

We now continue with the hybrid lemma application examples. In this chapter, we will focus on the examples considering encryption schemes. In all the examples below, we will assume that the encryption schemes used as building blocks in the new construction are secure.

In the following, $Enc(k, m; r)$ denotes running the Enc algorithm with input (k, m) and random tape r.

Example 4 Does it hold that
$\text{Enc}(k, m_1; r)||\text{Enc}'(k', m_1; r') \approx_c \text{Enc}(k, m_2; r)||\text{Enc}'(k', m_2; r')$?

Answer: Yes.

Proof. Consider the following hybrids:

1. H_0: $\{\text{Enc}(k, m_1; r)||\text{Enc}'(k', m_1; r')\}$
2. H_1: $\{\text{Enc}(k, m_1; r)||\text{Enc}'(k', m_2; r')\}$
3. H_2: $\{\text{Enc}(k, m_2; r)||\text{Enc}'(k', m_2; r')\}$

Informally, H_0 is computationally indistinguishable from H_1 by the indistinguishability based security of Enc′. Similarly, H_1 is computationally indistinguishable from H_2 by the indistinguishability based security of Enc. Intuitively, we can step from H_0 to H_1 and from H_1 to H_2 since the parameters for each encryption in the concatenation are completely independent, so we do not break the requirement that the randomness and key is chosen uniformly at random. Unlike OWFs, encryption schemes do not require message to be chosen at random. □

Example 5 Does it hold that $\text{Enc}(k, m_1; r)||\text{Enc}'(k, m_1, r') \approx_c \text{Enc}(k, m_2; r)||\text{Enc}'(k, m_2, r')$?

Answer: No.

Proof. We now provide intuition for why the claim is not true. Let us try to use the same hybrid approach as in the previous example:

1. H_0: $\{\text{Enc}(k, m_1; r)||\text{Enc}'(k, m_1; r')\}$
2. H_1: $\{\text{Enc}(k, m_1; r)||\text{Enc}'(k, m_2; r')\}$

Already, we have made an incorrect step in our hybrid argument: $H_0 \not\approx_c H_1$. The reason for this is because both encryptions Enc and Enc′ use the same key k. It is possible for both encryptions to each leak some part of k, that when combined allows an adversary to use k to distinguish between $\{\text{Enc}'(k, m_1; r')\}$ and $\{\text{Enc}'(k, m_2; r')\}$ which would mean $H_0 \not\approx_c H_1$. (For example, suppose Enc leaked the first half of k and only used the second for security, and then Enc′ leaked the second half and used the first half for security). Hence the hybrid argument breaks down and we cannot continue.

To summarize: given $\text{Enc}'(k, m; r)$, k may no longer be uniform. □

Example 6 Does it hold that $\text{Enc}(k, m_1; r)||\text{Enc}(k, m_1; r') \approx_c \text{Enc}(k, m_2; r)||\text{Enc}(k, m_2; r')$?

Answer: No (depends).

Proof. Looking at the question another way, what it is really asking is if the scheme is multi-message secure. If it is not, then the hybrid argument fails:

1. H_0: $\{\text{Enc}(k, m_1; r)||\text{Enc}(k, m_1; r')\}$
2. H_1: $\{\text{Enc}(k, m_1; r)||\text{Enc}(k, m_2; r')\}$
3. H_2: $\{\text{Enc}(k, m_2; r)||\text{Enc}(k, m_2; r')\}$

Note that if Enc is multi-message secure, we can claim that H_0 is computationally indistinguishable from H_1 (and H_1 is computationally indistinguishable form H_2). If the scheme is not multi-message secure, we cannot make such claims about H_0 and H_1 (and H_1 and H_2). Intuitively, if we were to try to argue that Enc is not secure if H_0 and H_1 are distinguishable, we would fail since to construct an input to the distinguisher we would need to know k (to compute $\text{Enc}(k, m_1; r)$). □

Example 7 Does it hold that $\text{Enc}(k, m_1; r)||\text{Enc}'(k', m_1; r) \approx_c \text{Enc}(k, m_2; r)||\text{Enc}'(k', m_2; r)$?

Answer: No.

Proof. We provide an intuition. Similar to Example 5, this example fails the hybrid argument as for an encryption scheme, the randomness must be sampled uniformly at random. An adversary given both encryptions with the same randomness may have enough information to distinguish between the distributions of the encryption scheme on m_1 vs m_2. Exploring this further:

1. H_0: $\{\text{Enc}(k, m_1; r)||\text{Enc}'(k', m_1; r)\}$
2. H_1: $\{\text{Enc}(k, m_1; r)||\text{Enc}'(k', m_2; r)\}$

At this point, we cannot claim $H_0 \approx_c H_1$.

1. Suppose an adversary A exists which can distinguish H_0 from H_1 with non-negligible probability. Then we will attempt to build adversary B against Enc′.
2. Take A's choices for m_1 and m_2 and send them to the challenger for B. Receive a ciphertext c which is either $\text{Enc}'(k', m_1, r)$ or $\text{Enc}'(k', m_2, r)$.
3. In order to use A to win the game against the challenger for B, we need to construct an input which is indistinguishable from the proposed scheme's ciphertext. To do this, we need to compute $\text{Enc}(k, m_1, r)$ to prepend to c. However, we cannot do so since the random choice r is the same for Enc and Enc′, and we do not know it.

   To summarize: given $\text{Enc}(k, m_1; r)$, r may not be uniform.

□

Facts from Number and Group Theory

In the previous lecture, we introduced the syntax of the symmetric key encryption scheme, where the same key is used for encryption and decryption procedures. However, it is not always feasible for the sender and the recipient to share and agree on the same secret key. In subsequent lectures, we will introduce public-key encryption schemes where the encryption and decryption keys are different but mathematically related. Such schemes are based on the assumption that certain problems in number theory are hard. To better understand such schemes, we need to recap some basic facts from number theory and group theory.

In the following, we will use the following notation:

- $\mathbb{N} =$ set of all natural numbers ($\{1, 2, 3, \ldots\}$)
- $\mathbb{R} =$ set of all real numbers (e.g., 5.25)
- $\mathbb{Z} =$ set of all integers ($\{-\infty, \ldots, -1, 0, 1, \ldots, +\infty\}$)
- For integers (a, b), $gcd(a, b)$ denotes the *greatest common divisor* (the largest number which divides both a and b).

Remark 12. The Greatest Common Divisor (GCD) can be computed in polynomial time using Extended Euclidean Algorithm (EEA). Additionally, the EEA allows us to compute (x, y) for any (a, b) such that $ax + by = gcd(a, b)$.

Theorem 10. If a, b are relatively prime (i.e., $gcd(a, b) = 1$) and $a > b$, then there exists y s.t.

$$by = 1 \mod a$$

Proof. Using EEA, we can get x, y such that $ax + by = gcd(a, b) = 1$. Thus, $(ax + by) \mod a = 1 \mod a$. Therefore, we get $by = 1 \mod a$. □

Remark 13. If $by = 1 \mod a$, we say that y is an inverse of b with respect to a.

5.1 GROUPS

A group is a fundamental object in algebra and is widely used in constructing cryptographic algorithms. For example, the Diffie–Hellman key exchange makes use of finite cyclic groups. Before delving into cryptographic schemes that utilize groups, let us familiarize ourselves with some basic group theory concepts and see some examples of groups.

5.1.1 Basic Definitions

Definition 29. Let G be a set, and $\odot$: $G \times G \to G$ be an operation. A *group* $(G, \odot)$ is an algebraic structure that satisfies the following properties:

1. Closure: If $a, b \in G$, then $a \odot b \in G$.
2. Existence of Identity: $\exists e \in G$ such that for all $a \in G$, $a \odot e = e \odot a = a$.
3. Associativity: $\forall a, b, c \in G$, $(a \odot b) \odot c = a \odot (b \odot c)$.
4. Existence of Inverse: $\forall a \in G, \exists b \in G$ such that $a \odot b = e = b \odot a$.

Definition 30. An *Abelian group* G is a group such that $\forall a, b \in G, a \odot b = b \odot a$.

Definition 31. The *order* of a group G (denoted as $|G|$) is the number of elements in G.

5.1.2 Examples of Groups

Now we take a look at some examples of groups. An additive group is a group where the group operation $\odot$ is an addition of some type.

Example of an Additive Group

$(\mathbb{Z}_n, \odot)$ where $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}, \odot =$ addition mod n, is an additive group.

Proof.

1. Closure: if $a, b \in \mathbb{Z}_n$, then $a \odot b = (a + b) \mod n$. Since $0 \leq (a + b) \mod n \leq n - 1$, it holds that $a \odot b \in G$.
2. Existence of Identity: $e = 0$.
3. Associativity: This follows from the fact that addition modulo n is associative.
4. Existence of Inverse: Given $a \in \mathbb{Z}_n$, its inverse is $b = n - a$. Since $0 \leq b \leq n - 1$, $b \in \mathbb{Z}_n$.

□

Example of a Multiplicative Group

A multiplicative group is a group where the group operation $\odot$ is a multiplication of some type.

$(\mathbb{Z}_p^*, \odot)$ where p is a prime number, $\mathbb{Z}_p^* = \{1, 2, 3, \ldots p-1\}$ [4], and $\odot =$ multiplication mod n, is a multiplicative group.

Proof.

1. Closure: Suppose $a, b \in \mathbb{Z}_p^*$. Since a, b are relatively prime to p, we know that $a \cdot b \neq k \cdot p$ (note that the left-hand side is not divisible by p, and the right-hand side is). Since additionally $0 < (a \cdot b) \mod p < p$, it holds that $a \cdot b \in \mathbb{Z}_p^*$.
2. Existence of Identity: $e = 1$.
3. Associativity: This follows from the fact that multiplication modulo n is associative.
4. Existence of Inverse: Given $a \in \mathbb{Z}_p^*$, we know by Theorem 10 that there exists b such that $a \cdot b = 1 \mod p$. Note that $b = (b - p) \mod p$. Thus, there exists b such that $a \cdot b = 1 \mod p$ and $1 \leq b \leq p - 1$.

□

Remark 14. For $(\mathbb{Z}_p^*, \odot)$, the order of the group is $p - 1$.

[4] In general, $\mathbb{Z}_n^*$ denotes the set of all integers from the set $\{1, 2, 3, \ldots n-1\}$ relatively prime to n.

Fundamental Theorem of Arithmetic

The Fundamental Theorem of Arithmetic is also called the Unique Factorization Theorem. It is important to know this theorem since integer factorization into primes is an important tool for securing public-key encryption.

Theorem 11 (Fundamental Theorem of Arithmetic)**.** For all integer $N > 1$, there exists $k > 0$ different primes $p_1, p_2, \ldots, p_k$, $p_1 < p_2 < \ldots < p_k$, and k positive integers $\{e_1, e_2, \ldots, e_k\}$ such that N can be expressed as

$$N = \prod_i p_i^{e_i},$$

and moreover, this representation is unique.

Euler's Phi(Totient) Function

We introduce Euler's Phi Function (also sometimes called Euler's Totient Function), denoted $\phi(N)$, which is important for defining the RSA encryption scheme.

Definition 32 (Euler's Phi (Totient) Function)**.** *Euler's phi function* of a positive integer N, denoted $\phi(N)$, is the number of integers from set $\{1, 2, \ldots N-1\}$ which are relative prime to N. We have

$$\phi(N) = \prod_i p_i^{e_i - 1} \cdot (p_i - 1),$$

where $N = \prod_i p_i^{e_i}$.

Example 1 For p prime $\phi(p) = p^0 \cdot p^0 \cdot (p-1) = p - 1$.
Example 2 Say N is a product of two primes $N = p_1 p_2$, where $p_1 \neq p_2$. Then, $\phi(N) = (p_1 - 1)(p_2 - 1) = |\mathbb{Z}_N^*|$.

Theorem 12. For all $N \geq 2$, $\phi(N) = |\mathbb{Z}_N^*|$.

Theorem 13 (Fermat's Little Theorem)**.** $\forall a \in \mathbb{Z}_p^*$, $a^{p-1} \mod p = 1$.

Proof. Suppose $a \in \mathbb{Z}_p^*$, then a is relatively prime to p. Multiplying each element in $\mathbb{Z}_p^*$ by a, we get a set $a\mathbb{Z}_p^* = \{a, 2a, \cdots, (p-1)a\}$. By the laws of modular arithmetic, these elements are distinct from one another modulo p (if $ma = na \mod p$, then $m = n \mod p$, but we know that each element in $\mathbb{Z}_p^*$ is distinct). Therefore, $\mathbb{Z}_p^* = a\mathbb{Z}_p^*$. Thus, if we multiply the elements in both sets, the resulting products must be equal: $1 \cdot 2 \cdot \ldots \cdot (p-1) = a \cdot 2a \cdot \ldots \cdot (p-1)a \mod p$. We can now divide both sides by $1 \cdot 2 \cdot \ldots \cdot (p-1)$ to get $a^{p-1} \mod p = 1$. □

Theorem 14 (Euler's Generalization)**.** $\forall a \in \mathbb{Z}_n^*$, $a^{\phi(n)} \mod n = 1$.

Multiplicative Group of Prime Order

Definition 33. A group whose order is prime is called a *group of prime order*, or *prime-order group*.

Multiplicative groups of prime order are extremely important in cryptography. Unfortunately, for an integer $N > 3$, the order of $\mathbb{Z}_N^*$ (i.e., Euler's phi function of N) is not prime. Therefore, we cannot use multiplicative groups $\mathbb{Z}_N^*$ and need to find another way to construct a multiplicative group of prime order.

Example Construction of a Multiplicative Group of Prime Order Choose $p = 2q + 1$ where p, q are prime numbers (such p is called a Sophie-Germain prime.) Then, $G_q = \{x^2 \mod p : x \in \mathbb{Z}_p^*\}$ is a multiplicative prime-order group.

Theorem 15. $|G_q| = q$.

Proof. Note that both x and $p-x$ from $\mathbb{Z}_p^*$ map to the same element $x^2 = (p-x)^2 \mod p$ of G_q. Since $|\mathbb{Z}_p^*| = p - 1 = 2q$, it holds that $|G_q| \leq q$. Suppose x, y are two different elements in $\mathbb{Z}_p^*$ and $y \neq p - x$. Assume $x^2 = y^2 \mod p$. This means that $(x + y)(x - y) = 0 \mod p$. Since p is prime, either $x + y$ or $x - y$ divides p. We know that $x + y \neq 0 \mod p$, so it must hold that $x - y = 0 \mod p$. Thus, $x = y \mod p$, which contradicts the assumption that $x \neq y$. Therefore, we know that $x^2 \neq y^2 \mod p$. Thus, unless two elements in G_q are not inverses of each other in $\mathbb{Z}_p$, they map to different elements in G_q. Therefore, we have $|G_q| = q$. □

Generator of a Multiplicative Group

Definition 34. $g \in G$ is a *generator* of the group G if $G = \{g^1, g^2, g^3, \ldots, g^{|G|}\}$.

Note that $g^x = g \odot g \odot \cdots \odot g$ (x times).

Remark 15. Suppose $g \in G$ is a generator of the group G, then $g^{|G|}$ is the identity element in G.

Theorem 16. Every non-identity element of a multiplicative prime order group is a generator.

5.2 Discrete Logarithm Assumption and its Variations

Informally, the *Discrete Log Problem* for multiplicative prime-order groups is the problem that given such a group G_q and an element g^x, where g is a generator, we want to try to find x (x is called the *discrete logarithm* of g^x). In cryptography (in particular in certain public-key encryption schemes), it is assumed that for certain groups this problem is hard.

5.2.1 Discrete Log Assumption (DLA)

Definition 35 (Discrete Log Assumption, DLA)**.** Suppose G_q is a prime order multiplicative group, and $g \in G_q$ is a generator. Then, for every PPT algorithm A, the following holds:

$$\Pr[A(g^x) = x : x \xleftarrow{\$} \mathbb{Z}_q] \leq negl(q)$$

Theorem 17. $f(x) = g^x$ is a one-way function.

5.2.2 Variations of DLA

Definition 36 (Computational Diffie-Hellman Assumption,CDH)**.** Suppose G_q is a prime order multiplicative group, and $g \in G_q$ is a generator. Then for every PPT algorithm A,

$$\Pr[A(g^x, g^y) = g^{xy} : x, y \xleftarrow{\$} \mathbb{Z}_q] \leq negl(q)$$

Intuitively, this means that given g^x, g^y randomly chosen from G_q, it is hard to compute g^{xy} in probabilistic polynomial time.

Remark 16. CDH is a stronger assumption than DLA, since we are assuming something extra than in DLA[5]. Note that solving the Discrete Log problem means we can also solve the CDH problem: given g^x, g^y, just compute y by DLA and we can compute $g^{xy} = (g^x)^y$.

Definition 37 (Decisional Diffie-Hellman Assumption)**.** Suppose G_q is a prime order multiplicative group, and $g \in G_q$ is a generator. Then for every PPT algorithm A, the following two distributions are computationally indistinguishable:

$$\{(g^{xy}, g^x, g^y) : x, y \xleftarrow{\$} \mathbb{Z}_q\} =_c \{(g^r, g^x, g^y) : r, x, y \xleftarrow{\$} \mathbb{Z}_q\}$$

Intuitively, this means that given g^x, g^y randomly chosen from G_q, it is hard to distinguish g^{xy} from some random g^r in G_q in probabilistic polynomial time.

Remark 17. DDH is a stronger assumption than CDH: given g^x, g^y, g^r, we can compute g^{xy} and compare it with g^r.

[5]Note that in cryptography weaker assumptions are typically more desirable.

Key Agreement

In the last chapter, we introduced the decisional Diffie-Hellman assumption. Now we will see that this is indeed a very useful assumption and describe a key exchange protocol that is based on this assumption.

6.1 Key Exchange Definition

Key exchange is a way for cryptographic keys to be exchanged between two parties over a public channel. This is necessary in symmetric cryptographic schemes like one-time pad, where the sender and the receiver must agree on a key in order for the message to be enc- and later decrypted.

More formally, we consider two PPT parties A and B with private randomness r_A and r_B, respectively. Parties A and B interact with each other over multiple rounds. Let n denote the size of the randomness, and π the key exchange protocol. Then, $\pi(r_A, r_B)$ denotes the execution of the protocol π between parties A and B that use randomness r_A and r_B. The protocol execution results in a transcript τ, consisting of the sequence of messages exchanged between A and B. The view of the party A in the protocol is (r_A, τ), and view of the party B is (r_B, τ). In the end, party A outputs $k_A = \textit{compute key}(r_A, \tau)$, and party B outputs $k_B = \textit{compute key}(r_B, \tau)$, where *compute key* is a PPT algorithm.

Definition 38 (Key Exchange Protocol (KEP))**.** A *key exchange protocol* (KEP) must satisfy the following two properties:

1. **Correctness:** If both parties A and B are honest, k_A must be equal to k_B

   $$Pr_{r_A, r_B}[k_A = k_B] = 1$$

   Note that if $k_A = k_B$, then we can simply denote $k = k_A = k_B$.

2. **Security:** Consider a passive PPT adversary E (an eavesdropper) observing all messages transmitted over the network. The view of E is τ. Denote the length of the key k by κ. Then, the protocol is secure if the following holds:

   $$\{k, \tau\} =_c \{U_\kappa, \tau\},$$

   Alternatively, the protocol is secure if for all PPT E the following holds:

   $$|Pr[E(k, \tau) = 1 : r_A, r_B \xleftarrow{\$} U_n, \tau = \pi(r_A, r_B)] - Pr[E(U_\kappa, \tau) = 1 : r_A, r_B \xleftarrow{\$} U_n, \tau = \pi(r_A, r_B)]| \leq negl(\kappa).$$

Remark 18. An alternative definition where we require k to be indistinguishable from a uniform group element is also valid.

6.2 DIFFIE-HELLMAN KEY EXCHANGE

We are now ready to introduce a classic key exchange protocol designed by Diffie and Hellman in 1976.

6.3 DIFFIE-HELLMAN KEY EXCHANGE PROTOCOL

Let G_q be a group where the DDH assumption holds. Let $g \in G_q$ be a generator in this group. The protocol proceeds as follows.

- A picks $r_A = x \xleftarrow{\$} \mathbb{Z}_q^*$ as its own private randomness. Then, A sends $X = g^x$ to B.[6]
- B picks $r_B = y \xleftarrow{\$} \mathbb{Z}_q^*$ as its own private randomness. Then, B sends $Y = g^y$ to A.
- A outputs key $k_A = Y^x$, and B outputs key $k_B = X^y$.

Note that this protocol satisfies the definition for KEP:

- It satisfies correctness since $k_A = Y^x = g^{xy} = g^{yx} = X^y = k_B$.
- As for the security, note that the adversary knows g^x and g^y. If the adversary can then distinguish g^{xy} from a uniformly random g^r with a non-negligible probability, this directly breaks the DDH assumption.

6.3.1 Active vs. Passive Adversaries

So far, we have only considered passive attackers. In general, however, attackers may tamper with messages sent over networks. As such, we can categorize attackers into *passive* adversaries and *active* adversaries. Passive adversaries just observe the messages sent over the network. Active adversaries can modify or drop messages sent by other parties.

Now, one might ask: assuming two parties A and B have never met before, is it possible to design a protocol that protects against active attackers? The answer is no, and that any key exchange protocol will fail in such settings. The idea is that if A and B have never met, there exist no shared secrets, no notion of identity. As such, an attacker E could perform a *man-in-the-middle* attack by impersonating A and B: E can engage in a key exchange separately with A and with B (A will think that it is exchanging keys with B and vice versa), and then mediate the conversation between the two parties.

6.3.2 Key Management Using Trusted Third Party

In the 90s, people faced the problem of key management. A community with N members, generally require $O(N^2)$ keys for each member to securely communicate with another member. This problem was later addressed using a trusted central server. *Keberos* is one such protocol. Imagine two parties that wish to communicate with each other. In Kerberos, they will first each authenticate themselves with a trusted central key distribution server. Then, instead of trying to directly exchange keys with each other, the parties will rely on the central server to generate the key and send the key to them.

[6] Remember that g^x is computed using the group operation (rather than plain multiplication).

Public Key Encryption

As we have seen in the previous chapter, using a symmetric key has some issues: if a party wants to communicate with N other parties, it first needs to agree on a joint key with each communication partner, and then store all N shared keys. Public key cryptography allows to drop this requirement. Instead of a shared key generated for each conversion, each party has a *public key* that can be disseminated widely, and a *secret key* that must be kept private. To send messages to party A, other parties use A's public key. To decrypt the received messages, A uses its secret key. In the late 1980s and early 1990s, public key encryption was considered too slow and therefore remained a largely theoretical research topic. Now, given faster computers, public key encryption is widely used.

Like symmetric key encryption, public key encryption also consists three algorithms.

Definition 39 (Public Key Encryption (PKE))**.** A *public key encryption scheme* consists of three algorithms (Gen, Enc, Dec) defined below:

- $(pk, sk) \leftarrow Gen(\kappa)$. The PPT *key generation* algorithm Gen takes as input a security parameter κ and generates a public key pk and a secret key sk.
- $c \leftarrow Enc(pk, m)$. The PPT *encryption* algorithm Enc takes as input a public key pk and a message m, and outputs a ciphertext c.
- $m \leftarrow Dec(sk, c)$. The deterministic polynomial time *decryption* algorithm Dec takes as input a secret key sk and a ciphertext c, and outputs a plaintext message m.

The public key encryption scheme must satisfy the following requirements:

- **Correctness** For all messages m the following holds:

$$Pr[Dec(sk, c) = m : (pk, sk) \leftarrow Gen(\kappa), c \leftarrow Enc(pk, m)] = 1$$

- **Security** For all pairs of messages (m_0, m_1) the following holds:

$$\{pk||Enc(pk, m_0) : (pk, sk) \leftarrow Gen(\kappa)\} =_c \{pk||Enc(pk, m_1) : (pk, sk) \leftarrow Gen(\kappa)\}$$

  Alternatively, the security holds if for all pairs of messages (m_0, m_1) and for all PPT adversaries A the following holds:

$$|Pr[A(pk, Enc(pk, m_b)) = b : (pk, sk) \leftarrow Gen(\kappa), b \xleftarrow{\$} \{0, 1\}] - \frac{1}{2}| \leq negl(n)$$

7.1 Stateless and Deterministic Public Key Encryption

As we have shown in previous chapters, in symmetric key encryption any deterministic and stateless protocol cannot be secure in a multi-message setting. The same applies for public key encryption, but in a stronger way. The reasoning is as follows:

- If a symmetric key encryption is deterministic, then an attacker can perform "replay attacks". Intuitively, an adversary may notice identical ciphertexts repeated transmitted over the network. Then, the adversary knows that the same message is being transmitted again (although it may not be able to compute the message).
- In public key encryption, an adversary can perform an even stronger attack. Imagine we have a deterministic public key encryption scheme, two messages m_0, m_1 and a ciphertext c which is either m_0 or m_1. Then, the attacker could simply run $Enc(p_k, m_0)$ and $Enc(p_k, m_1)$ to figure out what the original message is.

Thus, we have established that deterministic PKE is unsafe. That being said, there do exist interesting applications of deterministic PKE. Imagine you have a database containing sensitive entries such as patient medical records which, by law, cannot be uploaded to public storage systems unencrypted. So, you encrypt the entire database before uploading it to cloud. Your client, a hospital, wants to run queries on the database to answer questions like "how many patients have disease X". However, the search functionality is gone. Since the database is now encrypted, it is unclear how the hospital can proceed.

However, if the database was encrypted using deterministic PKE, conducting such search is indeed easy. Indeed, the hospital may encrypt the keyword "disease X" using the public key and perform simple string matching to count the number of occurrences of this disease. Note that a non-deterministic PKE scheme will not work in this scenario because the same key can generate different ciphertexts for the same message over multiple executions.

7.2 Multi-Message Public Key Encryption

As we did for symmetric key encryption, we now provide a formal definition for a multi-message secure public key encryption scheme.

Definition 40 (Multi-Message Public Key Encryption, security requirement)**.** For every choice of vectors $(m_0^1, m_0^2, \cdots, m_0^q)$ and $(m_1^1, m_1^2, \cdots, m_1^q)$, where $q = poly(n)$, the following holds:

$$\{pk||\{Enc(pk, m_0^i)\}_{i=1}^q : (pk, sk) \leftarrow Gen(n)\} =_c \{pk||\{Enc(pk, m_1^i)\}_{i=1}^q : (pk, sk) \leftarrow Gen(n)\}$$

We will now show that for public key encryption, our standard security notion already implies the multi-message security.

Theorem 18. Every one-time secure PKE (Definition 39) is also multi-time secure.

Proof. We use the hybrid argument. Given two message vectors $(m_0^1, m_0^2, \cdots, m_0^q)$ and $(m_1^1, m_1^2, \cdots, m_1^q)$, we construct hybrids $H_0, ..., H_q$ as follows: let H_0 be the set of encryptions of the messages $(m_0^1, m_0^2, \cdots, m_0^q)$. We slowly (one message at a time) change the set to encryptions of the messages in the second vector, and eventually arrive at H_q, the set of encryptions of the messages $(m_1^1, m_1^2, \cdots, m_1^q)$:

$$
\begin{aligned}
H_0 &= \{pk||Enc(pk, m_0^1), Enc(pk, m_0^2), \cdots, Enc(pk, m_0^q) : (pk, sk) \leftarrow Gen(n)\} \\
H_1 &= \{pk||Enc(pk, m_1^1), Enc(pk, m_0^2), \cdots, Enc(pk, m_0^q) : (pk, sk) \leftarrow Gen(n)\} \\
&\vdots \\
H_{k-1} &= \{pk||Enc(pk, m_1^1), \cdots, Enc(pk, m_1^{k-1}), Enc(pk, m_0^k), \cdots, Enc(pk, m_0^q) : (pk, sk) \leftarrow Gen(n)\} \\
H_k &= \{pk||Enc(pk, m_1^1), \cdots, Enc(pk, m_1^k), Enc(pk, m_0^{k+1}), \cdots, Enc(pk, m_0^q) : (pk, sk) \leftarrow Gen(n)\} \\
&\vdots \\
H_q &= \{pk||Enc(pk, m_1^1), Enc(pk, m_1^2), \cdots, Enc(pk, m_1^q) : (pk, sk) \leftarrow Gen(n)\}
\end{aligned}
$$

First, we prove the following claim about any two consecutive hybrids:

Claim 7. If a PKE is one-time secure, then for all $k \in \{1, 2, \cdots, q\}$, $H_{k-1} =_c H_k$.

Proof. Assume for the sake of contradiction that there exists some $k \in \{1, 2, \cdots, q\}$ such that H_{k-1} is not computationally indistinguishable from H_k. Then, by the definition of computational indistinguishability, there exists a PPT adversary A who can distinguish H_{k-1} from H_k with a non-negligible probability. Using A, we can construct a PPT adversary B that distinguishes between encryptions of some two messages (m_0, m_1) with a non-negligible probability:

- B receives a challenge public key pk.
- B runs A, sends it pk, and receives two sets of messages $(m_0^1, m_0^2, \cdots, m_0^q)$ and $(m_1^1, m_1^2, \cdots, m_1^q)$.
- B sends the k-th messages (m_0^k, m_1^k) to its challenger and receives a challenge c.
- B constructs a vector S of encrypted messages by encrypting the first $k-1$ elements from the first set of messages, and the last $q-k$ elements from the second set of messages. The k-th element is the challenge c. Formally,

$$S = \{Enc(pk, m_0^1), \cdots, Enc(pk, m_0^{k-1}), c, Enc(pk, m_1^{k+1}), \cdots, Enc(pk, m_1^q)\}$$

- Then, B gives $pk||S$ as input to A.
- If A outputs that S is from set b, then B outputs b.

Note that the only element that differentiates H_{k-1} from H_k is the kth element. Thus, if the challenge ciphertext is the encryption of m_0^k, then $pk||S$ is from H_{k-1}. Otherwise, $pk||S$ is from H_k. Thus, B succeeds with the same probability as A. Note that B is PPT because A is PPT, and the only extra work B does compared to A, is constructing vector S, which is of polynomial length. Since the existence of attacker B violates the security requirement of PKE, we reached a contradiction. Note that the difference from SKE is that B was able to encrypt messages on its own. □

Given Claim 1, we can apply the Hybrid Lemma to get

$$\text{PKE is one-time secure} \implies H_0 =_c H_q$$

Since the two hybrids are indistinguishable for any choice of message vectors $(m_0^1, m_0^2, \cdots, m_0^q)$ and $(m_1^1, m_1^2, \cdots, m_1^q)$, we have shown that if a PKE is single-message secure, then it is also multi-message secure. □

7.3 ELGAMAL ENCRYPTION SCHEME

We now introduce the ElGamal public key encryption scheme, which is closely related to the Diffie-Hellman key exchange protocol introduced in the last chapter. We first describe the idea at a high level:

Recall that in Diffie-Hellman key exchange, given a generator g of a group G_q, two parties A and B each sample a random element r_A, r_B from $\mathbb{Z}_q^*$, and send g^{r_A} (respectively, g^{r_B}) to the other party. In the end, both parties share a common secret key $g^{r_A \cdot r_B}$ that can be used for encrypting messages between them.

Note that g^{r_A}, g^{r_B} are recorded in the transcript in clear. Suppose A needs to receive a message from B. Then, A could publish g^{r_A}, and B could send (g^{r_B}, c_B) to A, where c_B is a message encrypted using $g^{r_A \cdot r_B}$. A can first compute $g^{r_A \cdot r_B}$ in the same way as in Diffie-Hellman key exchange, and then decrypt c_B. This is the core idea of the ElGamal encryption scheme.

We now formally describe this encryption scheme, and prove its correctness and security.

7.3.1 Construction

Let G_q be a group over which the decisional Diffie-Hellman Assumption holds. Let $g \in G_q$ be a generator. The encryption scheme consists of the following three algorithms:

- $Gen(n)$: sample $x \xleftarrow{\$} \mathbb{Z}_q^*$ and return $(pk = g^x, sk = x)$.
- $Enc(pk, m)$: sample $r \xleftarrow{\$} \mathbb{Z}_q^*$ and return $(g^r, pk^r \cdot m)$.
- $Dec(sk, (c_1, c_2))$: return $c_2 c_1^{-sk}$.

Given $(pk = g^x, sk = x) \leftarrow Gen(n)$, and any message $m \in G_q$, the correctness of the scheme is shown in the following derivation:

$$\begin{aligned} Dec(sk, Enc(pk, m)) &= Dec(sk, (g^r, pk^r m)) \\ &= pk^r m \cdot g^{-r \cdot sk} \\ &= g^{x \cdot r} m \cdot g^{-r \cdot x} \\ &= m \end{aligned}$$

We now prove that the ElGamal scheme satisfies the public key encryption security definition.

Proof. We define the following hybrid distributions:

$$H_0 = \{g^x, g^r, g^{xr} \cdot m_0 : x, r \xleftarrow{\$} \mathbb{Z}_q^*\}$$
$$H_1 = \{g^x, g^r, g^z \cdot m_0 : x, r, z \xleftarrow{\$} \mathbb{Z}_q^*\}$$
$$H_2 = \{g^x, g^r, g^z : x, r, z \xleftarrow{\$} \mathbb{Z}_q^*\}$$
$$H_3 = \{g^x, g^r, g^z \cdot m_1 : x, r, z \xleftarrow{\$} \mathbb{Z}_q^*\}$$
$$H_4 = \{g^x, g^r, g^{xr} \cdot m_1 : x, r \xleftarrow{\$} \mathbb{Z}_q^*\}$$

By the DDH assumption we get $H_0 =_c H_1$, and $H_3 =_c H_4$. Since g is a generator of G_q and $m_0, m_1 \in G_q$, there exists $a, b \in \mathbb{Z}_q^*$ such that $g^a = m_0$ and $g^b = m_1$. That is: $g^z \cdot m_0 = g^{z+a}$, $g^z \cdot m_1 = g^{z+b}$. Clearly, the distribution of a random sample z from $\mathbb{Z}_q^*$, and the distribution of a random sample plus a fixed element $z+a$ and $z+b$ from $\mathbb{Z}_q^*$ are exactly the same. Thus, we get $H_1 = H_2 = H_3$. By the Hybrid Lemma, we conclude that $H_0 =_c H_4$.

By construction, given any two messages $m_0, m_1 \in G_q$, we have $(pk = g^x, sk = x) \leftarrow Gen(n)$, and $(c_1^0 = g^r, c_2^0 = pk^r \cdot m_0) \leftarrow Enc(pk, m_0)$ for elements $x, r \in \mathbb{Z}_q^*$ sampled uniformly at random. The distribution pk, c_1^0, c_2^0 is exactly H_0. Similarly, with $(pk = g^x, sk = x) \leftarrow Gen(n)$, and $(c_1^1 = g^r, c_2^1 = p_k^r \cdot m_1) \leftarrow Enc(pk, m_1)$, the distribution pk, c_1^1, c_2^1 is exactly H_4. We can now conclude that the ElGammal encryption scheme satisfies the public key encryption security. □

7.4 RSA ENCRYPTION SCHEME

RSA encryption scheme is another public key encryption scheme. While ElGamal relies on the Decisional Diffie-Hellman assumption, RSA relies on the RSA assumption. We first describe the RSA function and state the RSA assumption.

Definition 41 (RSA function)**.** Let P_n denote the set of prime numbers of length n. We construct the *RSA function* $f_{N,e}$ as follows:

- Sample $p, q \xleftarrow{\$} P_n$, and set $N = p \cdot q$.
- Choose $e \in \mathbb{Z}_{\phi(N)}^*$ (i.e. e is relatively prime to $\phi(N)$, and $1 \leq e \leq \phi(N)$).
- For $x \in \mathbb{Z}_N^*$, $f_{N,e}(x) = x^e \mod N$.

In the above definition, N is called the *RSA modulus*, and $\phi(N)$ is the Euler's phi function ($\phi(N) = |\mathbb{Z}_N^*| = (p-1)(q-1)$).

Looking a bit ahead, informally, the RSA assumption states that inverting an RSA function is hard. However, given some additional information it becomes easy. Note that given $\phi(N)$, since e is relatively prime to $\phi(N)$, the Extended Euclidean algorithm efficiently computes two coefficients $x, y \in \mathbb{Z}$ such that

$$ex + \phi(N)y = \gcd(e, \phi(N)) = 1$$

Taking the above equation in modulo $\phi(N)$, we get $ex \equiv 1 \mod \phi(N)$. That is, there always exists d such that $ed \equiv 1 \mod \phi(N)$. Now, given d as described above and

$f_{N,e}(m) = y = m^e \mod N$, one can easily invert it as follows:

$$\begin{aligned} y^d \mod N &= m^{ed} \mod N \\ &= m^{ed \mod \phi(N)} \mod N \\ &= m^1 \mod N = m \end{aligned}$$

The second equality follows from Euler's theorem, which states that for all a and b that are relatively prime holds $a^{\phi(b)} \equiv 1 \mod b$. Since $m \in Z_N^*$, m and N are indeed relatively prime.

We have shown that given d, it becomes easy to invert the RSA function. For this reason, d is called the *trapdoor* of the RSA function $f_{N,e}$.

7.4.1 RSA assumption

We now give the formal definition of the RSA assumption.

Definition 42 (RSA assumption)**.** For all PPT adversaries A,

$$\Pr[A(y,(e,N)) = x : p,q \xleftarrow{\$} P_n, N = pq, e \xleftarrow{\$} \mathbb{Z}^*_{\phi(N)}, x \xleftarrow{\$} \mathbb{Z}^*_N, y = x^e \mod N] \leq negl(n)$$

An alternative statement of the above definition is that *computing the e^{th} roots in Z_N^* is hard*.

RSA vs. Factoring Assumption

As we discussed above, given $\phi(N)$, an adversary can efficiently compute the trapdoor d, and then return y^d as x. One way to compute $\phi(N)$ is to factor N and then compute $\phi(N) = (p-1)(q-1)$. Thus, for the RSA assumption to hold the Factoring assumption must hold. However, it is not clear whether this is sufficient or whether there are ways to compute x that do not involve factorization. It is an open problem whether the RSA assumption is equivalent to the Factoring assumption.

RSA Encryption – First Attempt

Our ultimate goal is to design a public key encryption scheme using the RSA assumption. A naive attempt is to use (N,e) as the public key, d as the secret key, $Enc((N,e),m) = f_{N,e}(m)$ as the encryption algorithm, and $Dec(d,c) = c^d$ as the decryption algorithm. Although this naive scheme is clearly correct, it is not secure, since the encryption algorithm is deterministic. In order to build a secure public key encryption scheme using the RSA assumption, we need to introduce a new variant of one-way functions.

7.4.2 Trapdoor One-way Permutations (Trapdoor OWP)

Definition 43 (Trapdoor One-way permuation)**.** A collection of functions $\mathcal{F}_n = \{f_i : D_i \to R_i\}_{i \in I_n}$ is a family of *trapdoor one-way permutations* if the following conditions are satisfied:

1. (Function sampler) There exists a PPT algorithm $G(n)$ that takes as input the security parameter n and outputs (i,t) where $i \in I_n$, and t is the trapdoor associated with $f_i \in \mathcal{F}_n$.
2. (Input sampler) There exists a PPT algorithm S that takes i as input and outputs a uniformly random element from D_i.

3. (Polynomial time) Given i, $f_i(x)$ can be computed in polynomial time for all $i \in I_n$ and $x \in D_i$.
4. (Hard to invert) For all PPT adversaries A the following holds:

$$\Pr[y = f_i(x') : (i,t) \leftarrow G(n), x \xleftarrow{\$} D_i, y \leftarrow f_i(x), A(i,y) = x'] \leq negl(n)$$

5. (Inverts with trapdoor) There exists a PPT algorithm I such that the following holds:

$$\Pr[y = f_i(x') : (i,t) \leftarrow G(n), x \leftarrow D_i, y \leftarrow f_i(x), I(i,y,t) = x'] = 1$$

6. (Permutation) For all $i \in I_n$, f_i is a permutation.

Remark 19. Note that in the above definition, we consider a family of functions instead of just a single function. The reason is that given a single function f and a trapdoor t, even though finding the trapdoor information may require exponential time, the adversary may simply have it inbuilt inside its code. If everything is encrypted with the same trapdoor, the adversary is then able to recover every message that was ever encrypted. Using a family of functions prevents this problem.

7.4.3 RSA Implies Trapdoor OWP

We now show that the RSA assumption implies trapdoor one-way permutations.

We construct a trapdoor one-way permutation as follows:

- Function sampler: $G(n)$ samples two prime numbers p, q, and computes $N = pq$. Then, G samples a random exponent $e \xleftarrow{\$} \mathbb{Z}^*_{\phi(N)}$ and computes d such that $ed \equiv 1 \mod \phi(N)$ using the Extended Euclidean algorithm. Finally, G outputs $(i = (N,e), t = d)$.
- Input Sampler: note that $D_i = \mathbb{Z}^*_N$. Thus there exists a PPT algorithm to sample a random element from D_i.
- For each $i = (N,e)$, $f_i(x) = x^e \mod N$.

Note that condition 3 of Definition 43 is satisfied because computing $f_i(x)$ is clearly possible in PPT. Condition 4 follows directly from the RSA assumption. Condition 5 holds since we can easily construct I as follows: taking $((N,e), y, d)$ as input, I outputs $y^d = x^{ed} = x \mod N$. Finally, notice that because of the existence of the trapdoor f_i is one-to-one for all $i \in I_n$. Additionally, the input and output space are both Z^*_N. Thus, for all $i \in I_n$ our construction f_i is a permutation.

7.4.4 From Trapdoor OWP to Public Key Encryption

Given a family of trapdoor one-way permutations with the corresponding function sampler algorithm $G(\cdot)$, inverting algorithm I, and an input sampler D, we can construct a secure public key encryption scheme as follows:

- $Gen(n)$: runs the function sampler algorithm $(i,t) \leftarrow G(n)$, selects the hard-core predicate h_i for f_i, and returns $(pk = (i, f_i, h_i), sk = t)$.
- $Enc(pk, m)$: for a single bit message $m \in \{0,1\}$, samples $r \xleftarrow{\$} D_i$ and outputs $c = (f_i(r), h_i(r) \oplus m)$.

- $Dec(sk, c = (c_1, c_2))$: recovers r from c_1 using the trapdoor t and the inverting algorithm I, and then outputs $m = c_2 \oplus h_i(r)$.

Notice that this construction is indeed a secure public-key encryption scheme. Correctness follows from the properties of the trapdoor OWP. Now consider the security property. Note that $pk = (i, f_i, h_i)$. Thus, we need to show that for all (m_0, m_1) the following holds:

$$\{(i, f_i, h_i)||(f_i(r), h_i(r) \oplus m_0) : ((i, f_i, h_i), t) \leftarrow Gen(\kappa), r \xleftarrow{\$} D_i\}$$
$$=_c \{(i, f_i, h_i)||(f_i(r), h_i(r) \oplus m_1) : ((i, f_i, h_i), t) \leftarrow Gen(\kappa), r \xleftarrow{\$} D_i\}$$

Consider the following four hybrids:

$$H_0 := \{(i, f_i, h_i)||(f_i(r), h_i(r) \oplus m_0) : ((i, f_i, h_i), t) \leftarrow Gen(\kappa), r \xleftarrow{\$} D_i\}$$
$$H_1 := \{(i, f_i, h_i)||(f_i(r), b \oplus m_0) : ((i, f_i, h_i), t) \leftarrow Gen(\kappa), r \xleftarrow{\$} D_i, b \xleftarrow{\$} \{0,1\}\}$$
$$H_2 := \{(i, f_i, h_i)||(f_i(r), b \oplus m_1) : ((i, f_i, h_i), t) \leftarrow Gen(\kappa), r \xleftarrow{\$} D_i, b \xleftarrow{\$} \{0,1\}\}$$
$$H_3 := \{(i, f_i, h_i)||(f_i(r), h_i(r) \oplus m_1) : ((i, f_i, h_i), t) \leftarrow Gen(\kappa), r \xleftarrow{\$} D_i\}$$

Informally, by the security of the hard-core predicate, $h_i(r)$ is indistinguishable from $b \xleftarrow{\$} \{0,1\}$. Thus, $H_0 =_c H_1$. By perfect security of the one-time pad we get $H_1 = H_2$. Finally, again by the security requirement of the hard-core predicate, we get $H_2 =_c H_3$. By the Hybrid Lemma we get $H_0 =_c H_3$.

MAC and Hash Functions

In the previous chapters we mostly considered passive adversaries. Now we would like to consider a stronger adversary who might want to alter messages passed between the two parties. Perhaps surprisingly, with some schemes that we have proven secure in the previous chapters doing so is not that hard.

Say Alice is trying to send message to Bob using a one-time pad, and an adversary Eve is able to observe the network and intercept messages passed over this network. Assuming that Alice only sends “Attack" or “Defend" to Bob, Eve can always change the message from “Attack” to “Defend” and vice versa by

$$c' = c \oplus \texttt{"Attack"} \oplus \texttt{"Defend"}$$

Thus, after intercepting some ciphertext c from Alice, Eve changes it to c' and sends it to Bob. In this case, although Eve knows nothing about the shared key between Alice and Bob, she can still change the message.

Similarly, in the RSA context, say the message being sent is m. Eve can easily change the message to $2m$ by multiplying the ciphertext with $2^e \mod N$:

$$c' = m^e * 2^e \mod N = (2m)^e \mod N$$

Thus, encryption only guarantees hiding, but not non-tamperability. Obviously, we would like to make sure that we know the originator of the message that we received and that this message was not altered. In this chapter we will introduce so-called Message Authentication Codes (*MAC*s) that provide us with such guarantee.

8.1 Message Authentication Code

8.1.1 Definition and properties

Definition 44 (Message Authentication Code)**.** A *Message Authentication Code* (MAC) scheme consists of the following algorithms:

- $k \leftarrow Gen(n)$: The PPT algorithm Gen takes a security parameter n as input and outputs key k.
- $\sigma \leftarrow Mac(k, m)$: The PPT algorithm MAC takes key k and message m as inputs and output MAC σ.
- $Verify(k, m, \sigma)$: The deterministic PT algorithm $Verify$ takes key k, message m and MAC σ as inputs and outputs a bit b indicating whether the MAC is correct or not.

A MAC scheme must satisfy the following two properties:

- **Correctness:** for all messages m, the following holds:

$$Pr[Verify(k, m, \sigma) = 1 : k \leftarrow Gen(n), \sigma \leftarrow MAC(k, m)] = 1$$

- **Unforgability:** Probability of any PPT adversary A wining the forging game defined below is negligible.

Forging Game

The game is played between a challenger C and an adversary A as follows:

- **Sampling:** C samples $k \leftarrow Gen(n)$.
- **Learning:** A sends m_i to C, C replies with $\sigma_i = MAC(k, m_i)$. This step is repeated $l(n)$ times where l is polynomial in n.
- **Guessing:** A outputs (m, σ). A wins if for all i, $m \neq m_i$ and $Verify(k, m, \sigma) = 1$

Intuitively, a forging game allows an adversary to see polynomially many valid MACs on the messages of his choosing. In the end, the adversary needs to generate a new valid (message, MAC) pair.

8.1.2 Construction based on PRF

We will now discuss a MAC construction that is based on PRF:

- $Gen(n)$: Takes a security parameter n as input and outputs $k \leftarrow Gen_{PRF}(n)$.
- $MAC(k, m)$: Takes key k and message m as inputs and outputs $\sigma = PRF(k, m)$ as a MAC.
- $Verify(k, m, \sigma)$: Takes key k, message m and MAC σ as inputs and outputs 1 if $\sigma = PRF(k, m)$, and 0 otherwise.

Correctness clearly holds. We now prove the security of this scheme.

Proof. Suppose there exists an adversary A who wins the forging game with noticeable probability δ. Then, we can construct an adversary B who wins the PRF guessing game with some noticeable advantage as follows:

1. Run A.
2. On input m_i from A, B passes it to C. On reply c_i from C, B passes c_i to A.
3. Once A outputs (m, σ), B queries C with m and matches output with σ. If they match and m has not been queried before by A, then B guesses "PRF", else B outputs a random bit.

Note that if C uses the PRF, then the game that A is playing is the forging game and A wins in this game with some noticeable probability δ. Thus, in this case B wins the game with a noticeable probability as well. If C uses a truly random function $RF(\cdot)$, then we do not know how A behaves. However, the probability that A outputs a new (m, σ) such that $\sigma = RF(m)$ and m has not been queried before is negligible. Thus, in this case B outputs a right guess with an overwhelming probability. Therefore, B wins the PRF guessing game with probability $\frac{1}{2} + notice(n)$. □

8.2 COLLISION RESISTANT HASH FUNCTION

We now introduce another useful cryptographic primitive - Collision Resistant Hash Functions(CRHFs). Intuitively, hash functions allow one to compress a message and are widely used in multiple cryptographic schemes. First we introduce the so-called universal hash functions:

Definition 45. A set H is a family of *Universal Hash functions* if for every $x \neq y$ the following holds:

$$Pr[h(x) = h(y) : h \leftarrow H] \leq \frac{1}{|R|},$$

where R is the range of h.

Now we introduce the so-called Collision Resistant Hash Functions:

Definition 46 (Collision Resistant Hash Functions (CRHF))**.** A family of functions H = $\{h_i : D_i \rightarrow R_i\}_{i \in I}$ is a CRHF family if

- **Sampling:** There exists a PPT algorithm Gen which outputs $i \in I$ such that h_i is uniformly random element from H.
- **Easy to evaluate:** For all $i \in I, x \in D_i, h_i(x)$ can be computed in polynomial time.
- **Compressing:** For all $i \in I, |D_i| > |R_i|$.
- **Collision Resistance:** For all PPT adversaries A the following holds:

$$Pr[x \neq x' \text{ AND } h_i(x) = h_i(x') : i \leftarrow Gen(n), (x, x') \leftarrow A(i)] \leq negl(n)$$

Intuitively, a CRHF is compressing since $|D_i| > |R_i|$, so there must be several inputs map to the same output. However, due to the collision resistance property it must be hard for an adversary to find such a collision.

8.2.1 Facts

- **Fact 1:** No single h can be collision resistant.
  Every hash function will necessarily have collisions because of the compression property: since $|D_i| > |R_i|$, the pigeonhole principle guarantees that some inputs will hash to the same output. Collision resistance does not mean that no collisions exist; it only means that they are hard to find. With only one h there exists a PPT adversary who might have two inputs with the same outputs hard coded in its code description. When we have a family of functions H, we make it computationally hard for adversaries to hard-core collision pairs for every hash function.
- **Fact 2:** A "sufficiently" compressing CRHF family implies one-way functions.
  Proof Say for all $i \in I$ holds $|h_i(x)| \leq \frac{|x|}{2}$. Then we claim that h_i is also a one-way function. Suppose this is not true, and there exists a PPT adversary A who is able to invert h_i with noticeable probability. Then we can construct a PPT algorithm B that is able to find collisions with noticeable probability as follows:
  1. Sample x.
  2. Invoke A on $y = h_i(x)$ and get x'.
  3. If $x \neq x'$, output (x, x') as a collision claim.

Thus we have:

$$\begin{aligned}
PR[B \text{ finds a collision}] &= Pr[A \text{ successfully inverts } h_i \text{ AND } x \neq x'] \\
&= Pr[A \text{ successfully inverts } h_i] * Pr[x \neq x'] \\
&= Pr[A \text{ successfully inverts } h_i] * (1 - \frac{1}{\text{number of preimages of } y}) \\
&= notice(n) * (1 - \frac{1}{\text{number of preimages of } y}) \\
&= notice(n)
\end{aligned}$$

Note that the second step can be done is because the probability of A to successfully inverting h_i is independent from the probability that $x \neq x'$.

The last step can be done because we assume that the length of the CRHF output is at most half the length of the input. Say there is at most some negligible fraction $\frac{p}{|R_i|}$ of outputs that have two or more preimages. Denote the set of these outputs by S (its size is p). Then, because the length of the output is at most half the length of the input, the number of inputs that map to one of the elements in S is $1-negl$. Thus, we can construct an adversary B' finding the collisions that works just by sampling and outputting two input elements x_1, x_2. The probability that both of these elements map to some elements in S is $(1-negl)^2$. Then, the probability that both x_1 and x_2 map to the same element in S is $(1-negl)^2 \cdot \frac{1}{p}$. Since the fraction $\frac{p}{|R_i|}$ is negligible, $\frac{1}{p}$ is noticeable and thus $(1-negl)^2 \cdot \frac{1}{p}$ is noticeable as well. Thus, B' is able to find a collision with a noticeable probability, which is a contradiction to the collision-resistance property. Therefore, there exists a noticeable fraction of outputs that have two or more inputs and thus the last equation step holds.

Since a CRHF family has the collision resistance property, the probability for B to win must be negligible. Thus, we find the contradiction and h_i is one-way.

8.3 Construction based on Discrete Logarithm Assumption

We now provide a construction of a hash function that is based on the discrete logarithm assumption:

Given a group G such that $g \in G$ is a generator, and $|G| = q$ where q is a prime number, we define the hash function as follows:

- $Gen(n)$: Sample $r \leftarrow \mathbb{Z}_q^*$, and set $h = g^r$. Output $i = h$.
- $Evaluate(h, x)$: Parse x as $(x_0||x_1)$ (it must hold that $x_0, x_1 \in \mathbb{Z}_q$). Output $g^{x_0}h^{x_1} \in G$.

Note that in this construction h is sampled uniformly at random from G and $g^{x_0}h^{x_1}$ clearly can be computed in polynomial time. Since $g^{x_0}h^{x_1} \in G$, the size of the output is $log(p)$ where $p = 2q + 1$ (p is the Sophie-Germain prime). Since $x = x_0||x_1$, where $x_0, x_1 \in \mathbb{Z}_q$, the size of the input is $2log(q)$. Thus, the construction compresses the input to roughly half of its length.

Now we prove the collision resistance of this construction. Suppose there exists an adversary A who can output $x = (x_0, x_1)$ and $x' = (x_0', x_1')$ such that $(x_0, x_1) \neq (x_0', x_1')$ AND $g^{x_0}h^{x_1} = g^{x_0'}h^{x_1'}$ with some noticeable probability. Then, we can construct adversary B which is able to compute the discrete log of h with a noticeable probability as follows:

1. Take h as input.
2. Execute A on h to get $x = (x_0, x_1)$ and $x' = (x'_0, x'_1)$.
3. Compute the discrete log as follows: $r = \frac{x_0 - x'_0}{x'_1 - x_1} \pmod q$

Intuitively, after executing A, we get $x = (x_0, x_1)$ and $x' = (x'_0, x'_1)$ such that with noticeable probability $x \neq x'$ and the following holds:

$$g^{x_0} h^{x_1} = g^{x'_0} h^{x'_1}$$

$$g^{(x_0 + r x_1) \bmod q} = g^{(x'_0 + r x'_1) \bmod q}$$

$$(x_0 + r x_1) \bmod q = (x'_0 + r x'_1) \bmod q$$

Note that if $x_1 \neq x'_1$, $r = \frac{x_0 - x'_0}{x'_1 - x_1} \pmod q$ is not a division by zero. Thus, since A can find x and x' with noticeable probability, B can compute the discrete log of h with noticeable probability, which is a contradiction to the discrete logarithm assumption.

8.3.1 Better compression functions?

An interesting question is: Can we achieve better compression rates? The answer is yes, and we can provide a construction similar to the one discussed above:

- $Gen(n)$: Sample $r_1, ..., r_k \in \mathbb{Z}_q^*$, and set $h_m = g^{r_m}$ for $1 \leq m \leq k$
- $Evaluate((g, h_1, \cdots, h_k), x)$: Parse x as $(x_0, x_1, ..., x_k)$. Output $g^{x_0} h_1^{x_1} h_2^{x_2} ... h_k^{x_k} \in G$.

Now we get a hash function which input is very long but the output is rather short and belongs to G.

8.4 FURTHER THOUGHTS

There are generally three security goals for the authentication methods including Hash, MAC and digital signature:

- **Integrity:** Can the recipient be confident that the message has not been modified?
- **Authentication:** Can the recipient be confident that the message originates from the sender?
- **Non-repudiation:** If the recipient passes the message and the proof to a third party, can the third party be confident that the message originated from the sender?

It is also important to note that if the recipient and the sender share a secret key, the message authentication code method we discussed in this chapter can fulfil the first and second requirements. Unfortunately, it does not achieve non-repudiation. Digital signatures can fulfil all three requirements, but require asymmetric keys to implement. We will discuss how digital signatures work in the following chapter.

Digital Signatures

A Digital Signature scheme differs from the Message Authentication Code (MAC) introduced in the previous lecture by having a pair of public and secrete key, instead of a single (secrete) key. As a consequence, while only the holder of the secrete key can *generate* a valid signature, everyone can use the public key to *verify* a signature.

We now give the formal definition of a digital signature scheme:

Definition 47 (Digital Signature)**.** A *Digital Signature* scheme consists of the following three PPT algorithms:

- $(pk, sk) \leftarrow Gen(n)$. The PPT key generation algorithm Gen takes as input a security parameter n and outputs a public key pk and a secret key sk.
- $\sigma \leftarrow Sign(sk, m)$. The PPT signing algorithm $Sign$ takes as input a secret key sk and a message mk and outputs a signature σ.
- $b \leftarrow Verify(pk, m, \sigma)$. The PT algorithm $Verify$ takes as input a public key pk, a message mk, and a signature σ, and outputs "1" if the signature is valid, and "0" otherwise.

A Digital Signature scheme must be *correct* and *secure*:

Definition 48 (Correctness of Digital Signature)**.** For all messages m the following holds:

$$\Pr[Verify(pk, \sigma, m) = 1 : (pk, sk) \leftarrow Gen(n), \sigma \leftarrow Sign(sk, m)] = 1$$

i.e. The $Verify(\cdot)$ algorithm always outputs 1 if σ is a valid signature.

Definition 49 (Security of Digital Signature (Unforgeability))**.** For all PPT adversaries A, the chances of winning in the following forging game are negligible:

$$\Pr[\text{A wins}] \leq negl(n)$$

Forging Game The game is played between a challenger C and an adversary A as follows:

- (Keygen): C samples a key pair $(pk, sk) \leftarrow Gen(n)$, and sends pk to A.
- (Learning): for $1 \leq i \leq l = poly(n)$, A sends a message m_i to C and receives $\sigma_i = Sign(sk, m_i)$.
- (Guessing): A outputs (m, σ) and wins if $m \notin \{m_1, ..., m_l\}$, and $Verify(pk, m, \sigma) = 1$.

Note that in the "Learning" phase of the forging game, the adversary A can choose m_i adaptively, after seeing σ_j for all $j < i$.

9.1 ONE-TIME SIGNATURES

We first start with a weaker scheme, called the One-time Signatures shown by Diffie and Hellman [7] in their famous paper "New Directions in Cryptography". This scheme is secure if and only if the adversary A queries at most one message in the "Learning" phase of the forging game. This scheme only assumes the existence of One-way Functions (OWF).

Definition 50 (Construction of One-time Signatures)**.** Suppose f is a OWF, and a message m has length k. The one-time signature scheme is constructed by

- $Gen(n)$: samples k pairs of random strings $x_0^1, x_1^1, \ldots, x_0^k, x_1^k \xleftarrow{\$} \{0,1\}^n$, and computes $y_b^i = f(x_b^i)$ for all $b = \{0,1\}, i = 1, \ldots, k$. The algorithm outputs

$$\left(pk = \begin{pmatrix} y_0^1 = f(x_0^1) & y_0^2 = f(x_0^2) & \ldots & y_0^k = f(x_0^k) \\ y_1^1 = f(x_1^1) & y_1^2 = f(x_1^2) & \ldots & y_1^k = f(x_1^k) \end{pmatrix}, \quad sk = \begin{pmatrix} x_0^1 & x_0^2 & \ldots & x_0^k \\ x_1^1 & x_1^2 & \ldots & x_1^k \end{pmatrix}\right)$$

- $Sign(sk, m)$: selects the random strings in sk according to the bits of m, and outputs the selected strings as the signature σ:

$$\sigma = (x_{m[1]}^1, x_{m[2]}^2, \ldots, x_{m[k]}^k),$$

  where $m[i]$ denotes the i^{th} bit of m.

- $Verify(pk, m, \sigma)$: parse σ as $\sigma = (x^1, ..., x^k)$, and compare $y^i = f(x^i)$ with $y_{m[i]}^i$ for all $i = 1, \ldots, k$. If all checks pass, output 1. Otherwise, output 0.

Intuitively, the security of this one-time signature scheme relies on the fact that given only $y_b^i = f(x_b^i)$ in the public key, it's hard to invert f and find out x_b^i. If an adversary is to forge a signature for a new message, he needs to invert at least one such string, and thus break the security of OWF.

To prove the security of this scheme, we first need define the weaker (one-time) security notion that we want to prove.

Definition 51 (Security of One-time Signatures)**.** A challenger C and an adversary A play the following forging game:

- (Keygen): C samples a key pair $(pk, sk) \leftarrow Gen(n)$, and sends pk to A.
- (Learning): A sends a single message m to C and receives $\sigma = Sign(sk, m)$.
- (Guessing): A outputs (m', σ') and wins if $m \neq m'$, and $Verify(pk, m', \sigma') = 1$.

For all PPT adversaries A, the chance of winning the forging game is negligible:

$$\Pr[\text{A wins}] \leq negl(n)$$

And we now prove the following theorem.

Theorem 19. The above construction of $Gen(\cdot)$, $Sign(\cdot)$, and $Verify(\cdot)$ satisfies the One-time Signature security notion.

[7] they credit it to Leslie Lamport

Proof. Suppose there exists PPT adversary A that wins the forging game with non-negligible probability ϵ. We construct B that is able to invert f with non-negligible probability as follows:

- B takes input y, picks at random $b \xleftarrow{\$} \{0,1\}, j \xleftarrow{\$} \{1,\ldots,k\}$, and sets $y_b^j = y$. For all other $b' \in \{0,1\}, j' \in \{1,\ldots,k\}$, B samples $x_{b'}^{j'} \xleftarrow{\$} \{0,1\}^k$, and sets $y_{b'}^{j'} = f(x_{b'}^{b'})$.
- B uses y_b^jas well as all $y_{b'}^{j'}$ to construct pk, and sends pk to A.
- Upon receiving a query m from A: If $m[j] = b$, then abort. Otherwise, B creates a valid signature σ following the construction of $Sign(\cdot)$, and sends σ to A.
- Upon receiving m', σ' from A: If $m'[j] = m[j]$, then abort. Otherwise, B parses $\sigma' = (x^1,\ldots,x^k)$, and outputs x^j.

The intuition is that in a lucky event, B creates a valid signature without needing to find x_b^j such that $f(x_b^j) = y_b^j = y$, while A returns a valid signature that contains such an x_b^j. The probability of B aborting in step 3 is exactly $\frac{1}{2}$, as b is chosen at random. If A succeeds in producing a valid pair m', σ' such that $m' \neq m$, then m', m differs by at least 1 bit. The probability that $m'[j] \neq m[j]$ is at least $\frac{1}{k}$ as j is chosen at random. Note that these three events are clearly independent. The overall probability of B inverting f on y is

$$\Pr[m[j] \neq b] \Pr[A \text{ succeeds}] \Pr[m'[j] \neq m[j]] = \frac{\epsilon}{2k}$$

□

9.2 SIGNING LONGER MESSAGES

The first weakness of the One-time Signature scheme is that the key sizes are linear in the message sizes, and also linear to the security parameter. To be able to sign longer messages, we make the additional assumption that Collision Resistant Hash Functions (CRHF) exists.

Given a CRHF family $h_i : \{0,1\}^* \to \{0,1\}^k$, we construct the following improved algorithms $longSign(\cdot)$, and $longVerify(\cdot)$, that takes in arbitrary (polynomial in k) size messages.

- $Sign(sk, h(m)) \leftarrow longSign(sk, m, h)$
- $Verify(pk, h(m), \sigma) \leftarrow longVerify(pk, m, \sigma, h)$

Where $Sign(\cdot)$, $Verify(\cdot)$ are as defined in the One-time Signature scheme, and h is a collision resistant hash function. We omit a detailed proof of its security, and provide a brief sketch instead:

Proof. (sketch) Suppose a PPT algorithm A sees m, σ, pk, and outputs m', σ', where $m' \neq m$, and $longVerify(pk, m', \sigma', h) = 1$. Then one of the following happened:

- $h(m') = h(m)$, and we have found a collision.
- $h(m') \neq h(m)$, and we have broken the security of $Sign(\cdot)$.

□

9.3 SIGNING MULTIPLE MESSAGES

Here is a simple idea that allows to extend the One-time Signature scheme for signing multiple messages: as a part of each signature, we also sign the next public key we will use. However, in the original scheme, signing a public key of size k would require a longer key of size at least $2k$, assuming the outputs of f are just single bits. For the idea to make sense, we need to assume the existence of a CRHF family $h_i : \{0,1\}^* \to \{0,1\}^k$ as in the previous section. We also need to keep a counter, hence the modified scheme is a stateful one.

We construct the new scheme as follows:

- $(pk^1, sk^1) \leftarrow Gen(n)$. (the same as original.)
- $Sign(sk^i, m, h, i)$:
  First, generate a new pair of keys $(pk^{i+1}, sk^{i+1}) \leftarrow Gen(n)$, and compute $m'_i = h(m_i||p_k^{i+1})$. Next, generate σ_i using sk^i following the original construction (i.e., generate $2k$ random strings, and select k of them according to the bits of m'). Finally, output $\sigma'_i = (\sigma_1, \sigma_2, \ldots, \sigma_i, (m_1||p_k^2), \ldots, (m_i||p_k^{i+1}))$
- $Verify(pk^1, m_i, i, \sigma'_i)$:
  First, parse $\sigma'_i = (\sigma_1, \ldots, \sigma_i, (m_1||p_k^2), \ldots, (m_i||p_k^{i+1})$. Next, verify σ_j and $h(m_j||pk^{j+1})$ with pk^j for all $j = 1, \ldots, i$, following the original construction. If all checks pass, then output 1. Otherwise, output 0.

The security of this scheme similarly relies on the security of CRHF and that of the original scheme. We omit a detailed proof.

Note that the signature size is linear in the number of messages we have signed. Instead of signing the next pk in each signature, and using a chain-like structure to establish security, there exists improved schemes that sign the next 2 p_k's, and use a tree-like structure to establish security. In such schemes, the signature size is only logarithmic in the number of message we have signed.

9.4 AN RSA BASED SCHEME

The schemes introduced in the previous sections have the advantages of only assuming the existence of a generic OWF and CRHF, but unfortunately suffer from their large overheads. In practice, more efficient constructions are used. One such construction is bases on the RSA assumption. We first show an intuitive attempt to construct such a scheme using only the RSA assumption, that unfortunately is not secure. Then, we provide the secure construction.

Definition 52 (RSA Digital Signature, Insecure)**.** The three algorithms $Gen(\cdot)$, $Sign(\cdot)$, $Verify(\cdot)$ are constructed as follows.

- $Gen(n)$: Sample $p, q \xleftarrow{\$} P_n$, where P_n are primes of length n, compute $N = pq$, and compute $e, d \in Z^*_{\phi(N)}$ such that $ed \equiv 1 \mod \phi(N)$. Output $(pk = (e, N), sk = (d, N))$.
- $Sign(sk, m)$: Output $\sigma = m^d \mod N$.
- $Verify(pk, \sigma, m)$: Compare $\sigma^e \mod N$ with m. If they are equal, output 1. Otherwise, output 0.

As mentioned above, this scheme is insecure. A simple attack exploits the homomorphic nature of the RSA function:

Given $(m_1, \sigma_1 = m_1^d \mod N)$ and $(m_2, \sigma_2 = m_2^d \mod N)$, we can easily compute $(m = m_1 m_2, \sigma = (\sigma_1 \sigma_2) \mod N = (m_1 m_2)^d \mod N)$. Clearly, σ is a valid signature of m.

To get around this issue, we require another assumption: the existence of a so-called Random Oracle H, which behaves like an exponentially large random table. For each query x, $H(x)$ simply looks up the address x and returns the corresponding random string. Note that H is deterministic, and $H(x)$ always returns the same string for same x. Now, we show a secure construction based on the RSA assumption:

Definition 53 (RSA Digital Signature, Secure)**.** The three algorithms $Gen(\cdot)$, $Sign(\cdot)$, $Verify(\cdot)$ are constructed as follows.

- $Gen(n)$: Sample $p, q \xleftarrow{\$} P_n$, compute $N = pq$, and compute $e, d \in Z^*_{\phi(N)}$ such that $ed \equiv 1 \mod \phi(N)$. Output $(pk = (e, N), sk = (d, N))$.
- $Sign(sk, m, H)$: Output $\sigma = H(m)^d \mod N$.
- $Verify(pk, \sigma, m, H)$: Compare $\sigma^e \mod N$ with $H(m)$. If they are equal, output 1. Otherwise, output 0.

Although we omit a full proof of security of the above construction, we illustrate the use of H for getting around the homomorphic issue in the previous attack:

Given $(m_1, \sigma_1 = H(m_1)^d \mod N)$, and $(m_2, \sigma_2 = H(m_2)^d \mod N)$, $\sigma = \sigma_1 \sigma_2 = (H(m_1)H(m_2))^d$ is not necessarily equal to $H(m_1 m_2)^d$. Hence, in general, σ is not a valid signature of $m = m_1 m_2$. In fact, the probability of forging a message m' with σ (i.e. finding m' such that $H(m') = H(m_1)H(m_2)$) is negligible (s denotes the output length of H):

$$\forall m_1, m_2 \quad \Pr[H(m_3) = H(m_1)H(m_2)] = \frac{1}{2^s} = negl(s)$$

Remark 20. A true Random Oracle cannot exist in practice. However, assuming a Random Oracle enables many efficient and secure constructions. In practice, random oracles are approximated by carefully designed and tested hash functions, such as SHA-256. This is called the "Random Oracle Heuristic".

Another technicality is that $H(m)$ may not be in $\mathbb{Z}^*_N$. We can solve this issue by computing $H(m||i)$ for $i = 1, 2, ...$ until $H(m||i)$ is in $\mathbb{Z}^*_N$.

How to Share a Secret

Suppose Alice is the CEO of a small company and she would like to scale up the operations by renting some cloud storage. However, she is wary of storing all of the confidential data on a single cloud provider. She decides to distribute the data across multiple cloud providers. Suppose she decides to use 10 different cloud providers. Intuitively, she could store the first 10% of the data on the first cloud provider, the second 10% on the second cloud provider and so on. Unfortunately, this is not a very good idea because in this case every individual cloud provider can still recover 10% of the original data. Is there a way for Alice to store the data such that no information is leaked?

What Alice is looking for is a cryptographic primitive called a *secret sharing scheme*. Such a primitive would allow Alice to compute n shares from a secret s such that no information about s would be leaked if someone got access to any $n-1$ shares. At the same time, the secret s can be recovered if someone has access to *all* the n shares. We will now formally define this primitive.

Definition 54 (n-of-n secret sharing)**.** An n-of-n *secret sharing scheme* consists of the following two algorithms:

1. $(s_1, s_2, \ldots, s_n) \leftarrow \mathsf{Share}(s)$: The PPT sharing algorithm takes as input a secret s, and outputs n shares of the secret.
2. $s \leftarrow \mathsf{Reconstruct}(s_1, s_2, \ldots, s_n)$: The PPT reconstruction algorithm takes as input n secret shares, and reconstructs the corresponding secret.

Every n-of-n secret sharing scheme must satisfy the following two properties:

1. **Correctness:** For all secrets s, the following holds:

$$\Pr[\mathsf{Reconstruct}(s_1, \ldots s_n) = s : (s_1, \ldots, s_n) \leftarrow \mathsf{Share}(s)] = 1$$

   In other words, if the shares are computed using the Share algorithm, then the Reconstruct algorithm must be able to recover the original secret with probability one.

2. **Security:** For all secrets s, s', and for all sets I such that $I \subseteq \{1, \ldots, n\}$ and $|I| \leq n-1$, the following holds:

$$\{(s_i)_{i \in I} : (s_1, \ldots, s_n) \leftarrow \mathsf{Share}(s)\} \approx_c \{(s_i)_{i \in I} : (s_1, \ldots, s_n) \leftarrow \mathsf{Share}(s')\}$$

   Informally, this means that if an adversary has only up to $n-1$ shares, then he has no information about the secret.

10.1 CONSTRUCTION OF A SECRET SHARING SCHEME

We will now construct an n-of-n secret sharing scheme. We define the scheme $\langle \mathsf{Share}, \mathsf{Reconstruct} \rangle$ as follows.

- $\mathsf{Share}(s)$: For $i \in \{1, \ldots, n-1\}$, sample $s_i \leftarrow U_{|s|}$. Set $s_n \leftarrow s \oplus s_1 \oplus \ldots \oplus s_{n-1}$. Output $(s_1, \ldots, s_n)$.
- $\mathsf{Reconstruct}(s_1, \ldots, s_n)$: Output $s \leftarrow s_1 \oplus \ldots \oplus s_n$.

Correctness: follows from the fact that s_n is the XOR of the first $n-1$ shares and the secret s itself.

Security: Let s be a secret. Suppose $I \subseteq \{1, \ldots, n\}$ and $|I| = n-1$. Let $(S_i)_{i \in I}$ denote the random variable representing the shares which are the output of $\mathsf{Share}(s)$ restricted to the index set I in the random experiment. Now, we will calculate the probability that these shares are equal to some particular sequence of shares $(s_i)_{i \in I}$. The following probability calculations hold for *any* secret s and sequence of shares $(s_i)_{i \in I}$. We consider two cases:

Case 1: $n \notin I$. Since $|I| = n-1$, it means that $I = \{1, \ldots, n-1\}$.

$$
\begin{aligned}
& \Pr[(S_i)_{i \in I} = (s_i)_{i \in I} : S_1 \ldots S_n \leftarrow \mathsf{Share}(s)] \\
&= \Pr[S_1 = s_1, S_2 = s_2, \ldots, S_{n-1} = s_{n-1} : S_1 \ldots S_n \leftarrow \mathsf{Share}(s)] \\
&= \prod_{i=1}^{n-1} \Pr[S_i = s_i : S_1 \ldots S_n \leftarrow \mathsf{Share}(s)] && \text{(Since } S_1, \ldots, S_{n-1} \text{ are independent)} \\
&= \prod_{i=1}^{n-1} \frac{1}{2^{|s|}} && \text{(Since } s_1, \ldots, s_{n-1} \text{ are sampled uniformly at random)} \\
&= \frac{1}{2^{|s|(n-1)}}
\end{aligned}
$$

Case 2: $n \in I$. Since $|I| = n-1,\ \exists k\ s.t.\ k \notin I$. WLOG suppose $k = 1$. Then, $I = \{2, \ldots, n\}$.

$$
\begin{aligned}
&\Pr[(S_i)_{i\in I} = (s_i)_{i\in I} : S_1 \ldots S_n \leftarrow \mathsf{Share}(s)] \\
&= \Pr[S_2 = s_2, S_3 = s_3, \ldots, S_n = s_n : S_1 \ldots S_n \leftarrow \mathsf{Share}(s)] \\
&= \prod_{i=1}^{n-2} \frac{1}{2^{|s|}} \cdot \Pr[S_n = s_n : S_2 = s_2, \ldots, S_{n-1} = s_{n-1}, S_1 \ldots S_n \leftarrow \mathsf{Share}(s)] \\
&= \prod_{i=1}^{n-2} \frac{1}{2^{|s|}} \cdot \Pr[s \oplus S_1 \oplus \ldots \oplus S_{n-1} = s_n : S_2 = s_2, \ldots, S_{n-1} = s_{n-1}, S_1 \ldots S_n \leftarrow \mathsf{Share}(s)] \\
&= \prod_{i=1}^{n-2} \frac{1}{2^{|s|}} \cdot \Pr[S_1 = s_n \oplus s \oplus S_2 \oplus \ldots \oplus S_{n-1} : S_2 = s_2, \ldots, S_{n-1} = s_{n-1}, S_1 \ldots S_n \leftarrow \mathsf{Share}(s)] \\
&= \prod_{i=1}^{n-2} \frac{1}{2^{|s|}} \cdot \Pr[S_1 = s_n \oplus s \oplus S_2 \oplus \ldots \oplus S_{n-1} : S_1 \ldots S_n \leftarrow \mathsf{Share}(s)] \\
&\qquad\qquad \text{(Since } S_1 \text{ is independent of } S_2, \ldots, S_{n-1}\text{)} \\
&= \prod_{i=1}^{n-2} \frac{1}{2^{|s|}} \cdot \frac{1}{2^{|s|}} \\
&= \frac{1}{2^{|s|(n-1)}}
\end{aligned}
$$

Therefore, for all secrets s, s', and all index sets $I \subseteq \{1, \ldots, n\}$ such that $|I| = n-1$, the following holds:

$$\Pr[(S_i)_{i\in I} = (s_i)_{i\in I} : s_1 \ldots s_n \leftarrow \mathsf{Share}(s)] = \Pr[(S_i)_{i\in I} = (s_i)_{i\in I} : s_1 \ldots s_n \leftarrow \mathsf{Share}(s')]$$

Therefore,

$$\{(s_i)_{i\in I} : s_1, \ldots, s_n \leftarrow \mathsf{Share}(s)\} \equiv \{(s_i)_{i\in I} : s_1, \ldots, s_n \leftarrow \mathsf{Share}(s')\}$$

Note that these two distributions are equivalent and we did not use any unproven cryptographic assumptions in the construction. In other words, this secret sharing scheme is secure even if $\mathbb{P} = \mathbb{NP}$.

One drawback of this particular scheme is that the length of each share is equal to the length of the secret itself. Coming back to our motivating example, if Alice wanted to store a 1 GB file on different cloud providers, she would have to generate n different shares, each of size 1 GB and upload them to the cloud. Clearly, this is not very useful in practice. A natural question to ask is the following: Can we do better? Can the length of each share be strictly less than the size of the secret?

If we want the two distributions to be identical like in the construction above, then the answer is no. However, we can have such a scheme if we just want the distributions to be *computationally indistinguishable*. In the following, we give the outline of such a construction. Specifically, given a SKE (Gen, Enc, Dec), we show how to build a secret sharing scheme using the SKE as a black box:

- $\mathsf{Share}(s)$: Set $k \leftarrow Gen(\cdot)$, $c \leftarrow Enc(k, s)$. Then, using the sharing algorithm defined in Section 10.1, compute $(k_1, \ldots, k_n) \leftarrow \mathsf{Share}(k)$. For $i \in \{1, \ldots, n-1\}$, set $s_i \leftarrow k_i$. Set $s_n \leftarrow k_n || c$. Output $(s_1, \ldots, s_n)$.

- $\mathsf{Reconstruct}(s_1, \ldots, s_n)$: Interpret $s_n = k_n || c$. Then, using the reconstruction algorithm defined in Section 10.1, compute $k \leftarrow \mathsf{Reconstruct}(s_1, \ldots, s_{n-1}, k_n)$. Output $s \leftarrow Dec(k, c)$.

10.2 t-OF-n SECRET SHARING SCHEMES

The secret sharing schemes that we have seen so far all require that we have access to *all* the n shares of the secret if we want to reconstruct the secret. However, suppose that Alice would like to still be able to reconstruct the secret even if some of the cloud providers are down, and she can only get a subset of the shares. Is such a scheme possible? We will now formally define this problem.

Definition 55 (t-of-n secret sharing)**.** An *t-of-n secret sharing scheme* consists of the following two algorithms:

1. $(s_1, s_2, \ldots, s_n) \leftarrow \mathsf{Share}(s)$: The PPT sharing algorithm takes as input a secret s, and outputs n shares of the secret.
2. $s \leftarrow \mathsf{Reconstruct}(I, (s_i)_{i \in I})$: The PPT reconstruction algorithm takes as input $|I|$ secret shares, and reconstructs the corresponding secret if $|I| \geq t$. Otherwise, the algorithm returns $\perp$.

Every t-of-n secret sharing scheme must satisfy the following two properties:

1. **Correctness:** For all secrets s and all sets $I \subseteq \{1, \ldots n\}$ such that $|I| \geq t$, the following holds:
$$\Pr[\mathsf{Reconstruct}(I, (s_i)_{i \in I}) = s : (s_1, \ldots, s_n) \leftarrow \mathsf{Share}(s)] = 1$$
2. **Security:** For all secrets s, s', and for all sets I such that $I \subseteq \{1, \ldots, n\}$ and $|I| \leq t-1$, the following holds:
$$\{(s_i)_{i \in I} : s_1, \ldots, s_n \leftarrow \mathsf{Share}(s)\} \approx_c \{(s_i)_{i \in I} : s_1, \ldots, s_n \leftarrow \mathsf{Share}(s')\}$$
Informally, this means that if an adversary has any $t-1$ shares, then they have no information about the secret. On the other hand, any t shares can be used to reconstruct the secret.

How do we construct such a scheme? A natural attempt is to extend the n-of-n secret sharing scheme that we constructed earlier. The basic idea is to make sure that any t-subset of the n secrets can be combined to reconstruct s. One way to do this is to use a t-of-t secret sharing scheme as follows:

- $\mathsf{Share}(s)$: For all subsets $I = \{n_1, \ldots, n_t\} \subseteq \{1, \ldots, n\}$ of size t, using the t-of-t sharing algorithm defined in Section 10.1, compute $(s^I_{n_1}, \ldots s^I_{n_t}) \leftarrow \mathsf{Share}(s)$. For all $i \in \{1, \ldots, n\}$, set s_i to be the concatenation of all $s^{I'}_i$ s.t. $i \in I'$.

Clearly, any t-subset of shares can be used to reconstruct s because we used t-of-t secret sharing scheme for every subset. When t is a constant, this scheme takes $\mathsf{poly}(n)$ time. However, in the general case, this algorithm runs in exponential time. Since $\binom{n}{t} \geq (\frac{n}{t})^t$, when $t = n/2$, we can see that the running time of the algorithm is at least $\Omega(2^{n/2})$. Therefore, this is not a valid t-of-n secret sharing scheme.

10.2.1 Shamir's Secret Sharing

Adi Shamir constructed a t-of-n secret sharing scheme in 1979. This scheme relies on some properties of the polynomials, which we will now recall. We assume that everything is computed $\pmod p$ where p is prime.

1. **Define:** A degree d polynomial $f(x) = a_d x^d + a_{d-1}x^{d-1} + \ldots + a_0$ with $a_d \neq 0$ can be defined by choosing $d+1$ points. One possibility is for example to simply pick $f(1), f(2), \ldots, f(d+1) \in \{0, 1, \ldots p-1\}$.
2. **Interpolate:** Any $t = d+1$ points are enough to compute a degree d polynomial $f(x)$ for any x. Given $\{x_i, f(x_i)\}_{i=1}^{t}$, $f(x') = \ell_1(x')f(x_1) + \ell_2(x')f(x_2) + \ldots + \ell_t(x')f(x_t)$, where $\ell_j(x)$ is a *Lagrange coefficient* given by $\ell_j(x) = \prod_{i \neq j} \frac{x - x_i}{x_j - x_i}$.
3. **No Interpolation (Informal property):** Given any $t-1$ points, every other point is "perfectly hidden".

Shamir's secret sharing scheme uses the above properties of polynomials to construct a t-of-n sharing scheme.

- $\mathsf{Share}(s)$: Define a polynomial f of degree $d = t-1$ as follows: $f(0) = s$, and $f(i) \xleftarrow{\$} \mathbb{F}_p$ for $i \in \{1, \ldots t-1\}$. Now, for $i \in \{1, \ldots n\}$, set $s_i \leftarrow f(i)$.
- $\mathsf{Reconstruct}(I, (s_i)_{i \in I})$: If $|I| < t$, output $\perp$. Otherwise we have at least $t = d+1$ points and can use these points to reconstruct $f(0)$. Output $f(0) = \sum_{i \in I} \ell_i(0) s_i$, where $i \in I$.

Correctness: Correctness follows from the interpolation property of polynomials.

Security: Suppose $I \subseteq \{1, \ldots, n\}$ and $|I| = t-1$. Let $(S_i)_{i \in I}$ denote the random variable representing the shares which are the output of $\mathsf{Share}(s)$ restricted to the index set I in the random experiment. Now, we will calculate the probability that these shares are equal to some particular sequence of shares $(s_i)_{i \in I}$. The following probability calculations hold for *any* secret s and sequence of shares $(s_i)_{i \in I}$. In the following, $\mathsf{Interpolate}(\cdot)$ takes as input t shares (points on the polynomial) and a desired index j and outputs the value of the j-th share.

$$
\begin{aligned}
&\Pr[(S_i)_{i \in I} = (s_i)_{i \in I} \mid S_1 \ldots S_n \leftarrow \mathsf{Share}(s)] \\
&= \Pr[(S_i)_{i \in I} = (s_i)_{i \in I} \wedge (S_j)_{\forall j} = \mathsf{Interpolate}((i, S_i)_{i \in I}, j) \mid S_1 \ldots S_n \leftarrow \mathsf{Share}(s)] \\
&= \Pr[(S_i)_{i \in I} = (s_i)_{i \in I} \wedge (S_j)_{j<t} = \mathsf{Interpolate}((i, S_i)_{i \in I}, j) \wedge (S_j)_{j \geq t} = \mathsf{Interpolate}((i, S_i)_{i \in I}, j) \mid S_1 \ldots S_n \leftarrow \mathsf{Share}(s)] \\
&\leq \Pr[(S_j)_{j<t} = \mathsf{Interpolate}((i, S_i)_{i \in I}) \mid S_1 \ldots S_n \leftarrow \mathsf{Share}(s)] \qquad (\Pr[A \wedge B] \leq \Pr[A]) \\
&= \prod_{i=1}^{t-1} \Pr[S_i = \mathsf{Interpolate}((i, S_i)_{i \in I}) \mid S_1 \ldots S_n \leftarrow \mathsf{Share}(s)] \text{ (Since } S_1, \ldots, S_{t-1} \text{ are independent)} \\
&= \prod_{i=1}^{t-1} \frac{1}{p} \qquad \text{(Since } S_1, \ldots, S_{t-1} \text{ are sampled uniformly randomly)} \\
&= \frac{1}{p^{t-1}}
\end{aligned}
$$

Now, suppose that there is some sequence of shares $(s_i)_{i\in I}$ such that the above probability is strictly less than $\frac{1}{p^{t-1}}$. Then, we have the following:

$$\begin{aligned}
&\sum_{(s_i)_{i\in I}} \Pr[(S_i)_{i\in I} = (s_i)_{i\in I} \mid S_1 \dots S_n \leftarrow \mathsf{Share}(s)] \\
&< \sum_{(s_i)_{i\in I}} \frac{1}{p^{t-1}} \\
&= p^{t-1}\frac{1}{p^{t-1}} \\
&= 1
\end{aligned}$$

But the total probability must be equal to 1. Therefore, for all secrets s and for all shares $(s_i)_{i\in I}$, we have

$$\Pr[(S_i)_{i\in I} = (s_i)_{i\in I} \mid S_1 \dots S_n \leftarrow \mathsf{Share}(s)] = \frac{1}{p^{t-1}}$$

Therefore,

$$\{(s_i)_{i\in I} : s_1, \dots, s_n \leftarrow \mathsf{Share}(s)\} \equiv \{(s_i)_{i\in I} : s_1, \dots, s_n \leftarrow \mathsf{Share}(s')\}$$

Thus, the scheme is secure. Note that we again did not assume any unproven cryptographic assumptions. In particular, Shamir's secret sharing scheme works even if $\mathbb{P} = \mathbb{NP}$.

Remark 21. There also exist variations of t-of-n secret sharing schemes where each party may have a different weight. In this case, if the sum of the weights of the parties in a particular subset exceeds a threshold, then the secret can be recovered. Another variation is one in which some particular subsets are explicitly authorized. In this scenario, there is a monotone boolean function[8]that is true when the input subset is authorized to reconstruct or not.

10.3 THRESHOLD PKE (t-OF-n)

A secret sharing scheme is a one-time process. However, it has a limitation. Once the secret is reconstructed, it is not a secret anymore. We can generalize this concept to what is known as a **threshold PKE**. Such a scheme could be used in a situation where Alice decides to distribute the partial secret keys to a group of people in her company. Then, any group of t people would be able to decrypt the ciphertext without revealing their secret key shares. The same partial keys could be reused to decrypt other ciphertexts in the future.

Definition 56 (Threshold PKE)**.** A *threshold PKE* scheme consists of the following four PPT algorithms:

- $Gen(\kappa)$: the generation algorithm takes as input a security parameter κ, and outputs a public key pk and secret key shares $sk_1, \dots sk_n$.

[8]A monotone function f is a function such that if $A_1 \subseteq A_2$, then $f(A_1) \implies f(A_2)$.

- $Enc(pk, m)$: the encryption algorithm takes as input a public key pk and a message m, and outputs a ciphertext c.
- $PDec(sk_i, c)$: the partial decryption algorithm takes as input a secret key share sk_i and a ciphertext c, and outputs a share m_i.
- $Recover(I, (m_i)_{i\in I})$: the recovery algorithm takes as input an index set I and a sequence of shares $(m_i)_{i\in I}$. If $|I| < t$, output $\perp$, else output m.

Any threshold PKE scheme must satisfy the following two properties:

1. **Correctness:** for all messages m, and all sets $I \subseteq \{1, \dots n\}$ such that $|I| = t$, the following holds:

$$\Pr[Recover(I, (m_i)_{i\in I}) = m : (pk, sk_1, \dots sk_n) \leftarrow Gen, c = Enc(pk, m), m_i = PDec(sk_i, c)] = 1$$

2. **Security:** for all message m_0, m_1, all sets $I \subseteq \{1, \dots n\}$ such that $|I| \leq t - 1$, the following holds:

$$\{(c, pk, (sk_i)_{i\in I}) : (pk, sk_1, \dots sk_n) \leftarrow Gen, c = Enc(pk, m_0)\} \approx_c \{(c, pk, (sk_i)_{i\in I}) : (pk, sk_1, \dots sk_n) \leftarrow Gen, c = Enc(pk, m_1)\}$$

We can construct a t-of-n PKE using any regular PKE (Gen, Enc, Dec) and any t-of-n secret sharing scheme as a black box as follows:

- $Gen(\kappa)$: Generate $(pk_i, sk_i) \leftarrow Gen(\kappa)$ for $1 \leq i \leq n$. Set $pk = pk_1||pk_2|| \dots ||pk_n$. Output $(pk, sk_1, \dots, sk_n)$.
- $Enc(pk, m)$: Using the t-of-n sharing scheme, get $(s_1, \dots s_n) \leftarrow \mathsf{Share}(m)$. Set $c_i \leftarrow \mathsf{Enc}(pk_i, s_i)$ for all $i \in \{1, \dots n\}$. Output $c = c_1||c_2|| \dots ||c_n$.
- $PDec(sk_i, c)$: Interpret $c = t_1||t_2|| \dots ||t_n$. Output $m_i \leftarrow (Dec(sk_i, t_i))$.
- $Recover(I, (m_i)_{i\in I})$: If $|I| \leq t - 1$, output $\perp$. Otherwise, we have $|I| \geq t$. Output $m \leftarrow \mathsf{Reconstruct}(I, (m_i)_{i\in I})$ using the t-of-n sharing scheme.

10.3.1 Threshold PKE based on El-Gamal PKE

This construction is based on El-Gamal's PKE. Suppose we have a prime-order multiplicative group G with a generator g. Then, using El-Gamal PKE and Shamir's secret sharing scheme, we have the following threshold PKE:

- $Gen(\kappa)$: Sample $x \xleftarrow{\$} \mathbb{Z}_q^*$. Set $pk \leftarrow g^x$. Set $(sk_1 \dots sk_n) \leftarrow \mathsf{Share}(x)$ using Shamir secret sharing. Output $(pk, sk_1 \dots sk_n)$.
- $Enc(pk, m)$: Sample $r \xleftarrow{\$} \mathbb{Z}_q^*$. Output $c = (g^r, pk^r \cdot m)$.
- $PDec(sk_i, c)$: Parse c as (c_1, c_2). Output $m_i \leftarrow (c_1^{sk_i}, c_2)$.
- $Recover(I, (m_i)_{i\in A})$: Parse each m_i as (c_1^i, c_2) (note that all c_2 are equal). Compute $\prod_{i\in I}(c_1^i)^{l_i(0)} = g^{r\sum_{i\in I} sk_i l_i(0)} = g^{rx}$. Compute $y \leftarrow g^{-rx}$. Output $m \leftarrow c_2 y$.

One advantage of this scheme is that during encryption, Alice doesn't have to change anything and she can simply encrypt using El-Gamal PKE as if she were not using a threshold PKE. Then later at any point of time, she can compute the shares of the secret key and use this as a Threshold PKE.

The idea used in the recovery algorithm is known as *interpolation in the exponent*. e

Blockchains

11.1 Bitcoin History

In 2008, a person under the pseudonym Satoshi Nakamoto published a paper *Bitcoin: A Peer-to-Peer Electronic Cash System*. Bitcoin software was released in January 2009 and the mining of the Bitcoin cryptocurrentcy officially started. The genesis block included the "The Times" headline: *"Chancellor on brink of second bailout for banks"*. The article was about the state of the British financial system following the 2007–2008 financial crisis, and many believe that this is a hint to the purpose of Bitcoin: to create a more stable financial system. Satoshi Nakamoto vanished from the digital space shortly after releasing the code for Bitcoin, and it is unknown who this person (or possibly a group of people) is. The first known commercial transaction using bitcoin happened in 2010 - two pizzas were bought for 10000 bitcoin.

11.2 Blockchain as a public ledger

Blockchain can be viewed as a public ledger, where a block is one page of the ledger. Anyone can write to the ledger, and the ledger is append-only. Data recorded on the ledger is typically referred to as *transactions*. The process of finding the next block to append to the existing ledger is called *mining*.

11.2.1 Single Miner

For simplicity, assume that there is only one miner M, whose public- and secret key is (pk, sk). Let I_i be the data to be written on the next block (the next page on the public ledger). B_0 represents the genesis block. To find the next block $B_{i+1} = (h_{i+2}, I_{i+1}, pk, n_{i+1})$[9], miner M needs to do the following:

- Find n_{i+1} s.t $H(h_{i+1}, I_{i+1}, pk, n_{i+1}) = \underbrace{000\cdots0}_{k} XX\cdots X$, set $h_{i+2} = H(h_{i+1}, I_{i+1}, pk, n_{i+1})$.

Here n_i is referred to as nonce, and k is the parameter that tunes the difficulty of mining the block. H is a hash function (SHA-256 in Bitcoin). B_i is the i-th block. The data I_i that the miner wants to include in the new block contains newly generated transactions.

Intuitively, the mining process can be described as the miner trying different nonce values until the resulting hash satisfies the requirement that the first k bits of the hash output are 0. The mining requirement is also referred to as *Proof-of-Work (PoW)* requirement. With the random oracle assumption, Pr[M successful in a single attempt] is approximately $\frac{1}{2^k}$. Once a nonce that satisfies the mining requirement was found, we say the new block B_{i+1}

[9]Note that h_{i+1} denotes the hash of the block B_i, it is used in the computation of the block B_{i+1}.

was mined. Then, the miner can repeat the process to mine the next block B_{i+2} by finding n_{i+2} such that $H(h_{i+2}, I_{i+2}, pk, n_{i+2})$ is of the required form. Thus, each block is linked by hash to the previous block and this way a chain of blocks is formed.

11.2.2 Multiple miners

In the case of multiple miners, miners try to mine new blocks independently. If one miner is successful in mining the block, it broadcasts the newly found block B_{i+1} to the rest of the miners. Other miners then check that B_{i+1} is indeed valid. The check consists of the following:

- The hash included in B_{i+1} satisfies the Proof-of-Work requirement: it starts with k zeroes, and is a valid hash of h_{i+1}, transactions I_{i+1}, public key pk, and nonce n_{i+1}.
- The data I_{i+1} contains valid transactions.

If the new block is indeed valid, other miners restart to mine B_{i+2} based on B_{i+1}. However, with multiple miners, the following issues arise:

Issue 1: blockchain forks

Due to the network delay for block propagation, there may be two independent miners who found two valid blocks roughly at the same time, B_i and B_i'. This is called a blockchain fork. Bitcoin adopts the following rules to resolve a fork:

1. Longest chain is the right chain.
2. First received chain is the right chain (to break ties that may arise in rule 1).

When a fork happens as shown in Figure 1, some of the miners receive B_i first, while others receive B_i' first. This is totally fine. The first group of miners continues mining the next block based on B_i, the second group works on B_i'. Eventually one miner will find the next block. Suppose this miner is from the group working on B_i (shown in Figure 2). Then, all miners switch to mine on B_{i+1} as the branch represented by B_{i+1} is longer than the one represented by B_i' now.

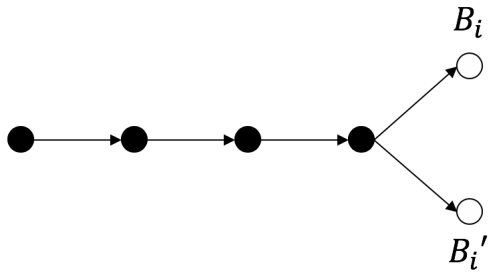


Figure 1: Blockchain fork

Issue 2: dishonest miners

As we will discuss later, miners currently receive a certain amount of Bitcoins as a reward for mining a new block. In the previous example, when all miners switch to mine new

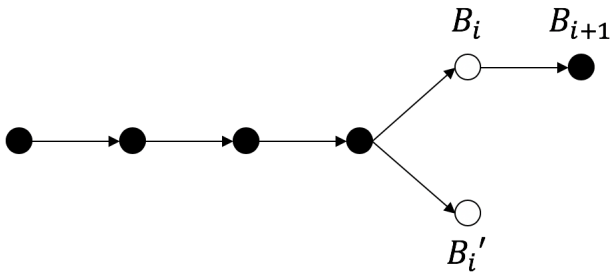


Figure 2: Resolving forks

blocks based on block B_{i+1}, the old block B_i' is discarded. Suppose the miner that mined B_i' does not want to loose the mining reward. Then this miner has the incentive to not follow the standard rules to resolve fork. Instead, that miner can try to keep mining on B_i'. If he is lucky enough and mines the next few blocks based on B_i' faster than other miners mine blocks on B_{i+1}, there is a chance that other miners will later switch back to the branch that contains the block B_i' if it becomes the longest branch. Once that happens, the blocks B_i, B_{i+1}, and their associated reward are discarded.

The probability of mining the next block is proportional to the computing power of the miner. In Bitcoin we assume that more than 50% of computing power is controlled by honest miners. Honest miners are those that strictly follow the prescribed rules for mining and validating blocks, while dishonest miners don't. With this fundamental 50% assumption, the probability that dishonest miners win a fork is less than that of the honest miners. However, there is still a non-zero probability that the dishonest miners can invalidate anything in the existing blockchain. A rule of thumb is to wait for 1 hour before a transaction is treated as final.

11.2.3 Infanticide

Some large stake holders in Bitcoin network do not want alternative cryptocurrencies to survive (the Bitcoin price would probably decrease if less people would use Bitcoin). For this reason, miners from the Bitcoin network sometimes perform a 50% attack on newer cryptocurrencies to destroy them. Typically, a large group of Bitcoin miners has a lot of computing power and can thus easily generate longer chains in the newer cryptocurrencies. People eventually loose confidence in the new cryptocurrency. This is called infanticide.

11.3 CRYPTOCURRENCY BUILT ON PUBLIC LEDGER

11.3.1 Creation of Bitcoin

Bitcoin is created whenever a new block is mined. Reward for mining a new block is halved every 4 years, and by 2040 no new Bitcoins will be rewarded. Then, the incentive for the miners will come solely from collecting transaction fees inside a block. This design favors the early adopters of Bitcoin as it is much harder to gain Bitcoins as time goes. This is regarded as one of the key reasons that make Bitcoin successful.

11.3.2 State and transactions

The reward of mining a new block is associated with the public key of the miner. Bitcoin system tracks how many Bitcoins a public key has. The state that records this information is called UTxOs, and it can be constructed by scanning the whole blockchain and keeping track of how many bitcoins each public key has received or sent. Bitcoins can be transferred from one public key to another. The transactions very roughly are of the following form:

$$Sign_{sk_i}(pk_i \ \ \underrightarrow{x \ coins} \ \ pk_j)$$

Essentially, it is a record showing that the public key pk_i is sending x coins to the public key pk_j. The record is signed by the secret key sk_i. Since only the holder of the secret key sk_i can generate this transaction, the coins associated with pk_i can only be spent by its true owner. Other miners check the validity of this transaction. The check includes the following:

1. Signature is valid.
2. pk_i has enough coins to send.

The first can be checked directly using the public key pk_i, the second can be checked against the Bitcoin system state, ie., the UTxOs. If both checks go through, the miners will add this transaction to their I_i statement and try to mine the next block using I_i. Intuitively, the miner who includes this transaction in the next block will get a transaction fee from pk_i. The holder of pk_i chooses the reward amount up front, and a higher amount typically results in a better chance of inclusion in the next block.

11.3.3 Weak anonymity

Bitcoin only provides weak anonymity. While people can perform transactions using public keys, and their real identities are hidden, the transaction graph itself as shown in Figure 3 is public and can be accessed by anyone. This graph can provide valuable information. For example, if you know that pk_2 is owned by Alice, then you can infer that pk_3 is controlled by Bob as you know that in real life Alice and Bob recently have transacted with each other. The inferred identity may not be perfectly accurate every time, but intuitively this is still information that you should not be able to gain. There exist alternative cryptocurrencies like Zcash, that are designed to hide the transaction graph. We will give more details on Zcash when we discuss zero-knowledge proofs.

11.3.4 Rate control

Recall that there exist a parameter k that can tune the difficulty of mining a new block. Bitcoin sets the mining difficulty such that on average one block is mined every 10 minutes. As the total computing power of Bitcoin miners increases, the difficulty parameter is also increased. More specifically, this parameter is automatically adjusted by code rather than any central authority.

11.3.5 Mining pools

It is quite difficult for individual miners to mine a block. It is even possible that a single miner never mines a block. To smooth out mining rewards and make them more predictable, individual miners can join large mining pools, where a set of miners share rewards with each other. A mining pool has a pool manager, with a public key pk_m. All

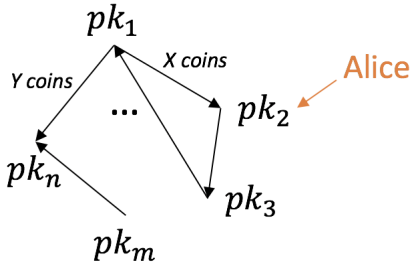


Figure 3: Transaction graph

miners in the pool mine blocks using this same key. Once a block is mined, the reward goes to the pool manager, and the manager then distributes the reward to its pool members according to a certain formula. The pool manager also makes sure that all its pool members are indeed contributing to the mining process. This is achieved by having all pool members submit the so-called *proof of partial work*:

$$H(h_i, I_i, pk_m, n) = \underbrace{000\cdots0}_{k-\epsilon} XX\cdots X$$

ϵ is set such that it is easy enough for each individual miner to successfully find a nonce that satisfies the partial requirement. Also the hash does not satisfy the Proof-of-Work requirement for mining the new block, it can only show to the pool manager that this miner is indeed contributing its computing power. Indeed, note that a partial proof of work is useless to a single miner - as long as $\epsilon > 0$, it does not satisfy the Proof-of-Work requirement that is needed to collect the reward (and even if $\epsilon = 0$, the public key used in the computation is the public key of the manager). The only purpose of such proof is to show to the manager how much effort you are putting in, and the reward that you get is proportional to this effort.

11.4 Applications built on top of Bitcoin and the Blockchain

There are many applications that can be built on top of Bitcoin and the Blockchain. In this section, we will enumerate a few examples of such applications:

- **Bitcoin scripting language** The Bitcoin scripting language enables custom transactions based on certain criteria being met (which could occur in the future). Examples of potential transactions enabled by the scripting language include:
  1. "A will transfer 1 BTC to anyone who publishes the value $f^{-1}(y)$, where f is a hard-to-invert function". Note that if no one is ever able to satisfy the condition by publishing the value $f^{-1}(y)$, then no one ever gets this 1 BTC and it is even lost by A. To prevent such scenarios, a time bound is put in the transactions. So, a more realistic transaction would be -"A will transfer 1 BTC to anyone who publishes the value $f^{-1}(y)$ in the next 7 days, where f is a hard-to-invert function".

2. "A will transfer 1 BTC to B if B publishes a signed statement that 'B will ship a TV to A'." Note that the transfer is automatic - if the condition is satisfied, the transfer can not be stopped by A.
3. "A will transfer 1 BTC to a script being run at a particular address".
   Typically, the script at this address will be run by a miner (who will be suitably rewarded for running the script). Possible use cases for such scripts include rent-collection, subscriptions, etc. These are examples of the so-called *smart contracts*.

- **Smart Assets** Smart Assets are either a virtual representation of a physical asset, or some virtual goods, such as an equity share. Examples of smart assets include:
  1. *Gold-backed cryptocurrency coins*
     The basic idea is that each coin represents a value of gold (e.g. 1 coin represents 1 gram of gold). The value of gold is stored by a trusted third-party custodian and can be traded with other coin holders. Such coins can be used in smart contracts, and the hope is that this approach would reduce volatility in coin value (since the coins are backed by gold, the value of which is relatively more stable).
  2. *Virtual representations of real-world goods*
     The basic idea is that real-world goods, such as shipping insurance, land records, etc. are represented as abstract assets (using virtual currency tokens) on the blockchain. These abstract assets act as a public record of ownership (using a mechanism similar to that used to represent Bitcoin ownership) and can be transferred between owners securely and safely.

11.5 Information Verification on the Blockchain - Merkle trees

Recall that each blockchain block contains I_i – the set of transactions. This set can get pretty big, so it seems advantageous to somehow shorten it so that not everyone has to download all transactions whenever only one particular transaction needs to be verified. The first naive idea would be to set I_i to be the hash all of the transactions. I_i would indeed become shorter. However, verification would still be difficult because we would still need to know all transactions in order to hash them and then verify the signature.

Merkle trees provide a solution to this problem. The key idea behind Merkle trees is to arrange messages as the leaf nodes of a tree, and then repeatedly hash pairs of nodes, until there is only one hash left at the root of the tree. Then, this root hash is signed.

Assume m_2 is the message whose signature needs to be verified. For verification, we need the signature, as well as the hashes of the siblings at every level. Referring to Figure 4, we need the signature σ, as well as the hashes of the siblings at every level, i.e. $H(m_1)$, $H(H(m_3), H(m_4))$. Using m_2 and the hashes of the siblings, we can generate the hash at the root, $H(H(H(m_1), H(m_2)), H(H(m_3), H(m_4)))$ which can then be verified against the signature σ using the public key.

Now, note the following:

- By signing the root, all messages are effectively signed.
- Size of the signature is of the order $log(n)$, where n is the number of messages.

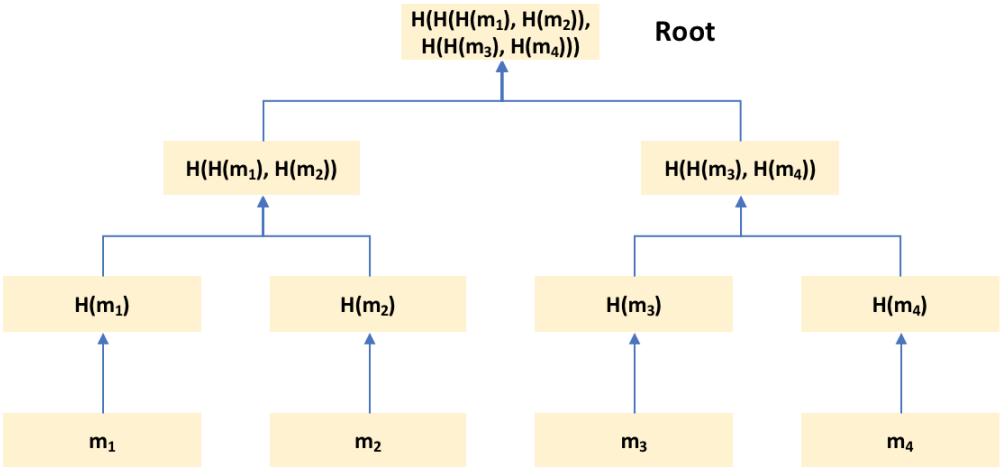


Figure 4: Merkle Trees. Image source: https://en.wikipedia.org/wiki/Merkle_tree

Note that this construction has a very nice feature: a transaction can be verified simply by downloading all blocks (which are now small since I_i's are small), and a single path in the Merkle tree.

Security. Intuitively, security relies on the Collision Resistance of the hash functions used as well as on the security of the signature scheme used: suppose the adversary generates a message m', say, in place of message m_2 as shown in the figure. Given this new message, and the original siblings at every level, the adversary must be able to generate the same hash at the root. Doing so will require the root hash generated with the new message to be the same as those generated using the old message, which is ruled out by the collision resistance property of the hash functions used.

Properties.

1. Able to sign messages as a block.
2. Enables light weight verification on low-memory devices such as mobile devices.

Usage in Bitcoin

Merkle trees are used for lightweight verification of transactions in Bitcoin. To verify a transactions in a block, one doesn't need to download all the transactions in the block. One can instead just download the hashes going down a particular branch of the Merkle tree and compute the hashes all the way to the root.

11.6 Limitations of Bitcoin

Bitcoins have several limitations including:

1. Transactions are not instantaneous.

2. Wastage of computational resources.
   Bitcoin uses Proof of Work as the mechanism for validation, which consumes a lot of resources. As a solution, we can use alternate energy-efficient validation mechanisms, such as:
   - *Proof of Stake*: Voting power is proportional to stake amount held by the miner (as opposed to computation power).
   - *Proof of Storage*: Voting power is proportional to storage.
3. Scalability
   Based on the current block sizes and mining rates, Bitcoin can process 7 transactions per second. In comparison, Visa can process > 5000 transactions per second.
4. Usability (Can be better)
   Bitcoin is completely decentralized and while this is great for network security, it presents a lot of usability issues. In the Fiat Money system, a bank takes care of any issues arising with one's bank account, making the system very usable. No such structure exists in the Bitcoin world, one is responsible for one's own wallet and consequentially one's secret key. No reversals exist as a feature of the network.
5. Anonymity
   The Bitcoin network makes all transactions between public keys visible. A user on the network participates in a transaction by using a public key, a cryptographic pseudonym. One can track users in a number of ways, from tracking payments to monitoring one's IP address related to transactions.

Moreover, here are some further considerations that could be problematic in some scenarios:

Government Intervention and Regulations. Governments remain a big threat to the future of the Bitcoin network as they have various tools at their disposal to halt the Bitcoin network. For eg. Due to a reservoir of funds at their disposal, governments can potentially launch a 51% attack against the Bitcoin network. Governments control the monetary policy of their respective country, and crytpocurrencies like Bitcoin are a direct threat to that. So, the adoption and regulation of Bitcoin in most countries is still up in the air.

Quantum Computers. Quantum computers pose a real threat to all Proof-of-Work blockchains like Ethereum and Bitcoin as they are extremely powerful and can be used to launch a 51% attack against blockchain networks. The threat is a while away though and a solution is being thought through.

11.7 TYPES OF FORKS IN BITCOIN

Sometimes, the code of the blockchain needs to be changed. This can be due to e.g., new features introduced to the blockchain, or fixing some issues. In such cases, the code need to be permanently changed and a single blockchain splits into multiple forks. We typically distinguish between the following two categories:

1. **Soft forks**
   Soft forks are backward compatible means of upgrading software on the nodes of the blockchain. Here, miners that did not upgrade are still able to verify transactions and validate new blocks. However, the blocks that such miners mine will be rejected.

Examples of soft forks [10] include:

- *BIP 66*: A soft fork on Bitcoin's signature validation.
- *P2SH*: A soft fork that enabled multi-signature addresses in Bitcoin's network.

2. **Hard forks**
   Hard forks are non-backward compatible means of upgrading software on nodes of the blockchain, where the original and forked chain run different software and rules, which are not compatible with each other. Blocks mined on one chain are not compatible with the other chain, and the miners that did not upgrade are not able to validate new transactions.

   Examples of hard forks [11] include:

   - *Ethereum's Byzantium*: Represents a planned multi-phase upgrade of Ethereum's blockchain base to deliver features such as better scalability and integration of private transactions.
   - *Monero hard fork*: Represents an upgrade to Monero network to implement a feature called Ring Confidential Transactions (RCT) to improve privacy and security.

11.8 Proof-of-Stake Blockchains

A proof of-work system is very expensive, since its mining process requires the user to solve hard-to-compute mathematical problems (such as inverting hash functions), and miners solve these problems using their own computer's computational power. This results in a huge amount of electricity being used. This is true especially in Bitcoin, where the mining difficulty only increases as more miners join the network and mine more Bitcoin. Currently, Bitcoin mining uses about $\frac{1}{400}$ of the world's electricity. Over a year, Bitcoin's power consumption amounts to roughly the same consumption as Switzerland.

To circumvent these high energy costs, an alternative system called *Proof-of-Stake* was introduced, first implemented in Peercoin in 2012. In proof-of-stake, the next miner is chosen through a pseudo-random process based on the potential miner's amount of coin (his "stake"). In contrast, proof-of-work chooses the next miner as the one who first solves the computationally hard problem. Since proof-of-stake chooses the next miner based on his amount of coins rather than computing power, more individuals are incentivized to join the network, leading to more decentralization.

11.8.1 Highlevel idea

A Proof-of-Stake protocol assigns a probability score to each miner and then chooses the miner with the highest score to mine the next block. This score is proportional to the product:

$$r_{pk} \cdot c_{pk}$$

where pk is the public key of the miner, r_{pk} is a randomly generated number for the miner with the public key pk (different schemes of generation will be defined), and c_{pk}

[10] https://masterthecrypto.com/guide-to-forks-hard-fork-soft-fork/
[11] https://masterthecrypto.com/guide-to-forks-hard-fork-soft-fork/

is the amount of coin that pk owns. The inclusion of the random number prevents new blocks from exclusively being mined by the wealthiest users.

In practice, instead of a single miner, a group of the miners that have the highest probability scores is chosen. This deals with the problem of offline miners.

11.8.2 Generating r_{pk}

Before we explain how the random parameter r_{pk} is generated, we consider a few straightforward, but insecure solutions.

1. Miner with pk chooses r_{pk}

Any adversary can choose a high r_{pk} and guarantee being chosen as the next miner. Thus, this scheme is insecure.

For the remaining schemes, let $H(\cdot)$ be a deterministic hash function which is used to generate the miners' random numbers, r_{pk}.

2. $r_{pk,i+1} = H(B_i, pk)$

Let B_i be the i-th and latest block which is not yet mined. B_0 is the genesis block. Suppose that the miner mining B_i has the public key pk_m and is dishonest. He can just keep computing $H(B_i, pk_m)$ for different values of B_i and searching for the one B_i that gives a high r_{pk} for the next block. Thus if you mine a block, you have an unfair advantage in the next block. Note that miners have some degree of freedom in deciding the block which they are mining since they can decide which transaction to include. Thus, this scheme is also insecure.

3. $r_{pk,i+1} = H(i, pk)$

Let i be the index of the latest block. Unlike the previous scheme, an adversary can just choose several public keys and compute their random numbers. Then he can transfer all his coins to whichever public key gives the highest r_{pk}. This is known as a prediction attack.

4. $r_{pk,i+1} = H(r_{pk,i}^{winner}, pk)$

Assume that an adversary knows all the public keys currently in the system as well as $r_{pk,i}^{winner}$, which is the random number of the latest block's winner. He can compute the random number for each public key for every future block. Then he will transfer funds to the public key that gives the maximum r_{pk}. This is known as a semi-predictable attack.

Finally, we consider the following proposal:

5. $r_{pk,i+1} = H(\mathtt{sign}(r_{pk,i}^{winner}), pk)$

Assume that when a miner mines B_i, he signs his winning random number, and broadcasts that signature. Now we distinguish the case where the miner is an honest party from the case where the miner is an adversarial party and provide an intuitive explanation as to why this scheme is secure.

If he is an honest party, then the broadcasted signature is unpredictable to an adversary. Also, the random number is not computable for any public key without knowing this signature.

Otherwise, if the miner is an adversarial party, then he can compute hash outputs of any public key. But it is also too late for the adversary to create new public keys and transfer

funds to the best one. Here, the random number is based on the sequence of miners who mined the previous blocks. However, to calculate the hash function, the signature must be generated, which can only be done by the miner who holds the corresponding signature key. Thus, intuitively, the random number is hard to predict and this scheme is secure.

11.8.3 Posterior Corruptions in Proof-of-Stake Blockchains

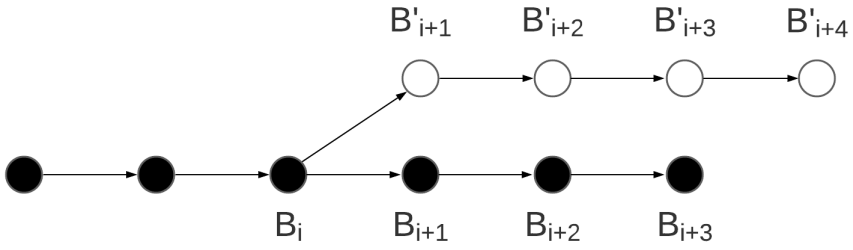


Figure 5: Posterior corruptions in PoS blockchains

One of the limitations of the above idea is that it suffers from posterior corruptions. For instance, say that in Figure 5 when block B_i is published, a set S has majority of stake. Next, suppose that the set S sells all of their stake in block B_{i+1} to someone to avail their services. After doing so, S becomes corrupt and creates a fork from node B_i and publishes a block B'_{i+1} where they don't sell all of their stake. Therefore, in the fork containing B'_{i+1}, S still has majority stake. The only problem is that this fork isn't the longest chain at this point of time. But, because S has the majority stake, they will get more opportunities to publish blocks on the forked chain. Thus, they can go on to publish enough blocks so that the forked chain becomes the longest. Once this happens, the forked chain becomes longest and accepted by everyone. Hence, S is able to recover the money they already spent. This is similar to double spending in Bitcoin, except that waiting for k more blocks to get published won't provide strong enough guarantee that a transaction will remain on the longest chain. This is a non-trivial problem to solve and has complex solutions beyond the scope of this course.

11.9 Existing Proposals to Solve POW Scalability Issues

The Bitcoin blockchain was designed such that new blocks are created at an average of every ten minutes. Typically, you would wait for a transaction to be about 6 blocks deep to make sure it really stays on the blockchain, meaning that about an hour is needed to spend the Bitcoin.

At first, it seems natural to simply increase the block size to solve the scalability issues. Unfortunately, if we increase the block size to hold more transactions, the new larger blocks take longer to propagate, allowing more miners to compute the new blocks by themselves. This leads to an increased number of forks and conflicts between blocks, resulting in the network being more vulnerable to attacks.

Similarly, if we decrease the (average) time after which a new block is mined, effectively increasing the rate of information being added to the blockchain, we will again increase the number of forks in the chain. This could make it easier for a malicious miner to attack the network with less than 50% hash power.

11.9.1 GHOST: Greedy Heaviest Observed Subtree

The GHOST protocol is an innovation first introduced by Yonatan Sompolinsky and Aviv Zohar in December 2013 to solve the scalability issue plaguing a lot of the proof of work based blockchains. The Ethereum blockchain implements a simplified version of GHOST which only goes down seven levels.

GHOST is a proof-of-work protocol similar to Bitcoin's, except that the main branch of the blockchain is chosen differently. Bitcoin follows a longest chain consensus rule, where the correct chain is the longest chain containing the most amount of total work. GHOST instead follows the path of the subtree with the most amount of total work, rather than a chained linked list. The advantage of GHOST over the usual Bitcoin protocol is that changing the block size doesn't compromise the security of the network.

GHOST is a recursive algorithm implemented as follows:

Let B_0 be the genesis block.

1. Look at all of B_i's subtrees, pick the heaviest computation subtree (the one with the most nodes).
2. If heaviest subtrees have equal weights, pick the one received first.

Problems with GHOST. Unfortunately, GHOST does not solve all issues. In some cases miners and mining pools can exploit the protocol to gain better rewards than their fair share:

1. Miners better connected to the network can gain rewards larger than their fair share.
2. Using the selfish mining strategy, small miners can deliberately add more forks to the network by keeping their blocks created private. This can enable them to create a private chain with a significant lead over the honest public chain.

11.9.2 Inclusive Blockchain Protocols

Motivation. The problem with present proof of work blockchains like Bitcoin is that the network security is compromised with high throughput. It's a trade-off between security and throughput. We need a protocol where the protocol security imposes no bottleneck to the throughput of the network. The network will be secure under all throughput parameters and the limitations of the network will be the infrastructure supporting the network. This will bring more/faster transactions on the main chain.

SPECTRE

Spectre was proposed by Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar in 2016 to solve the throughput vs security trade-off. The authors describe a DAG (Directed Acyclic Graph)-based permissionless, distributed ledger that aims to maintain high resilience to attackers while enabling high throughput through the network.

Bitcoin has a hash function that considers only the previous block to form the next one, enabling the chain. Spectre allows several past blocks in its hash function which

might have been created in parallel and be conflicting. The conflicts might include double-spending, where A tries to send money to B, and the same money to C, trying to bamboozle the network. However, this approach allows miners to create blocks concurrently and much more frequently as the nodes do not need to align their world view of the graph at the time of block creation.

Bitcoin's longest-chain rule can viewed as a voting mechanism. When conflicts in the chain occur, the nodes vote at the time of the next block's creation on which block they choose. This inhibits higher levels of throughput without disrupting the security of the network. SPECTRE employs a voting mechanism where the next block can safely vote on a conflicting previous block without making the miner align their world view at the time of the next block's creation. So, blocks are then voters and due to the design of SPECTRE, the majority vote becomes irreversible very quickly. Using the majority vote one can extract all accepted (non-conflicted) transactions easily.

The voting process to resolve conflicts is as follows:.

- For a pair of blocks a and b in conflict, determine whether a defeats b (represented as $a < b$), or b defeats a (represented as $b < a$). Here $a < b$, or a defeats b, means that a is before in the order of precedence than b.
- Release a set of accepted transactions to the network. Accepted transactions in block a are the ones which have defeated all conflicts and have all inputs accepted.

Properties of SPECTRE.

Consistency. Transactions are accepted if and only if the block wins the majority vote in a conflict and if all its inputs are accepted.

Safety. If a transaction is accepted by a node, then it is accepted by all nodes with a very high probability.

Weak Liveness. A transaction without any conflicts will be accepted by the main chain in the DAG very quickly.

Zero-Knowledge Proofs

12.1 Introduction

A *zero-knowledge protocol* is an interaction between two parties (say Peggy and Victor) that allows Peggy to convince Victor that some statement is true without revealing anything but the veracity of the statement. The critical part about this informal definition is the fact that Peggy does not reveal any extra knowledge: it would be trivial for Peggy to convince Victor that she has the knowledge by simply giving it to him. However, the point of a zero-knowledge protocol is that Victor is convinced that the statement is true but gains no additional information.

For example, suppose Peggy wishes to prove to Victor that there exists $(x_1, \ldots, x_6) \in \{0,1\}^6$ such that $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_4 \vee x_5 \vee \neg x_6) = 1$. This is an example of a 3-Satisfiability problem. Of course, she could just send the assignment of $(x_1, \ldots, x_6)$ to Victor but in this case, Victor learns more than whether there exists an assignment. All we want Victor to learn is the existence of an assignment. While initially this might seem impossible, zero-knowledge proofs were introduced in Goldwasser, Micali, and Rackoff's paper "The Knowledge Complexity of Interactive Proof-Systems". In the following, we will describe a zero-knowledge protocol for the graph isomorphism problem.

12.2 Formal Definition

Definition 57. Let $\mathcal{L}$ be a language, let x be a statement for which we are trying to prove $x \in \mathcal{L}$, and let w be a witness to $x \in \mathcal{L}$. A zero-knowledge protocol (P, V) for $\mathcal{L}$ consists of the following interactive probabilistic polynomial-time stateful algorithms P and V:

- $P(x, w, m_i^V)$: Outputs m_{i+1}^P where we set $m_0^V = \perp$.
- $V(x, m_i^P)$: Outputs m_i^V. At the end of the protocol, V outputs either `Accept` or `Reject`.

which satisfy the following properties:

1. (*Completeness*) Suppose w is a valid witness for statement x, the prover P, and the verifier V are honest. Then, V will always output `Accept`.
2. (*Soundness*) Suppose $x \notin \mathcal{L}$. For all PPT provers P^* that use such x, if V interacts with P^*, then V will output `Reject` with probability at least p. We call p the *soundness parameter*.
3. (*Zero-Knowledge*) Suppose $x \in \mathcal{L}$. For all PPT verifiers V^*, there exists an expected PPT simulator S such that $S(x, V^*)$ outputs a transcript $\tau_{sim} \approx_c \tau_{real}$ where τ_{real} is the transcript between an honest prover P and V^*.

Intuitively, P and V send messages between each other. The i^{th} message of P is m_i^P and the i^{th} message of V is m_i^V. After this interaction the verifier outputs `Accept` if it believes $x \in \mathcal{L}$, and `Reject` otherwise. The sequence of messages exchanged between the prover and verifier is called the *transcript* τ.

Completeness guarantees that the verifier believes the prover if $x \in \mathcal{L}$ and both parties act honestly. *Soundness* implies that if $x \notin \mathcal{L}$, then no dishonest prover can cause V to output `Accept` with probability greater than $1 - p$. *Zero-knowledge* implies that no dishonest verifier will learn anything from the interaction, besides the statement itself. Any knowledge in τ_{real} (i.e. any knowledge that V^* can learn from) must also be present in τ_{sim} since $\tau_{sim} \approx_c \tau_{real}$. Since S does not take w as an input, τ_{sim} contains no information on w. Note that we only guarantee that a simulator exists if $x \in \mathcal{L}$.

Given the zero-knowledge property, we might believe that the prover could simply run the simulator when interacting with the verifier. However, note the simulator has the code of the verifier and it is free to stop and rewind the verifier when creating its transcript. The prover cannot do this. For example, suppose Peggy wishes to convince Victor that she can make a fair coin land on heads 100 times in a row. With an interactive protocol, she might flip the coin, show Victor the result, and then repeat this 100 times. The simulator, on the other hand, can be thought of as a video recording. The video may show Peggy flipping 100 heads, but the video could have easily been edited. Peggy could have flipped the coin many times until she got 100 heads and then deleted the footage of all the flips that resulted in tails.

So far, we have been talking about the *computational zero-knowledge proofs*. If we require that the distribution of τ_{sim} and τ_{real} are equal, then we have *perfect zero-knowledge*. On the other hand, if we require that τ_{sim} and τ_{real} be statistically close, then we have *statistical zero-knowledge*.

12.3 ZERO-KNOWLEDGE GRAPH ISOMORPHISM

In this section we give the example of a zero-knowledge protocol for the graph isomorphism problem.

12.3.1 Graph Isomorphism Problem Definition

Two graphs, $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$ are isomorphic if and only if there exists a permutation π of V_1 such that $(i, j) \in E_0 \Leftrightarrow (\pi(i), \pi(j)) \in E_1$. Less formally, two graphs G_0 and G_1 are isomorphic if renaming the vertices in G_0 gives you G_1 and vice versa. We denote isomorphic graphs as $G_0 \approx G_1$, and non-isomorphic graphs as $G_0 \not\approx G_1$. The problem of deciding whether two graphs are isomorphic is not known to be in P nor is it known if it is NP-complete.

Before we go over a zero-knowledge protocol for graph isomorphism, we provide the following two facts:

1. If $G_0 \not\approx G_1$, then for any G such that $G \approx G_0, G \not\approx G_1$. Moreover, graph isomorphism is an equivalence relation, meaning that it is reflexive, symmetric and transitive.
2. Suppose $G_0 \approx G_1$. Let $\mathbb{P}$ be the group of permutations on these graphs. Then

$$\{\pi(G_0) : \pi \leftarrow \mathbb{P}\} = \{\pi(G_1) : \pi \leftarrow \mathbb{P}\},$$

12.3.2 Zero-Knowledge for Graph Isomorphism Protocol

The following is a ZK-protocol for the graph isomorphism problem. Both parties P and V share two graphs G_0 and G_1. P also has a witness π, which is a permutation of G_0 such that $\pi(G_0) = G_1$.

1. P chooses a random permutation $\sigma \in \mathbb{P}$. It sends $G = \sigma(G_0)$ to V.
2. V chooses a random bit b and sends this to P.
3. If $b = 0$, P sends $k = \sigma$ to V. Otherwise it sends $k = \sigma \circ \pi^{-1}$.
4. V checks if $k(G_b) = G$. If so, it outputs `Accept`. Otherwise, it outputs `Reject`.

Completeness. Assume that $G_0 \approx G_1$ and π is a valid witness for this. The point of our protocol is that V selects either G_0 or G_1 and P must prove that that graph is isomorphic to G. Since, by construction, $G \approx G_0$, we know that $G \approx G_1$. If $b = 0$, then V will receive $k = \sigma$ and can easily verify $k(G_b) = \sigma(G_0) = G$ (i.e. $G_0 \approx G$). If $b = 1$, then V will receive $k \circ \pi^{-1}$ and can easily verify $k(G_b) = \sigma \circ \pi^{-1}(G_1) = \sigma(G_0) = G$ (i.e. $G_1 \approx G$). Thus in either case, it outputs `Accept`.

Soundness. Suppose $G_0 \not\approx G_1$. For all graphs G, $G \not\approx G_0$ or $G \not\approx G_1$ (or G is not isomorphic to either). If this weren't true (i.e. $G \approx G_0$ and $G \approx G_1$) then we would conclude $G_0 \approx G_1$. In other words, there exists $b' \in \{0, 1\}$ such that $G \not\approx G_{b'}$. With probability 0.5, $b = b'$, where b is the random bit that V chose in Step 2 of the protocol. If $b' = b = 0$, then $G \not\approx G_0$. As specified by the protocol, P^* can send some permutation σ' to V. Note that if the two graphs are not isomorphic, there exists no permutation σ^* such that $G = \sigma^*(G_0)$. V will see that $\sigma'(G_b) = \sigma'(G_0) \neq G$ and output `Reject`. Similarly, if $b' = b = 1$, then $G \not\approx G_1$ and when P^* sends some permutation σ'' to V, V will see that $\sigma''(G_b) = \sigma''(G_1) \neq G$ and output `Reject`.

Thus, whenever $b' = b$, V outputs `Reject`. This occurs with probability 0.5 meaning that in general, V rejects with probability at least 0.5. Therefore, our protocol is sound with soundness parameter 0.5.

Zero-Knowledge. To show this property, for any verifier V^*, we construct a PPT simulator $S((G_0, G_1), V^*)$ that outputs a transcript that is computationally indistinguishable from the real transcript. Before we show the correct construction, we briefly discuss a naive incorrect attempt.

Incorrect Simulator

Recall that S only has access to G_0, G_1 and V^*, but not the witness π. The simulator works as follows:

1. S generates a random permutation ϕ and random bit b'. It sets $G = \phi(G_{b'})$.
2. S generates bit b at random.
3. If $b = b'$ send ϕ. Otherwise, erase the transcript and repeat from Step 1.
4. Output `Accept`. Return the transcript.

Note that in the actual protocol, if $b = 0$, then $k = \sigma$ defines an isomorphism between G and G_0. When $b = 1$ then $k = \sigma \circ \pi^{-1}$ defines an isomorphism beteween G and G_1. Similarly, in our simulator, when $b' = b = 0$, ϕ defines an isomorphism between G and G_0. When $b' = b = 1$, ϕ describes an isomorphism between G and G_1. Intuitively, Step 3 of our simulator is correct.

In the actual protocol σ is chosen randomly and in our simulator ϕ is chosen randomly. Thus, by property 2 above, $\sigma(G_0)$ is indistinguishable from $\phi(G_{b'})$ (note that for any graph G_0, $G_0 \approx G_0$). Thus, Step 1 in our simulator is correct.

However, Steps 2 and 4 pose a problem. For Step 2, an honest verifier chooses b at random which our simulator indeed simulates. However, our simulator must work for dishonest verifiers and a dishonest verifier can choose b however it wishes. So we need to modify our simulator to choose b with the same prbability that V^* does. Recall that S takes in V^* as an input. Therefore, we can simply run V^* to get b. Similarly, in Step 4, an honest verifier would always output `Accept` if $x \in \mathcal{L}$, but a dishonest verifier might not. Thus in Step 4, we feed ϕ to the V^* we are running which will return `Accept` or `Reject`. We will then output whatever this V^* returns.

Correct Simulator

1. S generates a random permutation ϕ and random bit b'. It sets $G = \phi(G_{b'})$.
2. S runs V^* and sends G to V^*. V^* returns b to S.
3. If $b = b'$ send ϕ to the V^* we are running. Otherwise, erase the transcript and repeat from Step 1.
4. V^* will return `Accept` or `Reject`. Output whichever V^* returns and return the transcript.

12.4 AMPLIFYING SOUNDNESS

The soundness parameter of a zero-knowledge protocol is essentially the probability that a dishonest prover can convince the verifier that $x \in \mathcal{L}$ when in fact it's not. A soundness parameter of 1 implies that the verifier always outputs `Reject` when $x \notin \mathcal{L}$. We would like the soundness of our protocol to be as close to 1 as possible. In this section, we explain how to amplify a protocol soundness.

Suppose a zero-knowledge protocol has soundness parameter p. If we perform k trials of the protocol, then the following must hold:

$$\Pr[V \text{ outputs } \texttt{Accept} \text{ all } k \text{ times } \mid x \notin \mathcal{L}] \leq (1-p)^k$$

Thus, to achieve a soundness of $1-\epsilon$, we can run the protocol $\log_{1-p} \epsilon$ times. If any of these trials output `Reject`, then we know $x \notin \mathcal{L}$ with probability 1.

The completeness of our new repeated protocol follows immediately from the completeness of the one-round protocol. Note that the protocol must be repeated in its entirety. Suppose that in the graph isomorphism zero-knowledge protocol, we kept the same σ between runs. Then in some run V could get $b = 0$ and find σ. In another run, it could get $b = 1$ and find $\sigma \circ \pi^{-1}$. Given these two permutations, it would be trivial for the verifier to calculate π, thus breaking the zero-knowledge property.

Additionally, suppose that our simulator for the repeated protocol was as following:

1.1 S generates a random permutation $\phi[1]$ and random bit $b'[1]$. It sets $G = \phi[1](G_{b'[1]})$

1.2 S runs V^* and sends G to V^*. V^* returns $b[1]$ to S.

1.3 If $b[1] = b'[1]$, send $\phi[1]$ to the V^* we are running. Otherwise, erase the transcript and repeat from Step 1.1.

2.1 S generates a random permutation $\phi[2]$ and random bit $b'[2]$. It sets $G = \phi[2](G_{b'[2]})$

2.2 S runs V^* and sends G to V^*. V^* returns $b[2]$ to S.

2.3 If $b[2] = b'[2]$, send $\phi[2]$ to the V^* we are running. Otherwise, erase the transcript and repeat from Step 1.1.

$\vdots$

k.1 S generates a random permutation $\phi[k]$ and random bit $b'[k]$. It sets $G = \phi[k](G_{b'[k]})$

k.2 S runs V^* and sends G to V^*. V^* returns $b[k]$ to S.

k.3 If $b[k] = b'[k]$, send $\phi[k]$ to the V^* we are running. Otherwise, erase the transcript and repeat from Step 1.1.

k+1.1 V^* will return `Accept` or `Reject`. Output whichever V^* returns and return the transcript.

Note that it would take exponential time for this simulator to finish. This is because the probability of $b[i] = b'[i]$ is 0.5 in every trial i. Thus, the probability of getting through all the steps is 2^{-k}. So instead of going all the way back to Step 1.1, whenever a trial fails, the simulator should simply go back to the beginning of that particular trial. For example, suppose on Step $k.3$ the simulator found $b \neq b'$. Then instead of going back to Step 1.1, it should simply go back to Step $k.1$.

There is a subtlety to this, however. The algorithm V^* is stateful. Therefore, if we restart a trial, V^* can still remember what occured during that failed trial. To mitigate this, we use an idea called *rewinding*. At the beginning of each trial, we save the state of V^*. If we need to restart that trial, we simply restore V^* to that saved state.

ZERO-KNOWLEDGE FOR GRAPH 3-COLORING

In the following, our goal will be to design a zero-knowledge protocol using the graph 3-coloring problem, an NP complete problem. Because all NP problems can be reduced to the graph 3-coloring problem, we obtain an entire class of zero-knowledge protocols for free. Before we provide the construction, we cover commitment schemes, an important cryptographic primitive which will be used in the zero-knowledge protocol.

12.5 COMMITMENT SCHEMES

A *commitment scheme* is a protocol that allows a committer to send a "locked" message, which the receiver cannot "unlock" by itself. In the future, the committer can provide a key to "unlock" the message, allowing the receiver to read the message while guaranteeing that the message that the receiver will obtain is exactly the message that was locked (the committer cannot lie about the locked message). Intuitively, it is similar to a safe: the committer puts a message in the safe, and hands the locked safe to a receiver without the corresponding key. At this point, the receiver cannot read the message but is guaranteed that the committer cannot change the contents of the message. The committer can then send the key to open the safe at any point in the future.

12.5.1 Definition

Formally, the protocol is separated in two stages:

1. **Commit phase**: Given a message m, the committer sends $c = Com(m, s)$ for some randomness s and for a PT algorithm Com to the receiver.
2. **Decommit phase**: The committer sends the pair (m, s), the receiver verifies that the pair is indeed correct by computing $Com(m, s)$, and comparing it with the previously received c. If they are equal, the receiver accepts, and otherwise, the receiver rejects.

A valid commitment scheme must satisfy the following two properties:

1. **Hiding**: $\forall(m_0, m_1)$,

$$\{Com(m_0, s) : s \leftarrow U\} =_c \{Com(m_1, s) : s \leftarrow U\}$$

2. **Binding**: $\forall(m_0, m_1)$ such that $m_0 \neq m_1$ and $\forall(s_0, s_1)$,

$$Com(m_0, s_0) \neq Com(m_1, s_1)$$

Intuitively, the hiding property establishes that the commitment does not leak any information about the locked message. The binding property then establishes that the receiver is guaranteed that the committer cannot send a fake (m', s') pair with $m \neq m'$ and still satisfy the equality check in the decommitment phase. Note: it may be possible to find (m, s') with $s \neq s'$.

Remark 22. The binding property defined above is also known as *perfect binding*. A weaker form of this is the computationally binding property, which merely establishes that it is very difficult to find a fake (m', s') pair with $m \neq m'$ that satisfies the decommitment requirement. In these lecture notes, we will only use commitment schemes with the perfectly binding property.

12.5.2 Examples

A natural attempt at constructing a commitment scheme might use encryption protocols where the committer sends an encrypted message during the commit phase and later sends the original message and the key in the decommit phase. However, it is not the case that all encryption algorithms are valid commitment schemes. For example, consider one-time pad: Suppose the committer sends $c = m \bigoplus s$. Then it is trivial for the committer to send a different m', s' pair such that $m \bigoplus s = m' \bigoplus s'$ and $m \neq m'$.

El-Gamal Commitment Scheme

The *El-Gamal commitment scheme*, which is very similar to the El-Gamal encryption scheme, relies on the DDH assumption. We are given a group G with order q and generator $g \in G$. The scheme proceeds as follows:

1. **Commit phase**: Given a message $m \in G$, the committer generates $s = (a, b)$ with $a, b \in Z_q$, and sends $c = (g^a, g^b, mg^{ab})$ to the receiver.
2. **Decommit phase**: The committer sends $m, (a, b)$, and the receiver checks if c matches (g^a, g^b, mg^{ab}).

Regarding the binding property, intuitively, note that the first two elements of c give us g^a, g^b, which means that a, b are fixed. Therefore, g^{ab} is fixed. Thus, m can be uniquely computed as $c(g^{ab})^{(-1)}$.

As for the hiding property, starting with the DDH assumption, we have:

$$\{g^a, g^b, g^{ab}\} \approx_c \{g^a, g^b, g^r\}$$

Thus,

$$\{g^a, g^b, mg^{ab}\} \approx_c \{g^a, g^b, mg^r\}$$

Since

$$\{g^a, g^b, mg^r\} = \{g^a, g^b, g^r\},$$

we get $\{g^a, g^b, mg^{ab}\} \approx_c \{g^a, g^b, g^r\}$. The same holds for any other m'. By the transitivity of computational indistinguishability, we are done.

Therefore, the proposed scheme is a valid commitment scheme.

Blum Commitment Scheme

The DDH assumption is considered to be a relatively strong assumption. It would be nice to have a commitment scheme that uses weaker assumptions. In particular, the *Blum commitment scheme* only uses 1-1 OWFs:

Suppose we are given a 1-1 OWF f. Then, the commitment scheme works as follows:

1. **Commit phase**: Given $m \in \{0, 1\}$, the committer samples s uniformly at random from the domain of f and sends $c = (f(s), m \bigoplus h(s))$ to the receiver, with h being a hardcore predicate of f.
2. **Decommit phase**: The committer sends (m, s) and the receiver verifies that $(f(s), m \bigoplus h(s))$ is indeed the same as c.

To show binding, we observe that given the first element of c, which is $f(s)$, s must be fixed because f is a 1-1 OWF. Thus, intuitively $h(s)$ is fixed because h is a deterministic function. Therefore, given $(f(s), m \bigoplus h(s))$, m is fixed.

To show hiding, we use the property of hardcore predicates that given $f(s)$, $h(s)$ is computationally indistinguishable from a uniformly sampled bit. Thus, $m \bigoplus h(s)$ is effectively a one-time pad.

Remark 23. The scheme can be easily extended for the multibit case: simply run this procedure for each bit. In more detail, for each bit m_i a new s_i is sampled, and then $c_i = (f(s_i), m_i \bigoplus h(s_i))$ is computed. The committer then sends all of c_i's to the receiver. During the decommitment phase, the committer sends all (m_i, s_i) pairs, and the receiver verifies that $(f(s_i), m_i \bigoplus h(s_i))$ is indeed the same as c_i for all indices i.

12.6 ZERO-KNOWLEDGE FOR GRAPH 3-COLORING

We now return to designing a zero-knowledge protocol for the graph 3-coloring problem.

12.6.1 Graph n-Coloring Problem

In general, the n-coloring problem is an NP-complete problem that asks whether all verticies of a graph can be colored using only n distinct colors, such that no two verticies that form an edge have the same colors. Although there are polynomial time algorithms to determine whether a graph is 1 and 2 colorable, there are no known algorithms to determine whether a graph is n colorable for $n \geq 3$.

In particular, a graph is 3-colorable if it can be colored using only 3 distinct colors. There are two important facts that we will use in subsequent sections.

1. Suppose we are given some graph and its 3-coloring. By permuting the colors, we obtain a new valid 3-coloring. Note that there are $3! = 6$ possible permutations.
2. Suppose we are given some graph that is not 3-colorable. In that case, any 3-coloring must contain at least one edge such that the two verticies that define the edge have the same colors.

12.6.2 Construction

In the zero-knowledge proof for the graph 3-coloring problem, the prover wants to prove that some graph G is 3-colorable (that is, there exists a witness w that 3-colors G). Of course, this has to be done without revealing any information about the coloring w; the prover merely wants to prove that some valid coloring exists. The protocol works as follows:

1. Given a coloring of G, the prover permutes the colors randomly to obtain a new (valid) coloring of G. For each vertex v_i in G, the prover sends $c_i = Com(\text{color}_i, s_i)$ to the verifier, where color_i is the (new) coloring of the vertex i and s_i is a random string.
2. Verifier selects a random edge (i, j) and then asks the prover to decommit the colors on v_i and v_j.
3. Upon receiving the decommitments from the prover, if the received decommitments are valid and the colors are different, the verifier accepts. Otherwise, the verifier rejects.

Now we show that this is a valid zero-knowledge protocol:

1. **Completeness**: This follows from the previously established fact 1 in that permuting colors in a valid 3-coloring of a graph will still yield a valid 3-coloring. Thus, regardless of the choice of (i, j), the colors of v_i and v_j will be different.
2. **Soundness**: From fact 2, we see that there is at least one edge where the coloring between two connected vertices will be the same if the graph is not 3-colorable (or if the prover provides a fake-coloring). Thus, because the verifier chooses an edge at random, and the prover is not able to change the committed colors due to the binding property of the commitment scheme, the verifier will reject with probability at least $\frac{1}{|E|}$.
3. **Zero Knowledge**: This is the most difficult part of the proof, and we will prove it below.

To show the zero-knowledge property, we design an EPPT simulator S that has G and the code of the verifier V^*. Let colors be $\{1, 2, 3\}$. Then, the simulator S works as follows:

1. S picks a random edge (i', j') and colors $k_{i'}, k_{j'} \in \{1, 2, 3\}$ at random such that $k_{i'} \neq k_{j'}$. S then generates commitments c_i for all vertices in G: if $i = i'$ or $i = j'$, S commits according to the protocol; else, S simply commits zero.
2. Invoke V^* and provide it with the commitments for all vertices in G.
3. Upon receiving an edge (i, j) from V^*, S does the following: If $(i, j) = (i', j')$, S reveals the colors by decommitting $c_{i'}$ and $c_{j'}$. Otherwise, restart, and goto step 1.

First, note that S is indeed an EPPT algorithm. This is because S picks i', j' at random, and so for any V^*, the probability that $(i, j) = (i', j')$ is roughly $\frac{1}{|E|}$. Therefore, S is expected to abort roughly $|E|$ times. The rest of the operations are in polynomial time, so the expected runtime is $|E|poly(n)$, which is still polynomial. Of course, this is assuming that V^* cannot purposefully select i, j such that $(i, j) \neq (i', j')$; we will briefly sketch a proof of this below.

Because of the hiding property of the commitment scheme, we are guaranteed that no PPT V^* can tell apart $c_{i'}$ and $c_{j'}$ from any of the other c_k. This means that V^* cannot distinguish i', j' from any other vertex just from c_i for all i, meaning it cannot query S for a particular i, j to forcibly abort S. Formally, if we are given a V^* such that

$$Pr[(i', j') = (i, j)] \geq \frac{1}{|E|} + notice(n)$$

it is possible to construct a PPT A that breaks the hiding property of the commitment scheme. Thus, S is EPPT. Now, we prove that $\tau_{sim} =_c \tau_{real}$ via a hybrid argument.

- H_0: In this hybrid, the algorithm S has a valid witness w and a code of V^*. S runs V^*, and in its interaction with V^*, S behaves like an honest prover: S commits to the colors of the vertices using (permutation of) the witness w, and opens commitments c_i, c_j upon receiving an edge (i, j) from V^*. In the end, S outputs its transcript of the interaction with V^*.
- H_1: In this hybrid, the algorithm S has a valid witness w and a code of V^*. S runs V^*, and in its interaction with V^*, S picks random indices i', j' and then commits honestly to the colors of the vertices using (permutation of) the witness w. When S receives an edge (i, j) from V^*, and it restarts (including restarting V^*) if $(i, j) \neq (i', j')$. Otherwise, S outputs its transcript of the (last) interaction with V^*.
- H_2: In this hybrid, S behaves the same as H_1, except that for each vertex i such that $i \neq i'$ and $i \neq j'$, S now commits zero instead of the actual vertex color.
- H_3: In this hybrid, instead of using a valid witness, S now samples two colors for i' and j' such that these two colors are different, and uses these colors in the commitments $c_{i'}, c_{j'}$ instead of the (permutation of the) actual vertex colors of vertices i and j. Note that S does not use the witness at all anymore.

First, note that $H_0 = \tau_{real}$ and $H_3 = \tau_{sim}$. Next, we see that $H_0 = H_1$ because i', j' are chosen at random and once (i', j') are equal to (i, j), the generated transcript is exactly the same as the one in H_0. $H_1 =_c H_2$ because of the hiding property of the commitment scheme. Finally, $H_2 =_c H_3$: note that in Step 1 of the protocol, we permute the colors. Thus, in H_3, the distribution of colors under the commitments $c_{i'}, c_{j'}$ does not change. Thus, $H_2 =_c H_3$, and therefore $H_0 =_c H_3$ and $\tau_{sim} =_c \tau_{real}$.

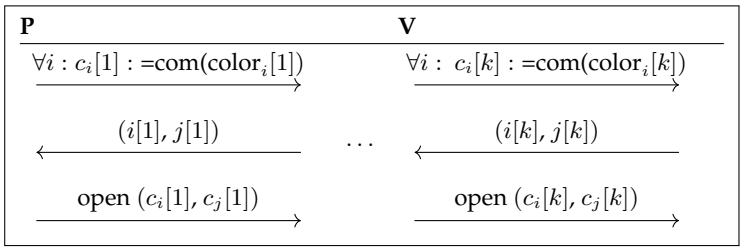


Figure 6: Protocol P_{cZK}

12.7 Amplifying Soundness - Part 2

Above, we constructed a zero-knowledge protocol for the graph 3-coloring problem with the soundness parameter $\frac{1}{|E|}$. However, what we actually want is to get a protocol with the soundness property close to 1. To reach this soundness, one option is to repeat the protocol we have multiple times sequentially. This way, after repeating the protocol k times, we achieve a soundness parameter $1 - (1 - \frac{1}{|E|})^k$. However, this improvement comes at a cost. Now, instead of a 3-round protocol, we actually have a $3k$-round protocol. For the expression $1 - (1 - \frac{1}{|E|})^k$ to get close to 1 in a graph with many edges, we would have to use a very high k. Thus, we end up with a not very efficient protocol with a very high number of rounds. So, the question is how we can change the protocol to make it more efficient.

Idea: use parallel repetition instead of the sequential repetition!

12.7.1 Candidate protocol for Graph 3-Coloring using parallel repetition

In the parallel repetition version of the protocol, instead of computing only one permutation in the beginning, the prover computes k permutations. In the first round, the prover commits to these k permutations. Next, the verifier sends k challenges to the prover. Finally, the prover opens the chosen k commitment pairs and the verifier accepts iff he would have accepted in each of the k individual cases. The protocol (we will call it P_{cZK} in the following) is pictured in Figure 6.

We will not discuss the Soundness and the Completeness properties of the protocol. Rather, we will directly consider the Zero Knowledge property. Interestingly, it is not trivial to see whether or not this protocol satisfies the Zero Knowledge property. In fact, it is still an open problem. In the following, we will try to adapt the proof of the ZK property for the sequential composition and see where this proof fails for the parallel composition.

Attempt to design a simulator.

1. For all t, the simulator makes a guess $(i'[t], j'[t])$ and commits to two random but different colors in $(c_{i'}[t], c_{j'}[t])$. In all other commitments, simulator puts 0.
2. For all t, the simulator gets $i[t]$ and $j[t]$ from V^*.
3. If for all t holds that $(i'[t], j'[t]) = (i[t], j[t])$, the simulator opens all selected commitments. Else, go to step 1.

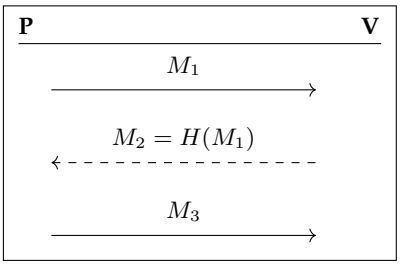


Figure 7: Protocol $P_{\text{ZK-FS}}$

Note that this is almost exactly the simulator designed for the sequential version, except that now we consider k instances of the problem in each step. The problem with this construction is the following: it is not a PPT construction. If there are k parallel repetitions, the probability that the simulator correctly guesses the edge in each of these repetitions is only $(\frac{1}{|E|})^k$. Thus, if k is big, this probability is very small and the simulator has to go to step 1 very often.

Now, it is possible that for some of the k instances, the simulator guessed the edge correctly. So, one might wonder whether the simulator could "keep" the correctly guessed edges and only change his guess where he did not guess correctly. Let us change Step 3 of our protocol as follows:

3. If for all t it holds that $(i'[t], j'[t]) = (i[t], j[t])$, the simulator opens all selected commitments. Else, go to step 1 and obtain a fresh guess for instances where the simulator was wrong.

The hope with this idea is that after each iteration, the number of unsuccessful guesses will be reduced and thus eventually become zero. However, this logic is incorrect: the verifier might be computing his challenge depending on how the prover's *entire* message looks like and thus there is no guarantee that all the "correct" guesses of the prover will remain correct after correcting one "incorrect" guess. For example, changing the first message in the t-th iteration could result in verifier message changing in $t' \neq t$-th session. Thus, this approach also does not work.

12.7.2 Fiat-Shamir-Heuristic

While it is not known whether or not the protocol we defined above is secure, this protocol has been known for many decades now and still no one came up with an attack for it. The general assumption is that this construction is indeed secure, and it is exactly what is used in the Fiat-Shamir-Heuristic we will now introduce.

Observe that P_{cZK} is a public coin protocol. This means that in the second round the verifier is essentially simply sampling and sending a random string to the prover.

Now, if the verifier is just sending some random string, we could instead just use an "appropriate" hash function H to do the verifier's job for him. Here, by "appropriate" we mean that the hash function behaves like a random oracle. This is exactly the assumption of a so-called random oracle model. Now we are able to construct a one-round protocol in this model. This protocol is presented in Figure 7.

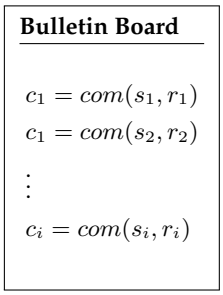


Figure 8: Zerocoin* Bulletin Board

Note that the prover "knows" that the verifier will send $M_2 = H(M_1)$ and can just send both M_1 and M_3 (according to $M_2 = H(M_1)$) in the first message.

The completeness of this protocol is easy to see. The soundness relies on the fact that $M_2 = H(M_1)$ is indeed random. Zero-knowledge can be proven in the random oracle model, but it is beyond the scope of this course.

12.7.3 NIZKs

Such protocols are known as *NIZK*s (non-interactive zero knowledge) in the random oracle model. NIZKs are very useful: imagine for example a blockchain, where you want to prove some statement to the entire world. You can not communicate with each party individually (there are too many of them, and some are possibly not even online at the moment, but will want to verify the statement in the future). In this setting, NIZKs are really the only option.

As mentioned in the previous lecture, Graph 3-Coloring is an NP-complete problem. Thus, each problem in NP can be reduced to Graph 3-Coloring and hence we already have a way to construct a non-interactive zero-knowledge proof for such problems. Note that this may not be the most efficient solution - if you are given some problem it might be better to design a protocol specifically for this problem rather than taking the detour using the Graph 3-Coloring. Still, the fact that we have a solution for each problem in NP makes the introduced solution very powerful.

12.7.4 ZK in the Context of Blockchains

We will now see an example of how zero-knowledge proofs can be used in the context of blockchains. In the following, we will consider a simplified version of the Zerocoin system (denoted by Zerocoin*). The main goal of this system is to make all the transactions unlinkable. In Zerocoin*, the participants are communicating through a bulletin board. The idea is that since there are a lot of communication happening at the same time, it is difficult to notice when some particular participant A is talking to some particular participant B. We will now look into this in more detail.

In Zerocoin*, each coin is a commitment to some serial number s_i (which is just some random number) and is depicted on the bulletin board (see Figure 8). The coins can be spent at most once and there exists a list of the spent coins (see Figure 9). If Alice wants to send some coin A to Bob, coin A gets destroyed and Bob receives a completely new coin B.

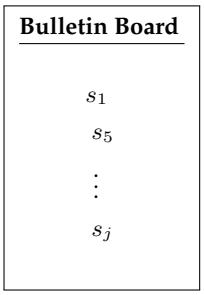


Figure 9: Zerocoin* spent coins

To initially obtain a coin in the Zerocoin* system, a user has to burn a Bitcoin. This gives him the right to write one entry on the bulletin board. We will now discuss the most interesting part: how exactly can some user (Alice) send a Zerocoin* c_i to some other user (Bob).

Attempt 1

1. Alice sends decommitment of c_i (including s_i and randomness r_i) to Bob.
2. Bob creates c_j.
3. Bob broadcasts c_j, c_i, decommitment of c_i.
4. Everyone can verify that s_i is not on the spent coin list and that c_i is a commitment to s_i. If so, c_i is put on the spent list and c_j on the bulletin board.

Now c_i has been destroyed and Bob has a new coin c_j. Bob can spend c_j by sending its decommitment to another party.

The obvious problem with this approach is that everyone sees the relationship between c_i and c_j and we thus do not get the desired unlinkability property – everyone is able to track the whole movement of money.

Idea: use zero-knowledge proofs!

Attempt 2

1. Alice sends decommitment of c_i (including s_i and r_i) to Bob.
2. Bob creates c_j.
3. Bob broadcasts c_j, s_i (but not c_i, r_i) along with the following NIZK statement (denoted as σ in the following): $\exists c_i$ s.t. c_i is a part of the bulletin board and s_i is the value inside c_i.
4. Everyone can verify that s_i is not on the spent coin list and that σ is a valid proof. If so, s_i is put on the spent list and c_j on the bulletin board.

Note that c_i is hidden in **Attempt 2** (because of the zero-knowledge property of the used proof). Furthermore, note that the statement in Step 3 is indeed an NP statement (can be proved by giving a short witness). In our case, we could prove the statement by giving the witness $w = (c_i, s_i, r_i)$, where r_i denotes the randomness used in the computation of c_i. The verifier could then check that c_i is on the bulletin board and c_i is indeed a commitment to s_i when using the randomness r_i.

One possible attack on **Attempt 2** is the following: one can easily change (c_j, s_i, σ) to (c'_j, s_i, σ) after Bob's broadcast in Step 3, as the NIZK σ is completely disconnected from c_j.

Thus, an adversary could use s_i, σ to create a coin c'_j for himself (prior to everyone putting s_i on the spent list and c_j on the board). One possible solution to this problem is to put c_j in the NIZK statement as well. Such modified NIZK could look as following:

- $\exists c_i$ s.t. c_i is a part of the bulletin board, s_i is the value inside c_i and c_j is a valid commitment to some string.

This is still an NP statement, the witness consists of the witnesses of each of the two parts of the AND-statement. Now, if one would try to change c_j without changing σ, the proof would not be valid anymore. However, there is still a minor problem with Attempt 2 – the NIZK can actually be malleable, so one might be able to change σ so that it accounts for the change in c_j. To get around this issue, one has to use the so-called *non-malleable zero knowledge*, which roughly guarantees the following:

- Given a NIZK σ for a statement x no PPT adversary can construct σ' for a related statement x' (unless this adversary already knows the witness to x').

For completeness purposes, we give the final version of the protocol.

Attempt 3 (final)

1. Alice sends decommitment of c_i (including s_i and r_i) to Bob.
2. Bob creates c_j.
3. Bob broadcasts c_j, s_i along with the following non-malleable NIZK statement: $\exists c_i$ s.t. c_i is a part of the bulletin board, s_i is the value inside c_i and c_j is a valid commitment to some string.
4. Everyone can verify that s_i is not on the spent coin list and that the given proof is valid. If so, s_i is put on the spent list and c_j on the bulletin board.

Note that this construction has the following nice property: nobody knows that Alice and Bob are transacting with each other because c_i is not revealed.

Secure Multi-Party Computation

13.1 Introduction

Intuitively, *Secure Multi-Party Computation* allows multiple mutually distrusting parties $P_1, P_2, \ldots, P_n$ compute the result of some function f when applied to the inputs of the parties: $f(x_1, x_2, \ldots, x_n)$ (here, x_i is the input of party P_i). This interaction must reveal nothing except the function result.

Secure multi-party computation was first introduced in 1982 in the paper *Protocols for secure computations*, where Yao's millionaire problem was famously introduced. In this hypothetical problem, two millionaires, Alice and Bob, want to know who of them is richer, but they want to do so without revealing their actual wealth.

13.1.1 Applications

There are many applications of Secure Multi-Party Computation.

- **Privacy Preserving Data Mining:** Privacy Preserving Data Mining is one such application. It would be beneficial to have multiple hospitals run data mining on the combined inputs of their data, but due to regulations like HIPAA, none of the hospitals can share their data with the others. Here, secure multi-party computation can be used to run the data mining algorithm.
- **Information exchange between Intelligence Agencies:** Another application could be between Intelligence agencies of Allied countries to share some information only if there was some information to be gained in return. In this case a secure multi-party computation can be run to verify if there is some valid information on both sides.

Note. *For the remainder of the class we will be focusing on Secure 2-Party Computation (in the following, 2PC). The formal definition for Secure Multi-Party Computation can be generalized from the Secure 2-Party Computation.*

13.2 Formal Definition

Consider parties P_1, P_2 holding inputs x_1, x_2 respectively, and trying to compute $f(x_1, x_2)$.

We say that a protocol π is a secure 2PC for the function $f(\cdot, \cdot)$ if there exists an EPPT simulator $S = (S_1, S_2)$ such that

$$\{View\ of\ corrupt\ P_i^*\ in\ \pi\} =_c \{Output\ of\ S_i\}$$

Recall that in Zero-Knowledge proofs we had a simulator which could produce a view indistinguishable from the real view without knowing the witness. Thus we were able to

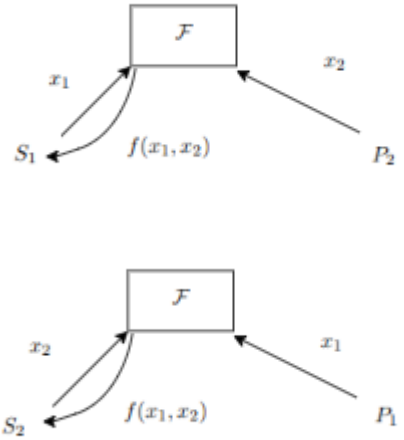


Figure 10: Simulators S_1 and S_2

conclude that no knowledge has been gained during the execution of the protocol. The current definition has a similar flavor: we define a simulator which produces a view indistinguishable from the real view, knowing only $f(x_1, x_2)$. Thus we conclude that no knowledge except for $f(x_1, x_2)$ has been gained during the execution of the protocol.

Note that we still need to formally define the inputs that the Simulator gets.

First Attempt Consider the following: S_i is given as input $f(x_1, x_2)$ and code of P_i^*. Unfortunately, this does not work because $f(x_1, x_2)$ is not defined before the protocol – if some party is malicious, it can use an input different from x_1/x_2.

Correct Definition S_i is given a one time access to an output Oracle, the so-called *ideal functionality* $\mathcal{F}$ (see Figure 12). Intuitively, this means that the simulator can choose an input, and query the oracle to find the corresponding output. The simulator can do so only once during the execution of the protocol.

Note. *This definition is a basic definition of 2PC. It does not capture all the properties such as correctness etc.*

Note. *MPC can be defined analogously by having n parties, n inputs and n Simulators.*

13.3 Types of Adversaries

Note that there are multiple kinds of adversaries:

- Semi-Honest adversary: such adversaries faithfully follow the protocol instructions, but analyze transcript in order to gain some secret information. In other words, this adversary behaves honestly during the execution of the protocol, but is corrupted afterwards.
- Fully malicious: These adversaries are corrupted throughout the execution of the protocol. They can deviate from the protocol arbitrarily.

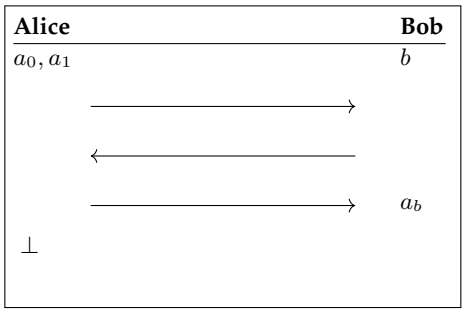


Figure 11: Oblivious Transfer

We can see that semi-honest adversary is weaker than fully malicious adversary, thus any protocol which is secure against fully malicious adversaries is secure against semi-honest adversaries. As we will see later there is a way to convert a protocol secure against semi-honest adversaries to a protocol secure against fully malicious adversaries using *Zero-Knowledge proofs of Honesty*.

13.4 OBLIVIOUS TRANSFER

We will now consider the first example of secure multi-party computation: *oblivious transfer (OT)*. As shown in Figure 11, in the OT setting two parties participate: Alice has inputs a_0, a_1, and Bob has an input bit b. The goal of OT is for Alice to communicate a_b to Bob without gaining any knowledge about the value of b, and without Bob gaining any knowledge about the value of a_{1-b}.

Note that this scenario can be represented as a generalized MPC where the functions learned by each of the parties differ. The definition of the generalized MPC is the following: Parties P_1, P_2 hold inputs x_1, x_2, and wish to compute $f_1(x_1, x_2)$ and $f_2(x_1, x_2)$, respectively. We say that, a protocol π is a secure 2PC for the functions $f_1(\cdot,\cdot)$, $f_2(\cdot,\cdot)$ if there exist EPPT simulators S_1 and S_2 such that

$$\{View\ of\ corrupt\ P_i^*\ in\ \pi\} =_c \{Output\ of\ S_i\},$$

where S_i is given a one time access to an oracle for $f_i(\cdot,\cdot)$.

13.4.1 Protocol against Semi-Honest Adversaries

We will now discuss an OT protocol that is secure against semi-honest adversaries. Let the sender A have input bits (a_0, a_1), and the receiver B have input b Given a family of trapdoor one-way permutations $\{f_i\}$, we define the following protocol:

- A samples (f_i, t), where t is the trapdoor associated with f_i, and sends f_i to B.
- B samples x from the domain of f_i uniformly at random, computes y_0, y_1 s.t. $y_b = f(x)$, and y_{1-b} is a random value chosen from the range of f_i. B sends (y_0, y_1) to A.
- A computes $v_0 = f_i^{-1}(y_0)$ as well as $v_1 = f_i^{-1}(y_1)$ using t, and then computes $c_0 = a_0 \oplus h(v_0)$ and $c_1 = a_1 \oplus h(v_1)$, and sends the computed values to B.

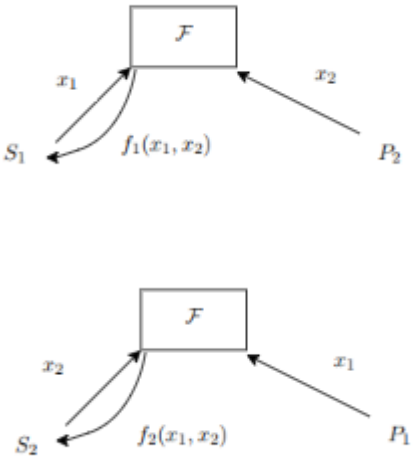


Figure 12: Simulators S_1 and S_2

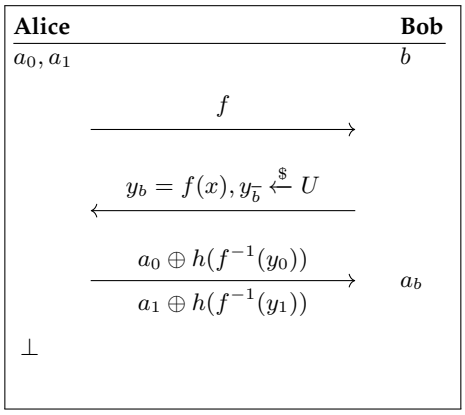


Figure 13: Oblivious Transfer Protocol

- B recover a_b as $c_b \oplus h(f_i^{-1}(y_b)) = c_b \oplus h(x)$.

Security Intuition

Since B does not know trapdoor, B cannot compute the preimage. Since h is a hard-core predicate, $h(f^{-1}(y_{1-b}))$ is a uniform bit to B behaving like a OTP and making a_{1-b} uniform to B. Since B has the preimage of y_b, it can correctly compute a_b. From A's point of view, y_0, y_1 have the same distribution and thus no information can be learned from them.

Formal Proof of Security

First, consider the case where B is corrupt. Then, the simulator S_2 that has input b works as follows:

- S_2 samples (f_i, t) honestly.
- S_2 uses b to compute (y_0, y_1) as specified by the protocol.
- S_2 queries $\mathcal{F}$ and gets a_b. Then, S_2 chooses a_{1-b} at random and computes c_0, c_1 based on a_b and a_{1-b}.
- S_2 outputs the recorded transcript τ_{S_2}.

Lemma 6. Distribution of output of $S_2 =_c$ real world transcript.

Proof. Note that τ_{S_2} and τ_{real} differ only in a_{1-b}. Because h is a hard-core predicate, no adversary is able to predict $h(f^{-1}(y_{1-b}))$ with probability noticeably better than $\frac{1}{2}$. Therefore, bit a_{1-b} cannot be distinguished from random. Thus, the transcripts are indistinguishable. □

Now, consider the case where A is corrupt. Then, the simulator S_1 that has input a_0, a_1 works as follows:

- S_1 samples (f_i, t) to compute the first message honestly.
- S_1 samples (y_0, y_1) uniformly at random from the range of f_i.
- S_1 uses t to invert y_0, y_1 and then uses (a_0, a_1) to compute c_0, c_1 honestly.
- S_1 outputs the recorded transcript τ_{S_1}.

Lemma 7. Distribution of output of $S_1 =_c$ real world transcript.

Proof. Note that τ_{S_1} and τ_{real} differ only in y_b. B chooses x from the domain of f_i and then computes y_b, while S_1 chooses y_b directly from the range of f_i. However, since f_i is a permutation, the distribution of y_b is the same in both cases. Since the same distribution did not change, we know the transcripts are indistinguishable.

□

Note. *This protocol is insecure against Fully malicious adversary. For example, A could choose f_i such that it is not a permutation and thus it can distinguish between y_0, y_1. B could choose y_0, y_1 such that it knows both the preimages.*

13.5 GMW COMPILER

The GMW compiler is a general conversion technique which converts a protocol secure against semi-honest adversaries into a protocol secure against fully malicious adversaries. The basic idea is that at each step the party sends a zero knowledge proof of honestly computing the messages. Before we introduce the correct compiler, we briefly discuss a naive, but unfortunately insecure, construction.

Attempt 1

Take any protocol π and convert it as in Figure 13.4.1, where r_1 and r_2 are the randomness used by Alice and Bob to execute π. Note that the statement "m_1 is consistent with values in c_1" is in NP since given the opening of c_1 the verification is easy.

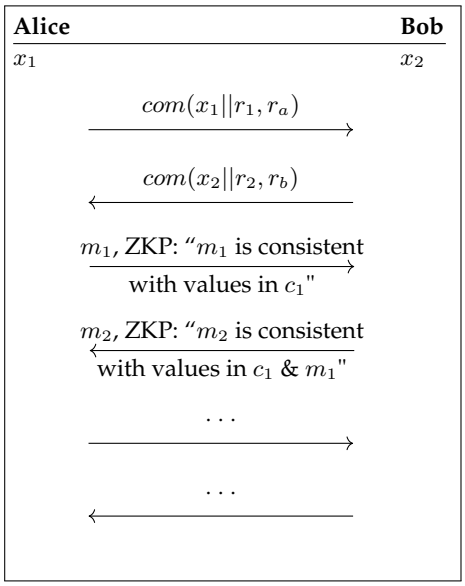


Figure 14: GMW Compiler

Note. *After this commitment, π becomes deterministic.*

Unfortunately, this attempt fails because r_1 and r_2 can be bad randomnesses which could lead to more information to the corresponding parties. For example in the previous example Bob can choose the randomness for y_{1-b} such that y_{1-b}^{-1} is already known to Bob.

Attempt 2 – Correct Construction To solve the problem in the previous attempt, we establish the randomness for both parties in a way that even if one of them is cheating, both parties are still forced to use the same uniform randomness. First, we define a simple protocol for the *coin flipping* problem that allows Alice and Bob to jointly flip a coin over a telephone:

1. A generates r_1 uniformly at random and sends $com(r_1, r_a)$ to B.
2. B generates r_2 uniformly at random and sends r_2 to A.
3. A sends $(r_, r_a)$ to B.
4. A and B both output $r = r_1 \oplus r_2$ as their agreed uniform randomness.

Lemma 8. At the end of this coin flipping protocol, $r = r_1 \oplus r_2$ is uniformly random.

Proof. First, suppose A is dishonest and B is honest. Then there are two cases:

1. A does not generate r_1 uniformly. In this case we still have $r = r_1 \oplus r_2$ is uniformly random, since r_2 is independently random with respect to r_1 and acts like a one-time pad.
2. A changes r_1 to some other string r_1' after it learns r_2 sent by B. However, A cannot do this due to the binding property of the commitment scheme. Since $com(r_1, r_a)$ is bound to r_1, after decommitting, B must have r_1 as the string received from A, otherwise it notices that A is malicious.

Now suppose A is honest and B is dishonest. Since $com(r_1, r_a)$ hides r_1, B intuitively has zero knowledge of r_1 before it generates r_2, and it follows that r_2 is generated independently from r_1. Since r_1 is generated by A uniformly at random and is independent with r_2, it follows that $r = r_1 \oplus r_2$ is still random in this case. □

Using the idea of the above coin flipping protocol, we can modify the protocol defined in our first attempt and construct a fully working GMW compiler which takes any oblivious transfer protocols that are secure against semi-honest adversaries and converts them into protocols that are secure against malicious adversaries. This compiler is defined as follows (let A's message be m_a, and B's message be m_b):

1. A generates r_1 uniformly and sends $com(m_a||r_1, r_a)$ to B. Note that r_1 is neither the randomness used in the commitment scheme nor the entire randomness used by A in the full new protocol (for example, r_a is not part of r_1). Instead, it is just the randomness used by A in the underlying protocol.
2. B generates r_1' uniformly at random and sends r_1' to A.
3. B generates r_2 uniformly and sends $com(m_b||r_2)$ to A. Note that r_2 is neither the randomness used in the commitment scheme, nor the random tape of B. Instead, it is just a random string used for generating B's randomness.
4. A generates r_2' uniformly and sends r_2' to B.
5. A sets its randomness to be $r_A = r_1 \oplus r_1'$, B sets its randomness to be$r_B = r_2 \oplus r_2'$.
6. For any m_j^A in the transcript that A sends to B in the original protocol, A also sends m_j^A to B, followed by running a ZKP protocol to prove the following NP statement: m_j^A is consistent with (1) the transcript so far, and (2) (m_a, r_A). Note that the transcript so far also contains messages sent from B to A.
7. For any m_j^B in the transcript that B sends to A in the original protocol, B also sends m_j^B to A, followed by running a ZKP protocol to prove the following NP statement: m_j^B is consistent with (1) the transcript so far, and (2) (m_b, r_B). Note that the transcript so far also contains messages sent from A to B.

The intuition is still the same: by leveraging the power of ZKP, we not only ensure both parties learn nothing more than if we were simply running an oblivious transfer protocol, but also prevent malicious parties from cheating because of the ZKP's soundness. One side note is that although this compiler is correct and can be used for all OT protocols, it is actually very inefficient since the ZKP involved require reductions to NP-complete problems.

13.6 More General Versions of Oblivious Transfer

In the following, we introduce a few notions that generalize the Oblivious Transfer problem.

13.6.1 1-out-of-n Oblivious Transfer

In this version, instead of only two, the sender now has n items. The 1-out-of-n OT can be constructed fairly simple based on our previous construction of 1-out-of-2 OT protocol:

1. A samples (f, t) from the trapdoor permutation family F_n and sends f to B.

2. B samples $k \in [n]$ and x from the domain of f. B then sets $y_k = f(x)$ and samples each $y_{i \neq k}$ uniformly at random. B then sends $(y_1, y_2, \ldots, y_n)$ to A.
3. A computes $x_i = f^{-1}(y_i)$ using t, and then computes $z_i = h(x_i) \oplus a_i$ for all i and sends $(z_1, z_2, \ldots, z_n)$ to B.
4. B can get a_k by computing $z_k \oplus h(x) = a_k$.

13.6.2 1-out-of-n String Oblivious Transfer

Ideally, we do not want to restrict ourselves to only being able to send bits, and it turns out that we can utilize the above 1-out-of-n Oblivious Transfer protocol to construct a 1-out-of-n string Oblivious Transfer protocol. We assume that $|a_i| = l$ for all i, and we denote by $a_i[j]$ the j-th bit of a_i. The protocol is defined as follows:

- Run the 1-out-of-n bit OT protocol for l times.
  - In the j-th run of the protocol, the input of A will be $(a_1[j], a_2[j], \ldots, a_n[j])$, and the input of B is k.

The correctness of this protocol is easy to see: in the j-th run of the 1-out-of-n bit OT protocol B correctly and securely gets $a_k[j]$. Therefore, after l runs of the above protocol B will get the whole string a_k.

13.6.3 General Secure 2PC for Small Input Size

We can further generalize our OT protocol. In fact, as long as one of the parties has polynomial many inputs, then 1-out-of-n string OT gives a secure 2-party communication protocol. Suppose that A has input x_1, B has x_2, where $x_2 \in [n]$, and A and B want to collaboratively compute a function $f(x_1, x_2)$. The secure 2-party communication protocol is defined as follows:

- $\forall i, 1 \leq i \leq n$, A sets $a_i = f(x_1, i)$.
- B sets $k = x_2$.
- A and B run 1-out-of-n string OT. B gets $a_k = f(x_1, k) = f(x_1, x_2)$. If at the end of the protocol A also wants the result, B can simply send the result back.

Note that the above protocol already solves Yao's millionaire problem. A millionaire can only have finite amount of money in his pocket. It can be millions or billions of dollars, but no matter how much it is, it is still of polynomial size. Hence, we can simply define $f(x_1, x_2)$ such that it returns the larger one of x_1 and x_2, and run the above protocol to get $f(x_1, x_2)$ without revealing how much money the two millionaires have. Note further that this protocol works even when x_1 is of exponential size. As long as B has polynomial many inputs, the protocol will still run in probabilistic polynomial time.

13.7 PRECURSOR TO YAO'S GARBLED CIRCUITS

Recall that in the previous section we have shown how to construct an 1-out-of-n OT based on a 1-out-of-2 construction. Unfortunately, the scheme we defined was not very efficient. Now, we will use a 1-out-of-2 OT in combination with a symmetric encryption scheme to provide an improved construction. Note that symmetric encryption requires only a very weak assumption – that of the existence of a OWF.

Construction Suppose party A has input $(a_1, a_2, \ldots, a_n)$ and party B has input $k \in [n]$. Then, the scheme works as follows:

1. Set $m = \lceil \log n \rceil$. Generate m pairs of random keys $(k_0^1, k_1^1), (k_0^2, k_1^2), \ldots, (k_0^m, k_1^m)$ for the given symmetric encryption scheme. Define sets $S_1, S_2, \ldots, S_n$ such that for all i holds that $|S_i| = m$.
   - Denote i as an m bit string. Let S_i contains exactly $k^j_{i[j]}$ for all $j \in [m]$, where $i[j]$ denotes the j-th bit in the binary representation of i. For example, if $m = 3$ and $i = 2$ (010 in binary), then $S_2 = \{k_0^0, k_1^1, k_0^2\}$. Note that by this construction, if $i \neq j$ we must have $S_i \neq S_j$ since i and j must have different bit string representation.
2. A encrypts a_i using all keys from set S_i, Denote the resulting ciphertext as c_i. This is done for all $i \in [n]$. Then, A then sends $(c_1, c_2, \ldots, c_n)$ to B.
3. A and B execute m instances of 1-out-of-2 OT protocol. In the i-th execution of the protocol, input of A is (k_0^i, k_1^i), and input of B is $k[i]$.
4. After m executions of 1-out-of-2 OT protocol, it is clear that B has all the keys used to encrypt a_k. Since the keys are symmetric encryption keys, B can then decrypt c_k using these keys.

Note that in terms of performance, this construction is much better than the 1-out-of-n scheme we defined in the previous section. In the previous scheme we had to make n calls of trapdoor permutation (which are very expensive), but now we only need to make $2m$ calls of trapdoor permutation in Step 3, and m is logarithmic in n. Of course, we have to consider the cost of encrypting, but typically symmetric encryption is much cheaper than trapdoor permutation, so the protocol above still has a much better performance.

Note that above, we made significant progress towards Secure 2 Party Computation by giving solutions for functions that operate on domains of polynomial size. At FOCS'86, Yao gave a solution for general circuits. This construction was cited as having contributed to the decision to grant Yao the prestigious Turing Award in the year 2000. It is, in some sense, a generalization of the construction we presented above. We will now present Yao's construction.

13.8 YAO'S GARBLED CIRCUITS

The following construction applies to boolean circuits that take two inputs x_1 and x_2 and compute $f(x_1, x_2)$. For clarity we restrict ourselves to the case where $f(x_1, x_2)$ is a single bit. An example of such a circuit is given by Figure 15. The circuits can be built with any gates that have two input wires and one output wire.

The high level idea is as follows: the sender will send the encrypted (in some sense) circuit to the receiver. The receiver will then use Oblivious Transfer to evaluate the circuit on its input, with the help of the sender. In the following we denote by P_1 the sender and by P_2 the receiver:

1. For every wire w in the circuit, sender P_1 will generate two random keys: (K_0^w, K_1^w). K_b^w represents the idea of the wire having value b.
2. P_2 will eventually need to receive the keys corresponding to the input $x_1||x_2$.
3. P_1 will create "gate tables" that enable P_2, given keys for the input wires to the gate, to recover the key for output wire. Consider a gate that has input wires i, j and output ℓ, and which implements functionality $g(\cdot, \cdot)$. Given keys K_0^i, K_0^j, P_2 can recover

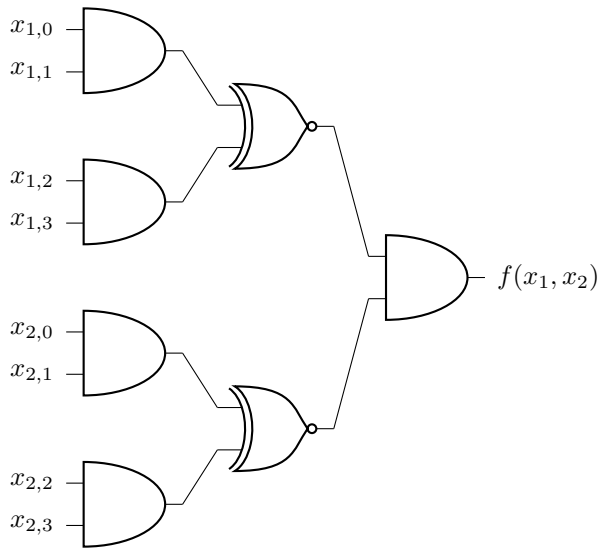


Figure 15: An example of a boolean circuit that could be securely evaluted using Yao's Garbled Circuits.

$K^{\ell}_{g(0,0)}$. Similarly given keys for K^i_0, K^j_1, P_2 can recover $K^{\ell}_{g(0,1)}$ and so on. The natural way to do this is to encrypt the $K^{\ell}_{g(b,b')}$ under K^i_b, $K^j_{b'}$. Formally, the gate table for a gate (i, j, ℓ) has 4 doubly-encrypted entries:

$$E(K^i_b, E(K^j_{b'}, K^{\ell}_{g(b,b')}||0 \ldots 0)), \forall b, b' \in \{0, 1\}$$

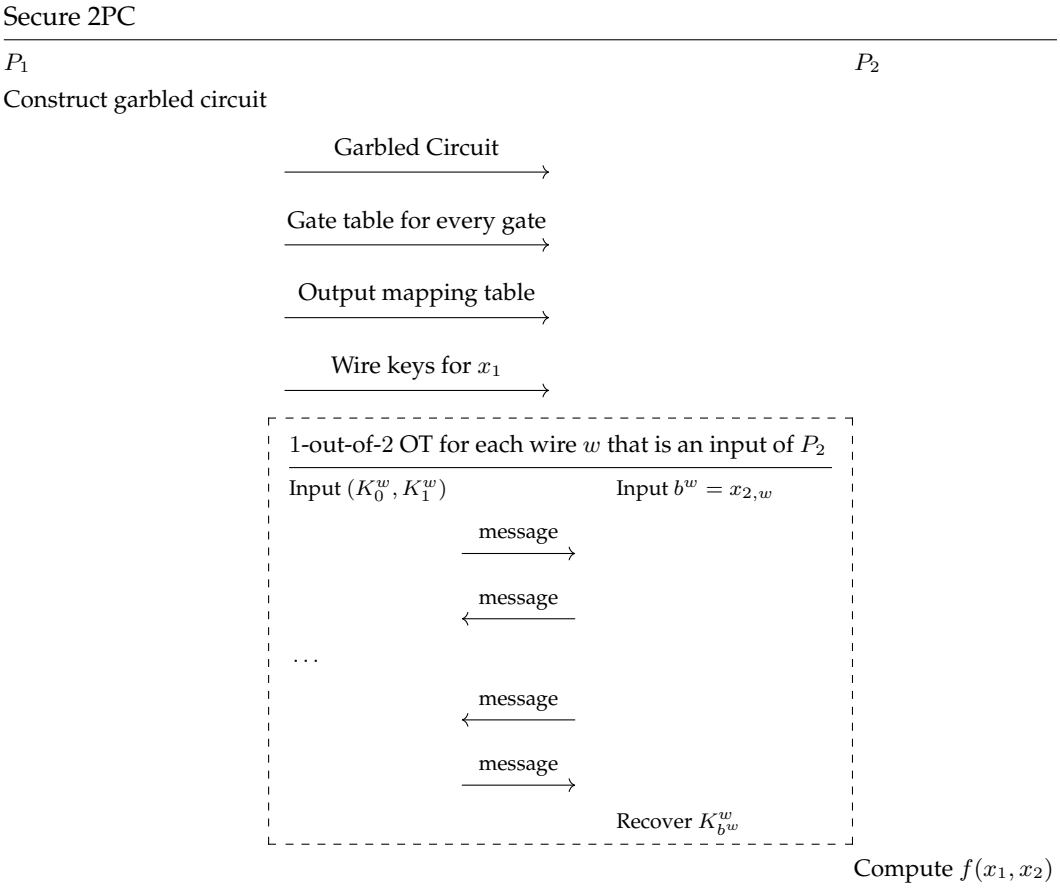
Note that we actually appended a number of zeroes after the value $g(b, b')$ before encrypting. This is because when P_2 does try to evaluate that gate, it will need some way to know that it succeeded. This can be done by checking that the λ last bits of the decryption are zeros, where λ is the security parameter, which would otherwise happen with negligible probability.

Additionally, note that if we were to send the gate table as such it would be easy for P_2 to tell what bits were being input to the gate by simply looking at the row where the matching with the zeros happened. P_1 will thus have to shuffle the rows. P_1 thus randomly permutes the 4 entries and sends all gate tables to P_2.

4. Given keys for the inputs wires and the gate tables, P_2 can compute a key for the output wire. But P_2 wants the output, not the key corresponding to the output.
5. Say w_{out} is the output wire. P_1 sends a mapping of keys to bits for w_{out}. More precisely, P_1 sends $(K^{w_{out}}_0, 0), (K^{w_{out}}_1, 1)$, where $K^{w_{out}}_b$ represents the idea of the output wire having value b. As the protocol executes P_2 will learn one of those values.

We have described how the circuit gets garbled and how it can then be evaluated given the input keys. All that remains to be clarified is how P_2 acquires the input keys in the first place. For value x_1, P_1, knowing x_1, can simply send the right keys to P_2 directly. The rest of the keys will be obtained via 1-out-of-2 OT. Recall that we are in the semi-honest setting

(the malicious setting can be solved using the compiler we studied during last lecture). If we want P_1 to also learn the output, then P_2 should also send it to P_1 at the end.



This protocol is highly efficient, in fact it is one of the most practical ideas for secure 2PC even to date. In the semi-honest setting it mostly relies on symmetric encryption. There are optimizations that make it possible to go to the malicious setting without relying on expensive Zero Knowledge techniques. One such optimization is based on an idea called *Cut-and-Choose*. P_1 creates a large number of circuits. P_2 asks P_1 to reveal the secrets of about half of them chosen at random to check that P_1 generated them honestly. Then, they evaluate the rest of the circuits and P_1 bases the value of $f(x_1, x_2)$ on plurality voting. There are also fairly efficient constructions for OT based on the DDH hardness assumption that are directly secure again malicious adversaries.

13.9 CONSTRUCTION FOR MPC

We now give a construction that enables multiple parties $P_1, P_2, \ldots, P_n$ with inputs $x_1, x_2, \ldots, x_n$ to compute a function $f(x_1, \ldots, x_n)$. Assume any two parties P_i, P_j have

a private channel between them. Again we only give a solution in the semi-honest setting, and we additionally assume that the parties do not collude.

1. P_1 creates a garbled circuit for f and sends it to P_n;
2. P_1 sends keys corresponding to x_1 to P_n;
3. P_n gets the keys corresponding to x_n from P_1 via 1-out-of-2 OT.
4. $\forall i$ such that $2 \leq i \leq n-1$, P_i gets the keys corresponding to x_i from P_1 via 1-out-of-2 OT. P_i then sends these keys to P_n.
5. P_n evaluates the circuit and announces its output $f(x_1, \ldots, x_n)$.

Theorem 20. This protocol is secure as long as the parties don't collude with each other.

Notice that if P_1 colludes with P_n they can recover every other party's input.

Goldreich, Micali and Wigderson (1987) also give a solution that remains secure even if $n-1$ parties collude with each other.

Additional Topics

In this last chapter, we would like to briefly explore the areas of cryptography we haven't touched upon yet.

14.1 Public-key Infrastructure

Assume Alice wants to send a message to Bob, but they do not have a shared key. Alice will first need to securely recover Bob's public key. We know that key exchange protocols exist, but they are not secure under Man-In-The-Middle Attacks, and on the internet, it might be difficult for Alice to know that she's actually exchanging a key with Bob (e.g. think of Bob as a website). *Certificate Authorities (CA)* exist to address this problem. At a high level they work as follows: Web browsers have the CA's public key hardcoded. Websites must go to the CA and get a certificate: a signed statement (from the CA's private key) that a given public key (the website's public key) is the valid public key of the site (identified by its domain name). That certificate lets the browser check the validity of the public key so long as the certificate authority remains trustworthy.

14.2 Fully-Hommorphic Encryption

Assume a situation involving a client and a server. The client (say, a hospital) wants to upload encrypted records (e.g., patient records) to the server (e.g., a cloud provider). Later on, the client decides it wants to perform computations on records (e.g., compute some statistics to help it do a better job at detecting or treating certain pathologies). Downloading the entire dataset on hospital servers is cumbersome and somewhat defeats the purpose of cloud computing. Sending the key to the server so it decrypts the data and performs the computations is harmful to the privacy of the patients, and in some cases might even be illegal. Instead, using *Fully Homomorphic Encryption*, the client can send a function f to the server and the server can compute and return $Enc(f(x))$, where x denotes the set of records. Fully Homomorphic Encryption was first envisionned in the late 70s, but no solution was available. The existence of a solution remained unknown until Craig Gentry, then one of Dan Boneh's PhD students at Stanford, gave a scheme based on lattices in 2008.

Note that schemes for simple homomorphisms had been known for a long time. ElGamal encryption is multiplicatively homomorphic. There are also additively homomorphic schemes. Some schemes are even additively homomorphic and allow one multiplication computation. Achieving additive and multiplicative homomorphism at the same time is the key to enabling the computation of any circuit.

14.3 NON-MALLEABLE CRYPTOGRAPHY

Consider a closed bid (first-price) auction. The bidders want to keep the values of their bids private. A natural thing to do might be to have every participant commit to their bid, send the commitment to the seller, and then, after the auction closes, reveal their bid. Not that encryption is clearly not good enough for this use case because it doesn't have any binding property. But commitments may not be good enough either: there is nothing in the definition of commitment schemes that says it shouldn't be possible for a bidder to intercept some other bidder's commitment and to compute $Com(b_1 + 1)$ from $Com(b_1)$.

This justifies introducing a new primitive called non-malleable commitments, that would provide such a guarantee.

Similarly, there are non-malleable encryption schemes and non-malleable Zero Knowledge proof systems (which roughly means that a verifier can't use a proof from some prover to prove a related statement).

14.4 ATTRIBUTE BASED ENCRYPTION

In many situations, organizations are interested in implementing Access Control on their data, so that whether one of their members can access the data or not depends on certain permissions that may or may not be granted to them. Often, this is done using software checks, but *Attribute Based Encryption* provides a cryptographic solution. An authority is charged with distributing keys according to every party's status within the organization. The cryptographic primitive ensures that you will only be able to decrypt data you have been granted access to. This means it's now theoretically possible to allow parties to download whatever data they want to, because they will not be able to decrypt anything they are not supposed to see.

14.5 PROGRAM OBFUSCATION

Program Obfuscation is an old problem in computer science. A typical use case would be that of a software vendor who would like to distribute software in such a way that it can't be modified or analyzed. Many heuristics have been proposed, but all have eventually been broken. In fact, in 2001, Barak et al. showed that an ideal definition of Software Obfuscation they termed Black Box Obfuscation was in fact impossible to achieve. They suggested that Indistinguishibility Obfuscation, a relaxed definition, might be a more reasonable goal. In 2013, Garg et al. gave a candidate solution for Indistinguishibility Obfuscation, and the problem has been an important area of cryptographic research in recent years. All solutions so far have been based on novel assumptions on which the community's confidence is limited, but one may hope that a satisfactory solution will appear in the next decade.

14.6 POSITION BASED CRYPTOGRAPHY

The cryptographic schemes we studied in this class were mostly based on keys. But sometimes, where you are physically might be a more important factor. For instance, the US Army in the Pentagon, near Washington DC, might want to send a message that could only be read by parties physically located in some US military base in Afghanistan. This is

a high level description of Position Based Encryption, but one may also think about Position Based Signatures, Position Based Key Exchange, or more generally about the area of *Position Based Cryptography*. An important problem is that of secure positioning: a prover is trying to prove his physical coordinates to some verifier, and solutions exist that do not rely on a trusted third party.

Last but not Least

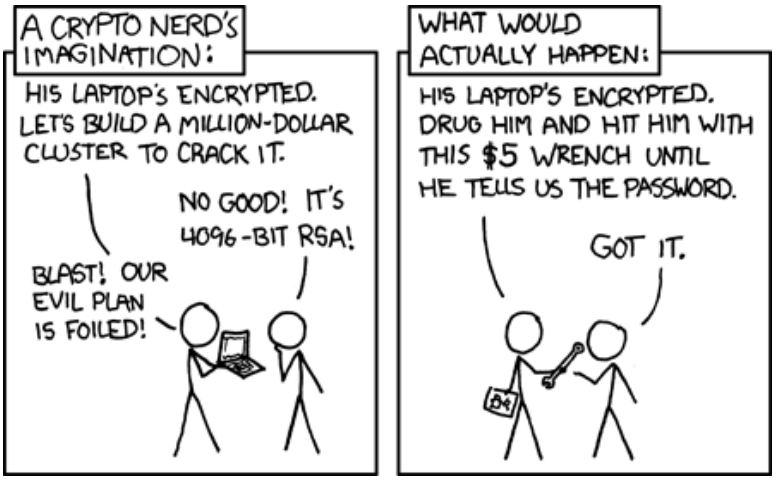


Figure 16: Comic from https://xkcd.com/538/