A Course in Cryptography

Rafael Pass
Abhi Shelat

11 11 11 11 11   15 14 13 12 11 10 9

First edition: June 2007
Second edition: September 2008

Contents

List of Algorithms and Protocols

List of Definitions

Preface

We would like to thank the students of CS 687 (Stephen Chong, Michael Clarkson, Michael George, Lucja Kot, Vikram Krishnaprasad, Huijia Lin, Jed Liu, Ashwin Machanavajjhala, Tudor Marian, Thanh Nguyen, Ariel Rabkin, Tom Roeder, Wei-lung Tseng, Muthuramakrishnan Venkitasubramaniam and Parvathinathan Venkitasubramaniam) for scribing the original lecture notes which served as a starting point for these notes. In particular, we are very grateful to Muthu for compiling these original sets of notes.

RAFAEL PASS
Ithaca, NY

ABHI SHELAT
Charlottesville, VA
August 2007

Notation

Algorithms

Let A denote an algorithm. We write $A(\cdot)$ to denote an algorithm with one input and $A(\cdot, \cdot)$ for two inputs. In general, the output of a (randomized) algorithm is described by a probability distribution; we let $A(x)$ denotes the probability distribution associated with the output of A on input x. An algorithm is said to be deterministic if the probability distribution is concentrated on a single element.

Experiments

We denote by $x \leftarrow S$ the experiment of sampling an element x from a probability distribution S. If F is a finite set, then $x \leftarrow F$ denotes the experiment of sampling *uniformly* from the set F. We use semicolon to describe the ordered sequences of event that make up an experiment, e.g.,

$$x \leftarrow S;\ (y, z) \leftarrow A(x)$$

Probabilities

If $p(.,.)$ denotes a predicate, then

$$\Pr[x \leftarrow S;\ (y, z) \leftarrow A(x) : p(y, z)]$$

is the probability that the predicate $p(y, z)$ is true after the ordered sequence of events $(x \leftarrow S;\ (y, z) \leftarrow A(x))$. The notation $\{x \leftarrow S;\ (y, z) \leftarrow A(x) : (y, z)\}$ denotes the probability distribution over $\{y, z\}$ generated by the experiment $x \leftarrow S;\ (y, z) \leftarrow A(x)$.

Chapter 1

Introduction

The word cryptography stems from the Greek words *kryptós*—meaning “hidden”—and *gráfein*—meaning “ to write”. Indeed, the classical cryptographic problem, which dates back millenia, considers the task of using “hidden writing” to secure, or conceal communication between two parties.

1.1 Classical Cryptography: Hidden Writing

Consider two parties, Alice and Bob. Alice wants to *privately* send messages (called *plaintexts*) to Bob over an *insecure channel*. By an insecure channel, we here refer to an “open” and tappable channel; in particular, Alice and Bob would like their privacy to be maintained even in face of an *adversary* Eve (for eavesdropper) who listens to all messages sent on the channel. How can this be achieved?

A possible solution Before starting their communication, Alice and Bob agree on some “secret code” that they will later use to communicate. A secret code consists of a *key*, an algorithm, Enc, to *encrypt* (scramble) plaintext messages into *ciphertexts* and an algorithm Dec to *decrypt* (or descramble) ciphertexts into plaintext messages. Both the encryption and decryption algorithms require the key to perform their task.

Alice can now use the key to encrypt a message, and the send the ciphertext to Bob. Bob, upon receiving a ciphertext, uses the key to decrypt the ciphertext and retrieve the orignal message.

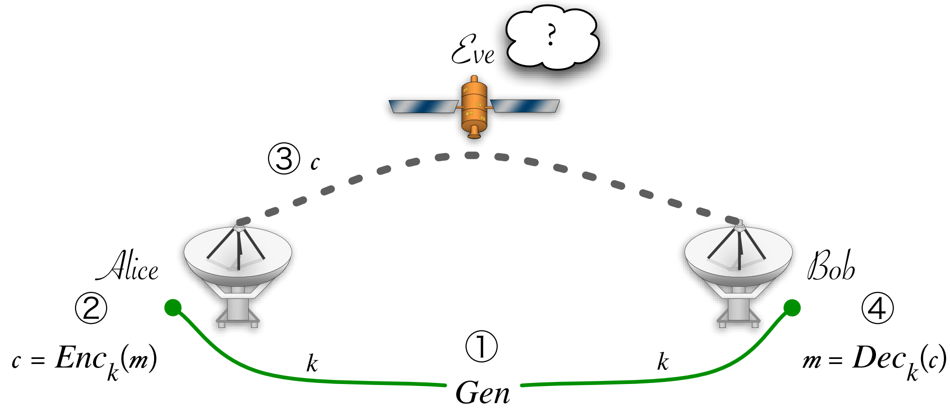


Figure 1.1: Illustration of the steps involved in private-key encryption. First, a key k must be generated by the Gen algorithm and privately given to Alice and Bob. In the picture, this is illustrated with a green "land-line." Later, Alice encodes the message m into a ciphertext c and sends it over the insecure channel—in this case, over the airwaves. Bob receives the encoded message and decodes it using the key k to recover the original message m. The eavesdropper Eve does not learn anything about m except perhaps its length.

1.1.1 Private-Key Encryption

To formalize the above task, we must consider an additional algorithm, Gen, called the *key-generation* algorithm; this algorithm is executed by Alice and Bob to generate the key k which they use to encrypt and decrypt messages.

A first question that needs to be addressed is what information needs to be "public"—i.e., known to everyone—and what needs to be "private"—i.e., kept secret. In the classical approach—*security by obscurity*—all of the above algorithms, Gen, Enc, Dec, and the generated key k were kept private; the idea was of course that the less information we give to the adversary, the harder it is to break the scheme. A design principle formulated by Kerchoff in 1884—known as *Kerchoff's principle*—instead stipulates that the only thing that one should assume to be private is the key k; everything else (i.e., Gen, Encand Dec) should be assumed to be public! Why should we do this? Designs of encryption algorithms are often eventually leaked, and when this happens the effects to privacy could be disastrous. Suddenly the scheme might be completely broken; this might even be the case if just a part of the algorithm's description is leaked. The more conservative approach advocated by Kerchoff instead guarantees that security is preserved even if everything but the key is known to the adversary. Furthermore, if a publicly known encryption scheme still has not been broken, this gives us more confidence in its "true" security (rather than if only the few people that designed it were unable to break it). As we will see later, Kerchoff's principle will be the first step to formally defining the security of encryption schemes.

Note that an immediate consequence of Kerchoff's principle is that all of the algorithms Gen, Enc, Dec can not be *deterministic*; if this were so, then Eve would be able to compute everything that Alice and Bob could compute and would thus be able to decrypt anything that Bob can decrypt. In particular, to prevent this we must require the key generation algorithm, Gen, to be randomized.

Definition 3.1 (Private-key Encryption). A triplet of algorithms (Gen, Enc, Dec) is called a *private-key encryption scheme* over the messages space $\mathcal{M}$ and the keyspace $\mathcal{K}$ if the following holds:

1. Gen(called the *key generation algorithm*) is a randomized algorithm that returns a key k such that $k \in \mathcal{K}$. We denote by $k \leftarrow \mathsf{Gen}$ the process of generating a key k.
2. Enc (called the *encryption algorithm*) is an (potentially randomized) algorithm that on input a key $k \in K$ and a message $m \in M$, outputs a ciphertext c. We denote by $c \leftarrow \mathsf{Enc}_k(m)$ the output of Enc on input key k and message m.
3. Dec (called the *decryption algorithm*) is a deterministic algorithm that on input a key k and a ciphertext c and outputs a message m.
4. For all $m \in M$,

$$\Pr[k \leftarrow \mathsf{Gen} : Dec_k(Enc_k(m)) = m] = 1$$

To simplify notation we also say that $(\mathcal{M}, \mathcal{K}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a private-key encryption scheme if (Gen, Enc, Dec) is a *private-key encryption scheme* over the messages space $\mathcal{M}$ and the keyspace $\mathcal{K}$. To simplify further, we sometimes say that $(\mathcal{M}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a private-key encryption scheme if there exists some key space $\mathcal{K}$ such that $(\mathcal{M}, \mathcal{K}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a private-key encryption scheme.

Note that the above definition of a private-key encryption scheme does not specify any secrecy (or privacy) properties; the only non-trivial requirement is that the decryption algorithm Dec uniquely recovers the messages encrypted using Enc (if these algorithms are run on input the same key k).

Later in course we will return to the task of defining secrecy. However, first, let us provide some historical examples of private-key encryption schemes and colloquially discuss their "security" without any particular definition of secrecy in mind.

1.1.2 Some Historical Ciphers

The *Ceasar Cipher* (named after Julius Ceasar who used it to communicate with his generals) is one of the simplest and well-known private-key encryption schemes. The encryption method consist of replacing each letter in the message with one that is a fixed number of places down the alphabet. More precisely,

Definition 4.1. The *Ceasar Cipher* denotes the tuple $(\mathcal{M}, \mathcal{K}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ defined as follows.

$$\begin{aligned}
\mathcal{M} &= \{A, B, \ldots, Z\}^* \\
\mathcal{K} &= \{0, 1, 2, \ldots, 25\} \\
\mathsf{Gen} &= k \text{ where } k \xleftarrow{r} \mathcal{K}. \\
Enc_k(m_1 m_2 \ldots m_n) &= c_1 c_2 \ldots c_n \text{ where } c_i = m_i + k \bmod 26 \\
Dec_k(c_1 c_2 \ldots c_n) &= m_1 m_2 \ldots m_n \text{ where } m_i = c_i - k \bmod 26
\end{aligned}$$

In other words, encryption is a cyclic shift of the same length (k) on each letter in the message and the decryption is a cyclic shift in the opposite direction. We leave it for the reader to verify the following proposition.

Proposition 4.1. *The* Caesar Cipher *is a private-key encryption scheme.*

At first glance, messages encrypted using the Ceasar Cipher look "scrambled" (unless k is known). However, to break the scheme we just need to try all 26 different values of k (which is easily done) and see if what we get back is something that is readable. If the message is "relatively" long, the scheme is easily broken. To prevent this simple *brute-force* attack, let us modify the scheme.

In the improved *Substitution Cipher* we replace letters in the message based on an arbitrary permutation over the alphabet (and not just cyclic shifts as in the Caesar Cipher).

Definition 4.2. The *Subsitution Cipher* denotes the tuple $\mathcal{M}, \mathcal{K}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}$ defined as follows.

$$\begin{aligned}
\mathcal{M} &= \{A, B, \ldots, Z\}^* \\
\mathcal{K} &= \text{the set of permutations over } \{A, B, \ldots, Z\} \\
\mathsf{Gen} &= k \text{ where } k \stackrel{r}{\leftarrow} \mathcal{K}. \\
Enc_k(m_1 m_2 \ldots m_n) &= c_1 c_2 \ldots c_n \text{ where } c_i = k(m_i) \\
Dec_k(c_1 c_2 \ldots c_n) &= m_1 m_2 \ldots m_n \text{ where } m_i = k^{-1}(c_i)
\end{aligned}$$

Proposition 5.1. *The* Subsitution Cipher *is a private-key encryption scheme.*

To attack the substitution cipher we can no longer perform the brute-force attack because there are now $26!$ possible keys. However, by performing a careful frequency analysis of the alphabet in the English language, the key can still be easily recovered (if the encrypted messages is sufficiently long)!

So what do we do next? Try to patch the scheme again? Indeed, cryptography historically progressed according to the following "crypto-cycle":

1. **A**, the "artist", invents an encryption scheme.

2. **A** claims (or even mathematically proves) that *known attacks* do not work.

3. The encryption scheme gets employed widely (often in critical situations).

4. The scheme eventually gets broken by improved attacks.

5. Restart, usually with a patch to prevent the previous attack.

Thus, historically, the main job of a cryptographer was *cryptanalysis*—namely, trying to break encryption algorithms. Whereas cryptoanalysis i still is an important field of research, the philosophy of modern cryptography is instead "if we can do the cryptography part right, there is no need for cryptanalysis!".

1.2 Modern Cryptography: Provable Security

Modern Cryptography is the transition from cryptography as an *art* to cryptography as a principle-driven *science*. Instead of inventing ingenious ad-hoc schemes, modern cryptography relies on the following paradigms:

- Providing mathematical *definitions of security*.

- Providing *precise mathematical assumptions* (e.g. "factoring is *hard*", where *hard* is formally defined). These can be viewed as axioms.
- Providing *proofs of security*, i.e., proving that, if some particular scheme can be broken, then it contradicts our assumptions (or axioms). In other words, if the assumptions were true, the scheme cannot be broken.

This is the approach that we will consider in this course.

As we shall see, despite its conservative nature, we will be able to obtain solutions to very paradoxical problems that reach far beyond the original problem of security communication.

1.2.1 Beyond Secure Communication

In the original motivating problem of secure communication, we had two honest parties, Alice and Bob and a malicious eavesdropper Eve. Suppose, Alice and Bob in fact do not trust each other but wish to perform some joint computation. For instance, Alice and Bob each have a (private) list and wish to find the intersection of the two list without revealing anything else about the contents of their lists. Such a situation arises, for example, when two large financial institutions which to determine their "common risk exposure," but wish to do so without revealing anything else about their investments. One good solution would be to have a trusted center that does the computation and reveals only the answer to both parties. But, would either bank trust the "trusted" center with their sensitive information? Using techniques from modern cryptography, a solution can be provided without a trusted party. In fact, the above problem is a special case of what is known as *secure two-party computation*.

Secure two-party computation - informal definition: A secure two-party computation allows two parties A and B with private inputs a and b respectively, to compute a function $f(a, b)$ that operates on joint inputs a, b while guaranteeing the same *correctness* and *privacy* as if a trusted party had performed the computation for them, even if either A or B try to deviate in any possible malicious way.

Under certain number theoretic assumptions (such as "factoring is hard"), there exists a protocol for secure two-party computation.

The above problem can be generalized also to situations with multiple distrustful parties. For instance, consider the task of electronic elections: a set of n parties which to perform an election in which it is guaranteed that the votes are correctly counted, but these votes should at the same time remain private! Using a so called *multi-party computation* protocol, this task can be achieved.

A toy example: The match-making game

To illustrate the notion of secure-two party computation we provide a "toy-example" of a secure computation using physical cards. Alice and Bob want to find out if they are meant for each other. Each of them have two choices: either they love the other person or they do not. Now, they wish to perform some interaction that allows them to determine whether there is a match (i.e., if they both love each other) or not—and nothing more! For instance, if Bob loves Alice, but Alice does not love him back, Bob does not want to reveal to Alice that he loves her (revealing this could change his future chances of making Alice love him). Stating it formally, if LOVE and NO-LOVE were the inputs and MATCH and NO-MATCH were the outputs, the function they want to compute is:

$$\begin{aligned} f(\text{LOVE}, \text{LOVE}) &= \text{MATCH} \\ f(\text{LOVE}, \text{NO-LOVE}) &= \text{NO-MATCH} \\ f(\text{NO-LOVE}, \text{LOVE}) &= \text{NO-MATCH} \\ f(\text{NO-LOVE}, \text{NO-LOVE}) &= \text{NO-MATCH} \end{aligned}$$

Note that the function f is simply an *and* gate.

The protocol: Assume that Alice and Bob have access to five cards, three identical hearts(♡) and two identical clubs(♣). Alice and Bob each get one heart and one club and the remaining heart is put on the table, turned over.

Next Alice and Bob also place their card on the table, also turned over. Alice places her two cards on the left of the heart which is already on the table, and Bob places the hard on the right of the heart. The order in which Alice and Bob place their two cards depends on their input (i.e., if they love the other person or not). If Alice loves, then Alice places her cards as ♣♡; otherwise she places them as ♡♣. Bob on the other hand places his card in the opposite order: if he loves, he places ♡♣, and otherwise places ♣♡. These orders are illustrated in Fig. 1.2.

When all cards have been placed on the table, the cards are piled up. Alice and Bob then each take turns to cut the pile of cards (once each). Finally, all cards are revealed. If there are three hearts in a row then there is a match and no-match otherwise.

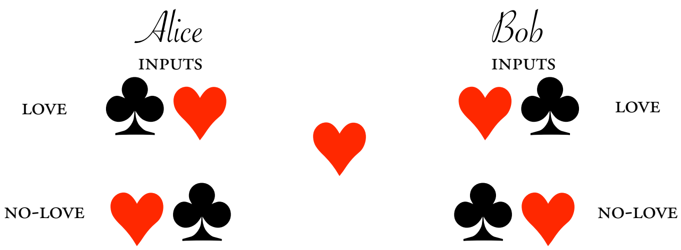


Figure 1.2: Illustration of the Match game with Cards

Analyzing the protocol: We proceed to analyze the above protocol. Given inputs for Alice and Bob, the configuration of cards on the table before the cuts is described in Fig. 1.3. Only the first case—i.e., (LOVE, LOVE)—results

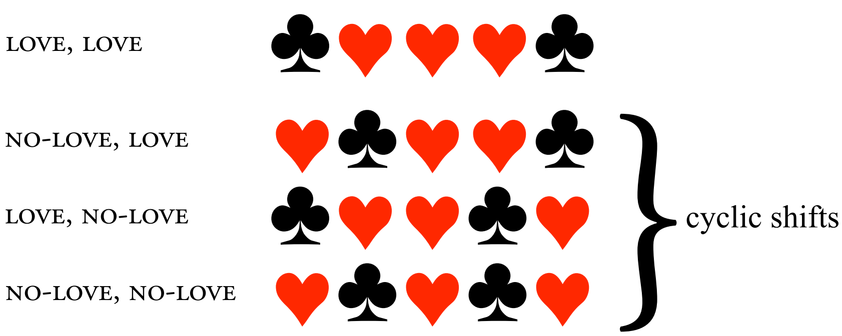


Figure 1.3: The possible outcomes of the Match Protocol. In the case of a mismatch, all three outcomes are cyclic shifts of one-another.

in three hearts in a row. Furthermore this property is not changed by the cyclic shift induced by the cuts made by Alice and Bob. We conclude that the protocols correctly computes the desired function.

Next, note that in the remaining three cases (when the protocol will output NO-MATCH, all the above configurations are cyclic shifts of one another. If one of Alice and Bob is honest—and indeed performs a random cut— the final card configuration that the parties get to see is identically distributed no matter which of the three cases we were in, in the first place. Thus, even if one of Alice and Bob tries to deviate in the protcol (by not performing a random cut), the privacy of the other party is still maintained.

Zero-knowledge proofs

Zero knowledge proofs is a special case of a secure computation. Informally, in a Zero Knowledge Proof there are two parties, Alice and Bob. Alice wants to convince Bob that some statement is true; for instance, Alice wants to convince Bob that a number N is a product of two primes p, q. A trivial solution would be for Alice to send p and q to Bob. Bob can then check that p and q are primes (we will see later in the course how this can be done) and next multiply the numbers to check if their products N. But this solution reveals p and q. Is this necessary? It turns out that the answer is no. Using a zero-knowledge proof Alice can convince Bob of this statement without revealing p and q.

1.3 Shannon's Treatment of Provable Secrecy

Modern (provable) cryptography started when Claude Shannon formalized the notion of private-key encryption. Thus, let us return to our original problem of securing communication between Alice and Bob.

1.3.1 Shannon Secrecy

As a first attempt, we might consider the following notion of security:

> The adversary cannot learn (all or part of) the key from the ciphertext.

The problem, however, is that such a notion does not make any guarantees about what the adversary can learn about the *plaintext* message! Another approach might be:

> The adversary cannot learn (all, part of, any letter of, any function of, or any partial information about) the plaintext.

This seems like quite a strong notion. In fact, it is too strong because the adversary may already possess some partial information about the plaintext that is acceptable to reveal. Informed by these attempts, we take as our intuitive definition of security:

> Given some *a priori* information, the adversary cannot learn any additional information about the plaintext by observing the ciphertext.

Such a notion of secrecy was formalized by Claude Shannon in 1949 **[?]** in his seminal paper constituting the birth of modern cryptography.

Definition 9.1 (Shannon secrecy). A private-key encryption scheme $(\mathcal{M}, \mathcal{K}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is said to be *Shannon-secret with respect to the distibution D* over $\mathcal{M}$ if for all $m' \in \mathcal{M}$ and for all c,

$$\mathsf{Pr}\left[k \leftarrow \mathsf{Gen}; m \leftarrow D : m = m' \mid Enc_k(m) = c\right] = \mathsf{Pr}\left[m \leftarrow D : m = m'\right].$$

An encryption scheme is said to be *Shannon secret* if it is Shannon secret with respect to all distributions D over $\mathcal{M}$.

The probability is taken with respect to the random output of Gen, the choice of m and the randomization of Enc. The quantity on the left represents the adversary's *a posteriori* distribution on plaintexts after observing a ciphertext; the quantity on the right, the *a priori* distribution. Since these distributions are required to be equal, we thus require that the adversary does not gain any additional information by observing the ciphertext.

1.3.2 Perfect Secrecy

To gain confidence in our definition, we also provide an alternative approach to defining security of encryption schemes. The notion of *perfect secrecy* requires that the distribution of ciphertexts for any two messages are identical. This formalizes our intuition that the ciphertexts carry no information about the plaintext.

Definition 10.1 (Perfect Secrecy). A private-key encryption scheme $(\mathcal{M}, \mathcal{K}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is *perfectly secret* if for all m_1 and m_2 in $\mathcal{M}$, and for all c,

$$\mathsf{Pr}\left[k \leftarrow \mathsf{Gen} : Enc_k(m_1) = c\right] = \mathsf{Pr}\left[k \leftarrow \mathsf{Gen} : Enc_k(m_2) = c\right].$$

Perhaps not surprisingly, the above two notions of secrecy are equivalent.

Theorem 10.1. *An encryption scheme is perfectly secrect if and only if it is Shannon secret.*

Proof. We prove each implication separately. To simplify the proof, we let the notation $\mathsf{Pr}_k[\cdot]$ denote $\mathsf{Pr}[k \leftarrow \mathsf{Gen}]$, $\mathsf{Pr}_m[\cdot]$ denote $\mathsf{Pr}[m \leftarrow D : \cdot]$, and $\mathsf{Pr}_{k,m}[\cdot]$ denote $\mathsf{Pr}[k \leftarrow \mathsf{Gen}; m \leftarrow D : \cdot]$.

Perfect secrecy implies Shannon secrecy. Assume that $(\mathcal{M}, \mathcal{K}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is perfectly secret. Consider any distribution D over $\mathcal{M}$, any message $m' \in \mathcal{M}$, and any c. We show that

$$\mathsf{Pr}_{k,m}\left[m = m' \mid Enc_k(m) = c\right] = \mathsf{Pr}_m\left[m = m'\right].$$

By the definition of conditional probabilities, the l.h.s. can be rewritten as

$$\frac{\mathsf{Pr}_{k,m}\left[m = m' \cap Enc_k(m) = c\right]}{\mathsf{Pr}_{k,m}\left[Enc_k(m) = c\right]}$$

which can be further simplified as

$$\frac{\mathsf{Pr}_{k,m}\left[m = m' \cap Enc_k(m') = c\right]}{\mathsf{Pr}_{k,m}\left[Enc_k(m) = c\right]} = \frac{\mathsf{Pr}_m\left[m = m'\right]\mathsf{Pr}_k\left[Enc_k(m') = c\right]}{\mathsf{Pr}_{k,m}\left[Enc_k(m) = c\right]}$$

The central idea behind the proof is to show that

$$\mathsf{Pr}_{k,m}\left[Enc_k(m) = c\right] = \mathsf{Pr}_k\left[Enc_k(m') = c\right]$$

which establishes the result. Recall that,

$$\mathsf{Pr}_{k,m}\left[Enc_k(m) = c\right] = \sum_{m'' \in \mathcal{M}} \mathsf{Pr}_m\left[m = m''\right]\mathsf{Pr}_k\left[Enc_k(m'') = c\right]$$

The perfect secrecy condition allows us to replace the last term to get:

$$\sum_{m'' \in \mathcal{M}} \mathsf{Pr}_m\left[m = m''\right]\mathsf{Pr}_k\left[Enc_k(m') = c\right]$$

This last term can be moved out of the summation and simplified as:

$$\mathsf{Pr}_k\left[Enc_k(m') = c\right]\sum_{m'' \in \mathcal{M}} \mathsf{Pr}_m\left[m = m''\right] = \mathsf{Pr}_k\left[Enc_k(m') = c\right]$$

Shannon secrecy implies perfect secrecy. Assume that $(\mathcal{M}, \mathcal{K}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is Shannon secret. Consider $m_1, m_2 \in \mathcal{M}$, and any c. Let D be the uniform distribution over $\{m_1, m_2\}$. We show that

$$\mathsf{Pr}_k\left[Enc_k(m_1) = c\right] = \mathsf{Pr}_k\left[Enc_k(m_2) = c\right].$$

The definition of D implies that $\mathsf{Pr}_m\left[m = m_1\right] = \mathsf{Pr}_m\left[m = m_2\right] = \frac{1}{2}$. It therefore follows by Shannon secrecy that

$$\mathsf{Pr}_{k,m}\left[m = m_1 \mid Enc_k(m) = c\right] = \mathsf{Pr}_{k,m}\left[m = m_2 \mid Enc_k(m) = c\right]$$

By the definition of conditional probability,

$$\begin{aligned}\mathsf{Pr}_{k,m}\left[m = m_1 \mid Enc_k(m) = c\right] &= \frac{\mathsf{Pr}_{k,m}\left[m = m_1 \cap Enc_k(m) = c\right]}{\mathsf{Pr}_{k,m}\left[Enc_k(m) = c\right]} \\ &= \frac{\mathsf{Pr}_{m}\left[m = m_1\right]\mathsf{Pr}_{k}\left[Enc_k(m_1) = c\right]}{\mathsf{Pr}_{k,m}\left[Enc_k(m) = c\right]} \\ &= \frac{\frac{1}{2}\cdot\mathsf{Pr}_{k}\left[Enc_k(m_1) = c\right]}{\mathsf{Pr}_{k,m}\left[Enc_k(m) = c\right]}\end{aligned}$$

Analogously,

$$\mathsf{Pr}_{k,m}\left[m = m_2 \mid Enc_k(m) = c\right] = \frac{\frac{1}{2}\cdot\mathsf{Pr}_{k}\left[Enc_k(m_2) = c\right]}{\mathsf{Pr}_{k,m}\left[Enc_k(m) = c\right]}.$$

Cancelling and rearranging terms, we conclude that

$$\mathsf{Pr}_{k}\left[Enc_k(m_1) = c\right] = \mathsf{Pr}_{k}\left[Enc_k(m_2) = c\right].$$

□

1.3.3 The One-Time Pad

Given our definition of security, we turn to the question of whether perfectly secure encryption schemes exists. It turns out that both the encryption schemes we have seen so far (i.e., the Caesar and Substitution ciphers) are secure as long as we only consider messages of length 1. However, when considering messages of length 2 (or more) the schemes are no longer secure—in fact, it is easy to see that encryptions of the strings AA and AB have disjoint distributions, thus violating perfect secrecy (prove this!).

Nevertheless, this suggests that we might obtain perfect secrecy by somehow adapting these schemes to operate on each element of a message independently. This is the intuition behind the *one-time pad* encryption scheme, invented by Gilbert Vernam and Joseph Mauborgne in 1919.

Definition 12.1. The One-Time Pad encryption scheme is described by the following tuple $(\mathcal{M}, \mathcal{K}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

$$\begin{aligned}\mathcal{M} &= \{0,1\}^n \\ \mathcal{K} &= \{0,1\}^n \\ \mathsf{Gen} &= k = k_1k_2\ldots k_n \leftarrow \{0,1\}^n \\ Enc_k(m_1m_2\ldots m_n) &= c_1c_2\ldots c_n \text{ where } c_i = m_i \oplus k_i \\ Dec_k(c_1c_2\ldots c_n) &= m_1m_2\ldots m_n \text{ where } m_i = c_i \oplus k_i\end{aligned}$$

The $\oplus$ operator represents the binary xor operation.

Proposition 12.1. *The One-Time Pad is a perfectly secure private-key encryption scheme.*

Proof. It is straight-forward to verify that the One Time Pad is a private-key encryption scheme. We turn to show that the One-Time Pad is perfectly secret and begin by showing the the following claims.

Claim 13.1. *For any $c, m \in \{0,1\}^n$,*

$$\mathsf{Pr}\left[k \leftarrow \{0,1\}^n : Enc_k(m) = c\right] = 2^{-k}$$

Claim 13.2. *For any $c \notin \{0,1\}^n, m \in \{0,1\}^n$,*

$$\mathsf{Pr}\left[k \leftarrow \{0,1\}^n : Enc_k(m) = c\right] = 0$$

Claim 13.1 follows from the fact that for any $m, c \in \{0,1\}^n$ there is only one k such that $Enc_k(m) = m \oplus k = c$, namely $k = m \oplus c$. Claim 13.2 follows from the fact that for every $k \in \{0,1\}^n$ $Enc_k(m) = m \oplus k \in \{0,1\}^n$.

From the claims we conclude that for any $m_1, m_2 \in \{0,1\}^n$ and every c it holds that

$$\mathsf{Pr}\left[k \leftarrow \{0,1\}^n : Enc_k(m_1) = c\right] = \mathsf{Pr}\left[k \leftarrow \{0,1\}^n : Enc_k(m_2) = c\right]$$

which concludes the proof. □

So perfect secrecy is obtainable. But at what cost? When Alice and Bob meet to generate a key, they must generate one that is as long as all the messages they will send until the next time they meet. Unfortunately, this is not a consequence of the design of the One-Time Pad, but rather of perfect secrecy, as demonstrated by Shannon's famous theorem.

1.3.4 Shannon's Theorem

Theorem 13.1 (Shannon)**.** *If an encryption scheme $(\mathcal{M}, \mathcal{K}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is perfectly secret, then $|\mathcal{K}| \geq |\mathcal{M}|$.*

Proof. Assume there exists a perfectly secret $(\mathcal{M}, \mathcal{K}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ such that $|\mathcal{K}| < |\mathcal{M}|$. Take any $m_1 \in \mathcal{M}$, $k \in \mathcal{K}$, and let $c \leftarrow Enc_k(m_1)$. Let $\mathbf{Dec}(c)$ denote the set $\{m \mid \exists k \in \mathcal{K} \;.\; m = Dec_k(c)\}$ of all possible decryptions of c under all possible keys. As Dec is deterministic, this set has size at most $|\mathcal{K}|$. But since $|\mathcal{M}| > |\mathcal{K}|$, there exists some message m_2 not in $\mathbf{Dec}(c)$. By the definition of a private encryption scheme it follows that

$$\mathsf{Pr}\left[k \leftarrow \mathcal{K} : Enc_k(m_2) = c\right] = 0$$

But since

$$\Pr[k \leftarrow \mathcal{K} : Enc_k(m_1) = c] > 0$$

we conclude that

$$\Pr[k \leftarrow \mathcal{K} : Enc_k(m_1) = c] \neq \Pr[k \leftarrow \mathcal{K} : Enc_k(m_2) = c] > 0$$

which contradicts the perfect secrecy of $(\mathcal{M}, \mathcal{K}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. □

Note that the proof of Shannon's theorem in fact describes an *attack* on every private-key encryption scheme $(\mathcal{M}, \mathcal{K}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ such that $|\mathcal{K}| < |\mathcal{M}|$. In fact, it follows that for any such encryption scheme there exists $m_1, m_2 \in M$ and a constant $\epsilon > 0$ such that

$$\Pr[k \leftarrow \mathcal{K}; Enc_k(m_1) = c : m_1 \in \mathbf{Dec}(c)] = 1$$

but

$$\Pr[k \leftarrow \mathcal{K}; Enc_k(m_1) = c : m_2 \in \mathbf{Dec}(c)] \leq 1 - \epsilon$$

The first equation follows directly from the definition of private-key encryption, whereas the second equation follows from the fact that (by the proof of Shannon's theorem) there exists some c in the domain of $Enc.(m_1)$ such that $m_2 \notin \mathbf{Dec}(c)$. Consider, now, a scenario where Alice uniformly picks a message m from $\{m_1, m_2\}$ and sends the encryption of m to Bob. We claim that Eve, having seen the encryption c of m can guess whether $m = m_1$ or $m = m_2$ with probability higher than $\frac{1}{2}$. Eve, upon receiving c simply checks if $m_2 \in \mathbf{Dec}(c)$. If $m_2 \notin \mathbf{Dec}(c)$, Eve guesses that $m = m_1$, otherwise she makes a random guess. If Alice sent the message $m = m_2$ then $m_2 \in \mathbf{Dec}(c)$ and Eve will guess correctly with probability $\frac{1}{2}$. If, on the other hand, Alice sent $m = m_1$, then with probability ϵ, $m_2 \notin \mathbf{Dec}(c)$ and Eve will guess correctly with probability 1, whereas with probability $1 - \epsilon$ Eve will make a random guess, and thus will be correct with probability $\frac{1}{2}$. We conclude that Eve's success probability is

$$\frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot (\epsilon \cdot 1 + (1 - \epsilon) \cdot \frac{1}{2}) = \frac{1}{2} + \frac{\epsilon}{2}$$

Thus we have exhibited a very *concise* attack for Eve, which makes it possible for her to guess what message Alice sends with probability better than $\frac{1}{2}$.

A possible critique against our argument is that if ϵ is very small (e.g., 2^{-100}), then the utility of this attack is limited. However, as we shall see in the following stonger version of Shannon's theorem which states that if $\mathcal{M} = \{0,1\}^n$ and $\mathcal{K} = \{0,1\}^{n-1}$ (i.e., if the key is simply one bit shorter than the message), then $\epsilon = \frac{1}{2}$.

Theorem 14.1. *Let* $(\mathcal{M}, \mathcal{K}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a private-key encryption scheme where* $\mathcal{M} = \{0,1\}^n$ *and* $\mathcal{K} = \{0,1\}^{n-1}$*. Then, there exist messages* $m_0, m_1 \in \mathcal{M}$ *such that*

$$\Pr\left[k \leftarrow \mathcal{K}; Enc_k(m_1) = c : m_2 \in \mathbf{Dec}(c)\right] \leq \frac{1}{2}$$

Proof. Given $c \leftarrow Enc_k(m)$ for some key $k \in \mathcal{K}$ and message $m \in \mathcal{M}$, consider the set $\mathbf{Dec}(c)$. Since Dec is deterministic it follows that $|\mathbf{Dec}(c)| \leq |\mathcal{K}| = 2^{n-1}$. Thus, for all $m_1 \in \mathcal{M}, k \leftarrow \mathcal{K}$,

$$\Pr\left[m' \leftarrow \{0,1\}^n; c \leftarrow Enc_k(m_1) : m' \in \mathbf{Dec}(c)\right] \leq \frac{2^{n-1}}{2^n} = \frac{1}{2}$$

Since the above probability is bounded by $\frac{1}{2}$ for *every* key $k \in \mathcal{K}$, this must also hold for a random $k \leftarrow Gen$.

$$\Pr\left[m' \leftarrow \{0,1\}^n; k \leftarrow \mathsf{Gen}; c \leftarrow Enc_k(m_1) : m' \in \mathbf{Dec}(c)\right] \leq \frac{1}{2} \tag{1.1}$$

Additionally, since the bound holds for a random message m', there must exist some *particular* message m_2 that minimizes the probability. In other words, for every message $m_1 \in \mathcal{M}$, there exists some message $m_2 \in \mathcal{M}$ such that

$$\Pr\left[k \leftarrow \mathsf{Gen}; c \leftarrow Enc_k(m_1) : m_2 \in \mathbf{Dec}(c)\right] \leq \frac{1}{2}$$

□

Remark 15.1. Note that the theorem is stronger than stated. In fact, we showed that for *every* $m_1 \in \mathcal{M}$, there exists some string m_2 that satisfies the desired condition. We also mention that if we content ourselves with getting a bound of $\epsilon = \frac{1}{4}$, the above proof actually shows that for *every* $m_1 \in \mathcal{M}$, it holds that for at least *one fourth* of the messages $m_2 \in \mathcal{M}$,

$$\Pr\left[k \leftarrow \mathcal{K}; Enc_k(m_1) = c : m_2 \in \mathbf{Dec}(c)\right] \leq \frac{1}{4};$$

otherwise we would contradict equation (1.1).

Thus, by relying on Theorem 14.1, we conclude that if the key length is only one bit shorter than the message message length, there exist messages m_1, m_2 such that Eve's success probability is $\frac{1}{2} + \frac{1}{4} = \frac{3}{4}$ (or alternatively, there exist *many* messages m_1, m_2 such that Eve's success probability is at least $\frac{1}{2} + \frac{1}{8} = \frac{5}{8}$). This is clearly not acceptable in most applications of an encryption scheme! So, does this mean that to get any "reasonable" amount of security Alice and Bob must share a long key?

Note that although Eve's attack only takes a few lines of code to describe, its running-time is high. In fact, to perform her attack—which amounts to checking whether $m_2 \in \mathbf{Dec}(c)$)—Eve must try all possible keys $k \in \mathcal{K}$ to check whether c possibly could decrypt to m_2. If, for instance, $\mathcal{K} = \{0,1\}^n$, this requires her to perform 2^n (i.e., exponentially many) different decryptions! Thus, although the attack can be simply described, it is not "feasible" by any *efficient* computing device. This motivates us to consider only "feasible" adversaries—namely adversaries that are *computationally bounded*. Indeed, as we shall see later in the course, with respect to such adversaries, the implications of Shannon's Theorem can be overcome.

1.4 Overview of the Course

In this course we will focus on some of the key *concepts* and *techniques* in modern cryptography. The course will be structured around the following notions:

Computational Hardness and One-way Functions. As illustrated above, to circumvent Shannon's lower bound we have to restrict our attention to computationally bounded adversaries. The first part of the course deals with notions of resource-bounded (and in particular *time-bounded*) computation, computational hardness, and the notion of one-way functions. One-way functions—i.e., functions that are "easy" to compute, but "hard" to invert (for computationally-bounded entities)—are at the heart of most modern cryptographic protocols.

Indistinguishability. The notion of (computational) indistinguishability formalizes what it means for a computationally-bounded adversary to be unable to "tell apart" two distributions. This notion is central to modern definitions of security for encryption schemes, but also for formally defining notions such as pseudo-random generation, bit commitment schemes, and much more.

Knowledge. A central desideratum in the design of cryptographic protocols is to ensure that the protocol execution *does not leak more "knowledge" than what is necessary*. In this part of the course, we provide and investigate "knowledge-based" (or rather *zero knowledge*-based) definitions of security.

Authentication. Notions such as *digital signatures* and *messages authentication codes* are digital analogues of traditional written signatures. We explore different notions of authentication and show how cryptographic techniques can be used to obtain new types of authentication mechanism not achievable by traditional written signatures.

Computing on Secret Inputs. Finally, we consider protocols which allow mutually distrustful parties to perform arbitrary computation on their respective (potentially secret) inputs. This includes *secret-sharing* protocols and *secure two-party (or multi-party) computation* protocols. We have described the later earlier in this chapter; secret-sharing protocols are methods which allow a set of n parties to receive "shares" of a secret with the property that any "small" subset of shares leaks no information about the secret, but once an appropriate number of shares are collected the whole secret can be recovered.

Composability. It turns out that cryptographic schemes that are secure when executed in isolation can be completely compromised if many instances of the scheme are simultaneously executed (as is unavoidable when executing cryptographic protocols in modern networks). The question of *composability* deals with issues of this type.

Chapter 2

Computational Hardness

2.1 Efficient Computation and Efficient Adversaries

We start by formalizing what it means to compute a function.

Definition 19.1 (Algorithm)**.** An *algorithm* is a deterministic Turing machine whose input and output are strings over some alphabet Σ. We usually have $\Sigma = \{0, 1\}$.

Definition 19.2 (Running-time of Algorithms)**.** An algorithm $\mathcal{A}$ runs in time $T(n)$ if for all $x \in B^*$, $\mathcal{A}(x)$ halts within $T(|x|)$ steps. $\mathcal{A}$ runs in *polynomial time* (or is an *efficient* algorithm) if there exists a constant c such that $\mathcal{A}$ runs in time $T(n) = n^c$.

Definition 19.3 (Deterministic Computation)**.** Algorithm $\mathcal{A}$ is said to *compute* a function $f : \{0, 1\}^* \to \{0, 1\}^*$ if $\mathcal{A}$, on input x, outputs $f(x)$, for all $x \in B^*$.

It is possible to argue with the choice of polynomial-time as a cutoff for "efficiency", and indeed if the polynomial involved is large, computation may not be efficient in practice. There are, however, strong arguments to use the polynomial-time definition of efficiency:

1. This definition is independent of the representation of the algorithm (whether it is given as a Turing machine, a C program, etc.) as converting from one representation to another only affects the running time by a polynomial factor.

2. This definition is also closed under composition, which is desirable as it simplifies reasoning.

3. "Usually", polynomial time algorithms do turn out to be efficient ('polynomial" almost always means "cubic time or better")

4. Most "natural" functions that are not known to be polynomial-time computable require *much* more time to compute, so the separation we propose appears to have solid natural motivation.

Remark: Note that our treatment of computation is an *asymptotic* one. In practice, actual running time needs to be considered carefully, as do other "hidden" factors such as the size of the description of $\mathcal{A}$. Thus, we will need to instantiate our formulae with numerical values that make sense in practice.

2.1.1 Some computationally "hard" problems

Many commonly encountered functions are computable by efficient algorithms. However, there are also functions which are known or believed to be hard.

Halting: The famous Halting problem is an example of an *uncomputable* problem: Given a description of a Turing machine M, determine whether or not M halts when run on the empty input.

Time-hierarchy: There exist functions $f : \{0,1\}^* \to \{0,1\}$ that are computable, but are not computable in polynomial time (their existence is guaranteed by the Time Hierarchy Theorem in Complexity theory).

Satisfiability: The famous **SAT** problem is to determine whether a Boolean formula has a satisfying assignment. **SAT** is *conjectured* not to be polynomial-time computable—this is the famous conjecture that $\mathbf{P} \neq \mathbf{NP}$.[1]

2.1.2 Randomized Computation

A natural extension of deterministic computation is to allow an algorithm to have access to a source of random coins tosses. Allowing this extra freedom is certainly plausible (as it is easy to generate such random coins in practice), and it is believed to enable more efficient algorithms for computing certain tasks. Moreover, it will be necessary for the security of the schemes that we present later. For example, as we discussed in chapter one, Kerckhoff's principle states that all algorithms in a scheme should be public. Thus, if the private key generation algorithm Gen did not use random coins, then Eve

[1]See Appendix B for definitions of **P** and **NP**.

would be able to compute the same key that Alice and Bob compute. Thus, to allow for this extra reasonable (and necessary) resource, we extend the above definitions of computation as follows.

Definition 21.0 (Randomized Algorithms - Informal)**.** A *randomized algorithm* is a Turing machine equipped with an extra random tape. Each bit of the random tape is uniformly and independently chosen.

Equivalently, a randomized algorithm is a Turing Machine that has access to a "magic" randomization box (or oracle) that output a truly random bit on demand.

To define efficiency we must clarify the concept of *running time* for a randomized algorithm. There is a subtlety that arises here, as the actual run time may depend on the bit-string obtained from the random tape. We take a conservative approach and define the running time as the upper bound over all possible random sequences.

Definition 21.1 (Running-time of Randomized Algorithms)**.** A randomized Turing machine $\mathcal{A}$ runs in time $T(n)$ if for all $x \in B^*$, $\mathcal{A}(x)$ halts within $T(|x|)$ steps (independent of the content of $\mathcal{A}$'s random tape). $\mathcal{A}$ runs in *polynomial time* (or is an *efficient* randomized algorithm) if there exists a constant c such that $\mathcal{A}$ runs in time $T(n) = n^c$.

Finally, we must also extend our definition of computation to randomized algorithm. In particular, once an algorithm has a random tape, its output becomes a distribution over some set. In the case of deterministic computation, the output is a singleton set, and this is what we require here as well.

Definition 21.2. Algorithm $\mathcal{A}$ is said to *compute* a function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ if $\mathcal{A}$, on input x, outputs $f(x)$ with probability 1 for all $x \in B^*$. The probability is taken over the random tape of $\mathcal{A}$.

Thus, with randomized algorithms, we tolerate algorithms that on *rare* occasion make errors. Formally, this is a necessary relaxation because some of the algorithms that we use (e.g., primality testing) behave in such a way. In the rest of the book, however, we ignore this rare case and assume that a randomized algorithm always works correctly.

On a side note, it is worthwhile to note that a polynomial-time randomized algorithm $\mathcal{A}$ that computes a function with probability $\frac{1}{2} + \frac{1}{poly(n)}$ can be used to obtain another polynomial-time randomized machine $\mathcal{A}'$ that computes the function with probability $1 - 2^{-n}$. ($\mathcal{A}'$ simply takes multiple runs of $\mathcal{A}$ and finally outputs the most frequent output of $\mathcal{A}$. The Chernoff bound (see Chapter A) can then be used to analyze the probability with which such a "majority" rule works.)

Polynomial-time randomized algorithms will be the principal model of efficient computation considered in this course. We will refer to this class of algorithms as *probabilistic polynomial-time Turing machine* (*p.p.t.*) or *efficient randomized algorithm* interchangeably.

Given the above notation we can define the notion of an efficient encryption scheme:

Definition 22.1 (Efficient Private-key Encryption). A triplet of algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is called an *efficient private-key encryption scheme* if the following holds:

1. Gen is a p.p.t such that for every $n \in N$, $k \leftarrow \mathsf{Gen}(1^n)$.
2. $c \leftarrow \mathsf{Enc}_k(m)$ is a p.p.t. algorithm that given k and $m \in \{0,1\}^*$ produces a ciphertext c.
3. $m \leftarrow \mathsf{Dec}_k(c)$ is a p.p.t. algorithm that given a ciphertext c and key k produces a message $m \in \{0,1\}^* \cup \perp$.
4. For all $n \in N$, $m \in \{0,1\}^n$,

$$\Pr\left[k \leftarrow \mathsf{Gen}(1^n) : \mathsf{Dec}_k(\mathsf{Enc}_k(m)) = m\right] = 1$$

In the sequel, when discussing encryption schemes we always refer to efficient encryption schemes. As a departure from our notation in the first chapter, here no longer refer to a message space $\mathcal{M}$ or a key space $\mathcal{K}$ because we assume that both are bit strings. In particular, on security parameter 1^n, our definition requires a scheme to handle messages of length n bits. It is also possible, and perhaps more simple, to define an encryption scheme that only works on a single-bit message space $\mathcal{M} = \{0,1\}$ for every security parameter.

2.1.3 Efficient Adversaries.

When modeling adversaries, we use a more relaxed notion of efficient computation. In particular, instead of requiring the adversary to be a machine with constant-sized description, we allow the size of the adversary's program to increase (polynomially) with the input length. As before, we still allow the adversary to use random coins and require that the adversary's running time is bounded by a polynomial. The primary motivation for using non-uniformity to model the adversary is to simplify definitions and proofs.

Definition 22.2 (Non-uniform p.p.t. machine)**.** A non-uniform p.p.t. machine A is a sequence of probabilistic machines $A = \{A_1, A_2, \ldots\}$ for which there exists a polynomial d such that the description size of $|A_i| < d(i)$ and the running time of A_i is also less than $d(i)$. We write $A(x)$ to denote the distribution obtained by running $A_{|x|}(x)$.

Alternatively, a non-uniform p.p.t. machine can also be defined as a *uniform* p.p.t. machine A that receives an advice string for each input length.

2.2 One-Way Functions

At a high level, there are two basic desiderata for any encryption scheme:

- it must be feasible to generate c given m and k, but
- it must be hard to recover m and k given c.

This suggests that we require functions that are easy to compute but hard to invert—*one-way functions*. Indeed, these turn out to be the most basic building block in cryptography.

There are several ways that the notion of one-wayness can be defined formally. We start with a definition that formalizes our intuition in the simplest way.

Definition 23.1 (Worst-case One-way Function). A function $f : \{0,1\}^* \rightarrow \{0,1\}$ is (worst-case) *one-way* if:

1. there exists a p.p.t. machine $\mathcal{C}$ that computes $f(x)$, and
2. there is no non-uniform p.p.t. machine $\mathcal{A}$ such that

$$\forall x \ \Pr[\mathcal{A}(f(x)) \in f^{-1}(f(x))] = 1$$

It can be show that assuming **NP** $\notin$ **BPP**, one-way functions according to the above definition must exist.[2] In fact, these two assumptions are equivalent (show this!). Note, however, that this definition allows for certain pathological functions—e.g., those where inverting the function for *most* x values is easy, as long as every machine fails to invert $f(x)$ for infinitely many x's. It is an open question whether such functions can still be used for good encryption schemes. This observation motivates us to refine our requirements. We want functions where for a randomly chosen x, the probability that we are able to invert the function is very small. With this new definition in mind, we begin by formalizing the notion of *very small*.

[2]See Appendix B for definitions of **NP** and **BPP**.

Definition 23.2 (Negligible function)**.** A function $\varepsilon(n)$ is *negligible* if for every c, there exists some n_0 such that for all $n_0 < n$, $\epsilon(n) \leq \frac{1}{n^c}$. Intuitively, a negligible function is asymptotically smaller than the inverse of any fixed polynomial.

We say that a function $t(n)$ is *non-negligible* if there exists some constant c such that for infinitely many points $\{n_0, n_1, \ldots\}$, $t(n_i) > n_i^c$. This notion becomes important in proofs that work by contradiction.

We are now ready to present a more satisfactory definition of a one-way function.

Definition 24.1 (Strong one-way function)**.** A function $f : \{0,1\}^* \to \{0,1\}^*$ is *strongly one-way* if it satisfies the following two conditions.

1. **Easy to compute.** There is a p.p.t. machine $\mathcal{C} : \{0,1\}^* \to \{0,1\}^*$ that computes $f(x)$ on all inputs $x \in \{0,1\}^*$.

2. **Hard to invert.** Any efficient attempt to invert f on random input will succeed with only negligible probability. Formally, for any non-uniform p.p.t. machines $\mathcal{A} : \{0,1\}^* \to \{0,1\}^*$, there exists a negligible function ϵ such that for any input length $n \in \mathbb{N}$,

$$\Pr\left[x \leftarrow \{0,1\}^n;\ y = f(x);\ \mathcal{A}(1^n, y) = x' : f(x') = y\right] \leq \epsilon(n).$$

Remark:

1. The algorithm $\mathcal{A}$ receives the additional input of 1^n; this is to allow $\mathcal{A}$ to take time polynomial in $|x|$, even if the function f should be substantially length-shrinking. In essence, we are ruling out some pathological cases where functions might be considered one-way because writing down the output of the inversion algorithm violates its time bound.

2. As before, we must keep in mind that the above definition is *asymptotic*. To define one-way functions with concrete security, we would instead use explicit parameters that can be instantiated as desired. In such a treatment we say that a function is (t, s, ϵ)-one-way, if no $\mathcal{A}$ of size s with running time $\leq t$ will succeed with probability better than ϵ.

2.3 Multiplication, Primes, and the Factoring Assumption

A first candidate for a one-way function is the function $f_{\mathsf{mult}} : \mathbb{N}^2 \to \mathbb{N}$ defined by

$$f_{\mathsf{mult}}(x, y) = \begin{cases} 1 & \text{if } x = 1 \vee y = 1 \\ xy & \text{otherwise} \end{cases}$$

Is this a one-way function? Clearly, by the multiplication algorithm, f_{mult} is easy to compute. But f_{mult} is not always hard to invert! If at least one of x and y is even, then their product will be even as well. This happens with probability $\frac{3}{4}$ if the input (x, y) is picked uniformly at random from $\mathbb{N}^2$. So the following attack A will succeed with probability $\frac{3}{4}$:

$$A(z) = \begin{cases} (2, \frac{z}{2}) & \text{if } z \text{ even} \\ (0, 0) & \text{otherwise.} \end{cases}$$

Something is not quite right here, since f_{mult} is conjectured to be hard to invert on *some*, but not all, inputs[3] . Our current definition of a one-way function is too restrictive to capture this notion, so we will define a weaker variant that relaxes the hardness condition on inverting the function. This weaker version only requires that all efficient attempts at inverting will fail with some non-negligible probability.

Although, f_{mult} is not always hard to invert, it is conjectured to be hard to invert on *some* (but not all) inputs.

2.3.1 The Factoring Assumption

Let us denote the (finite) set of primes that are smaller than 2^n as

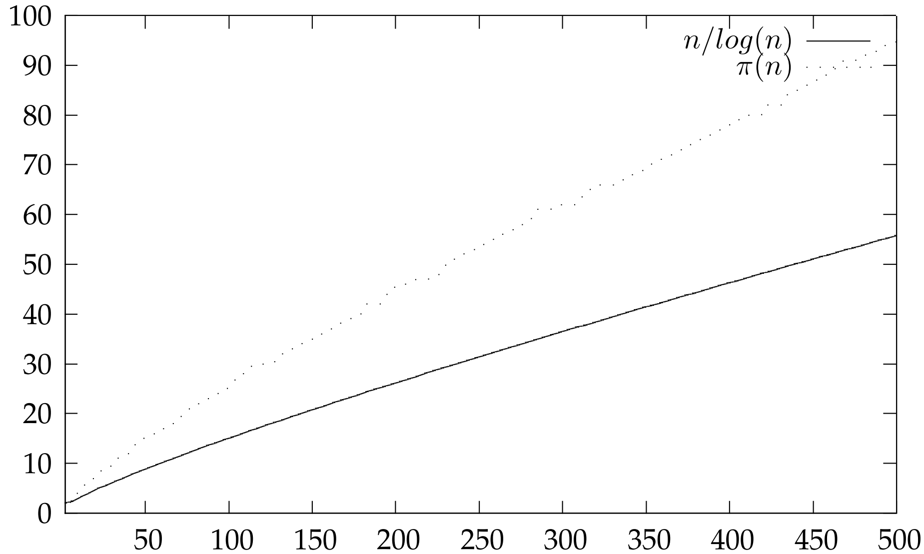
$$\Pi_n = \{q \mid q < 2^n \text{ and } q \text{ is prime}\}$$

Consider the following assumption, which we shall assume for the remainder of these notes:

Conjecture 25.1 (Factoring Assumption)**.** *For every non-uniform p.p.t. algorithm $\mathcal{A}$, there exists a negligible function ϵ such that*

$$\textsf{Pr}\left[p \xleftarrow{r} \Pi_n; q \leftarrow \Pi_n; N = pq : \mathcal{A}(N) \in \{p, q\}\right] < \epsilon(n)$$

[3]Notice that by the way we have defined f_{mult}, $(1, xy)$ will never be a pre-image of xy. That is why some instances might be hard to invert.



The factoring assumption is a very important, well-studied conjecture that is widely believed to be true. The best provable algorithm for factorization runs in time $2^{O((n \log n)^{1/2})}$, and the best heuristic algorithm runs in time $2^{O(n^{1/3} \log^{2/3} n)}$. Factoring is hard in a concrete way as well: at the time of this writing, researchers have been able to factor a 663 bit numbers using 80 machines and several months.

2.3.2 There are many primes

The problem of characterizing the set of prime numbers has been considered since antiquity. Early on, it was noted that there are an infinite number of primes. However, merely having an infinite number of them is not reassuring, since perhaps they are distributed in such a haphazard way as to make finding them extremely difficult. An empirical way to approach the problem is to define the function

$$\pi(x) = \text{number of primes} \leq x$$

and graph it for reasonable values of x:

By empirically fitting this curve, one might guess that $\pi(x) \approx x/\log x$. In fact, at age 15, Gauss made exactly this conjecture. Since then, many people have answered the question with increasing precision; notable are Chebyshev's theorem (upon which our argument below is based), and the famous *Prime Number Theorem* which establishes that $\pi(N)$ approaches $\frac{N}{\ln N}$ as N grows to infinity. Here, we will prove a much simpler theorem which only lower-bounds $\pi(x)$:

Theorem 27.1 (Chebyshev). *For $x > 1$, $\pi(x) > \frac{x}{2\log x}$*

Proof. Consider the value

$$X = \frac{2x!}{(x!)^2} = \left(\frac{x+x}{x}\right)\left(\frac{x+(x-1)}{(x-1)}\right)\cdots\left(\frac{x+2}{2}\right)\left(\frac{x+1}{1}\right)$$

Observe that $X > 2^x$ (since each term is greater than 2) and that the largest prime dividing X is at most $2x$ (since the largest numerator in the product is $2x$). By these facts and unique factorization, we can write

$$X = \prod_{p<2x} p^{\nu_p(X)} > 2^x$$

where the product is over primes p less than $2x$ and $\nu_p(X)$ denotes the integral power of p in the factorization of X. Taking logs on both sides, we have

$$\sum_{p<2x} \nu_p(X)\log p > x$$

We now employ the following claim proven below.

Claim 27.1. $\nu_p(X) < \frac{\log 2x}{\log p}$

Substituting this claim, we have

$$\sum_{p<2x}\left(\frac{\log 2x}{\log p}\right)\log p = \log 2x \left(\sum_{p<2x} 1\right) > x$$

Notice that the second sum on the left hand side is precisely $\pi(2x)$; thus

$$\pi(2x) > \frac{x}{\log 2x} = \frac{1}{2}\cdot\frac{2x}{\log 2x}$$

which establishes the theorem for even values. For odd values, notice that

$$\pi(2x) = \pi(2x-1) > \frac{2x}{2\log 2x} > \frac{(2x-1)}{2\log(2x-1)}$$

since $x/\log x$ is an increasing function for $x \geq 3$.

Proof Of Claim 27.1. Notice that

$$\nu_p(X) = \sum_{i>1} \left(\lfloor 2x/p^i \rfloor - 2\lfloor x/p^i \rfloor \right) < \log 2x / \log p$$

The first equality follows because the product $2x! = (2x)(2x-1)\ldots(1)$ includes a multiple of p^i at most $\lfloor 2x/p^i \rfloor$ times in the numerator of X; similarly the product $x! \cdot m!$ in the denominator of X *removes* it exactly $2\lfloor x/p^i \rfloor$ times. The second inequality follows because each term in the summation is at most 1 and after $p^i > 2x$, all of the terms will be zero. □

□

An important corollary of Chebyshev's theorem is that at least a fraction $\frac{1}{2n}$ of n-bit numbers are prime. As we shall see in Section 2.7.5, primality testing can be done in polynomial time—i.e., we can efficiently check whether a number is prime or not.

2.4 Weak One-way Functions

To capture the "one-wayness" of f_{mult}, we define a weaker variant of one-wayness. This relaxed version only requires that all efficient attempts at inverting will fail with some non-negligible probability.

Definition 28.1 (Weak one-way function)**.** A function $f : \{0,1\}^* \to \{0,1\}^*$ is *weakly one-way* if it satisfies the following two conditions.

1. **Easy to compute.** (Same as that for a strong one-way function.) There is a p.p.t. machine $\mathcal{C} : \{0,1\}^* \to \{0,1\}^*$ that computes $f(x)$ on all inputs $x \in \{0,1\}^*$.

2. **Hard to invert.** There exists a polynomial function $q : \mathbb{N} \to \mathbb{N}$ such that for any non-uniform p.p.t. machine $\mathcal{A} : \{0,1\}^* \to \{0,1\}^*$, for sufficiently large $n \in N$,

$$\Pr\left[x \leftarrow \{0,1\}^n;\ y = f(x);\ \mathcal{A}(1^n, y) = x' : f(x') = y\right] \leq 1 - \frac{1}{q(n)}$$

Given this definition, we can show that, under the factoring assumption, f_{mult} is a weak one-way function.

Theorem 28.1. *Assume the factoring assumption. Then f_{mult} is a weak one-way function.*

Proof. As already mentioned, $f_{mult}(x, y)$ is clearly computable in polynomial time; we just need to show that it is hard to invert.

Consider a certain input length $2n$ (i.e,$|x| = |y| = n$). Intuitively, by Chebyshev's theorem, with probability $\frac{1}{4n^2}$ a random input pair x, y will consists of two primes; in this case, by the factoring assumption, the function should be hard to invert (except with negligible probability).

We proceed to a formal proof. Let $q(n) = 8n^2$; we show that no n.u. p.p.t. can invert f_{mult} w.p higher than $1 - \frac{1}{q(n)}$ for sufficiently large input lengths n. Assume, for contradiction, that there exists a n.u. p.p.t A that inverts f_{mult} w.p. at least $1 - \frac{1}{q(n)}$ for infinitely many $n \in N$. That is,

$$\Pr\left[x, y \leftarrow \{0,1\}^n, z = xy : A(1^{2n}, z) \in \{x, y\}\right] \geq 1 - \frac{1}{8n^2} \tag{2.1}$$

We construct a n.u. p.p.t machine A' which uses A to break the factoring assumption.

Algorithm 1: $A'(z)$: Breaking the factoring assumption

1: Sample $x, y \leftarrow \{0,1\}^n$
2: **if** x and y are both prime **then**
3: $\quad z' \leftarrow z$
4: **else**
5: $\quad z' \leftarrow xy$
6: **end if**
7: $w \leftarrow A(1^n, z')$.
8: Return w if x and y are both prime.

Note that since primality testing can be done in polynomial time, and since A is a non-uniform p.p.t., A' is also a n.u. p.p.t. Suppose we now feed A' the product of a pair of random n-bit primes, z. In order to give A a uniformly distributed input (i.e. the product of a pair of random n-bit numbers), A' samples x, y uniformly, and replace the product xy with the input z iff both x and y are prime. From (2.1), A fails to factor its input with probability at most $\frac{1}{8n^2}$; by Chebychev's Theorem, A' fails to pass z to A with probability at most $1 - \frac{1}{4n^2}$. Using the union bound, we conclude that A' fails with probability at most

$$\left(1 - \frac{1}{4n^2}\right) + \frac{1}{8n^2} \leq 1 - \frac{1}{8n^2}$$

for large n. In other words, A' factors z with probability at least $\frac{1}{8n^2}$ for infinitely many n. This means that there does not exists a negligible function that bounds the success probability of A', which contradicts the factoring assumption. $\square$

Remark 29.1. Note that in the above proof we relied on the fact that primality testing can be done in polynomial time. This was done only for ease of exposition, and is, in fact, not necessary: simply note that the machine A'', that proceeds just as A', but always lets $z = z'$ and always outputs w, must succeed with even higher probability. But A'' then never needs to check if x and y are prime.

2.5 Hardness Amplification

As we show below, the existence of weak one-way functions is equivalent to the existence of (strong) one-way functions. To show this, we present an efficient transformation from any weak one-way function to a strong one. The main insight is that running a weak one-way function f on enough random inputs x_i produces a list of elements y_i which contains at least one member that is hard to invert for f.

Theorem 30.1. *For any weak one-way function $f : \{0,1\}^* \to \{0,1\}^*$, there is a polynomial $m(\cdot)$, such that the following function $f' : (\{0,1\}^n)^{q(n)} \to (\{0,1\}^*)^q(n)$ is strongly one-way:*

$$f'(x_1, x_2, \ldots, x_{m(n)}) = (f(x_1), f(x_2), \ldots, f(x_{m(n)})).$$

We prove this theorem by contradiction. We assume that f' is not strongly one-way so that there is an algorithm $\mathcal{A}'$ that inverts it with non-negligible probability. From this, we construct an algorithm $\mathcal{A}$ that inverts f with high probability. The complete proof is found in §2.5.1 below. Here we first prove the hardness amplification theorem for the function f_{mult} because it is significantly simpler than the proof for the general case, and it is similar to the proof of Theorem 28.1.

Proposition 30.1. *Assume the factoring assumption and let $m(n) = 4n^3$. Then $f' : (\{0,1\}^n)^{m(n)} \to (\{0,1\}^*)^m(n)$ is strongly one-way:*

$$f'(x_1, x_2, \ldots, x_{m(n)}) = (f_{\mathit{mult}}(x_1), f_{\mathit{mult}}(x_2), \ldots, f_{\mathit{mult}}(x_{m(n)}))$$

Proof. Recall that by Chebyschev's Theorem, a pair of random n-bit numbers are both prime with probability at least $\frac{1}{4n^2}$. So, if we choose $m = 4n^3$ pairs, the probability that none of them is a prime pair is at most

$$\left(1 - \frac{1}{4n^2}\right)^{4n^3} = \left(1 - \frac{1}{4n^2}\right)^{4n^2 n} \leq e^{-n} \tag{2.2}$$

Thus, intuitively, by the factoring assumption f' is strongly one-way. More formally, suppose the contradiction that f' is not a strong OWF. That is, there exists a n.u. p.p.t machine A and a polynomial p s.t.:

$$\Pr[\vec{x}, \vec{y} \leftarrow \{0,1\}^{nk} : A(1^{2nk}, g(\vec{x}, \vec{y})) \in g^{-1}(\vec{x}, \vec{y})] \geq \frac{1}{p(2nm)} \tag{2.3}$$

We construct a n.u. p.p.t A' which uses A to break the factoring assumption.

Algorithm 2: $A'(z_0)$: Breaking the factoring assumption

```
Sample x⃗, y⃗ ← {0,1}^{nm}
z⃗ ← g(x⃗, y⃗)
if some pair (x_i, y_i) are both prime then
    replace z_i with z_0 (only one z_i if there are many)
end if
x̄_1, ..., x̄_n ← A(1^{2nm}, z⃗)
output x̄_i (or just x̄_1 if there was no pairs of prime (x_i, y_i))
```

Note that since primality testing can be done in polynomial time, and since A is a non-uniform p.p.t., A' is also a n.u. p.p.t. Also note that $A'(z_0)$ feeds A the uniform input distribution by uniformly sampling $(\vec{x}, \vec{y})$ and replacing some product $x_i y_i$ with z_0 only if both x_i and y_i are prime. From (2.3), B fails to factor its inputs with probability at most $1 - 1/p(2nm)$; from (2.2), B' fails to substitute in z_0 with probability at most e^{-n} By the union bound, we conclude that A' fails to factor z_0 with probability at most

$$1 - \frac{1}{p(2nm)} + e^{-n} \leq 1 - \frac{1}{2p(2nm)}$$

for large n. In other words, A' factors z_0 with probability at least $1/(2p(2nm))$ for infinitely many n. This contradicts the factoring assumption. □

Remark 31.1. We note that just as in the proof of Theorem 28.1 the above proof can be modified to not make use of the fact that primality testing can be done in polynomial time. We leave this as an exercise to the reader.

2.5.1 *Proof of Theorem 30.1

Proof. Since f is weakly one-way, let $q : \mathbb{N} \to \mathbb{N}$ be a polynomial such that for any non-uniform p.p.t. algorithm $\mathcal{A}$ and any input length $n \in \mathbb{N}$,

$$\Pr\left[x \leftarrow \{0,1\}^n;\ y = f(x);\ \mathcal{A}(1^n, y) = x' : f(x') = y\right] \leq 1 - \frac{1}{q(n)}.$$

We want to let m such that $\left(1 - \frac{1}{q(n)}\right)^m$ tends to 0 for large n. Since

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

we pick $m = 2nq(n)$.

Assume that f' as defined in the theorem statement is not strongly one-way. Then let $\mathcal{A}'$ be a non-uniform p.p.t. algorithm and $p' : \mathbb{N} \to \mathbb{N}$ be a polynomial such that for infinitely many input lengths $n \in \mathbb{N}$, $\mathcal{A}'$ inverts f' with probability $p'(n)$. i.e.,

$$\Pr\left[x_i \leftarrow \{0,1\}^n; y_i = f(x_i) : f'(\mathcal{A}'(y_1, y_2, \ldots, y_m)) = (y_1, y_2, \ldots, y_m)\right] > \frac{1}{p'(m)}.$$

Since m is polynomial in n, then the function $p(n) = p'(m) = p'(2nq(n))$ is also a polynomial. Rewriting the above probability, we have

$$\Pr\left[x_i \leftarrow \{0,1\}^n; y_i = f(x_i) : f'(\mathcal{A}'(y_1, \ldots, y_m)) = (y_1, \ldots, y_m)\right] > \frac{1}{p(n)}. \tag{2.4}$$

A first idea for using A to invert f would be to, given a string y, feed A as input $yy...y$ (i.e. $y_i = y$ for all i). But, it is possible that A always fails when the input has the format above, i.e., consists of a string repeated m times (these strings form a very small fraction of all strings of length mn); so this plan will not work. A sligthly better approach would be to feed A the string $y_1 \ldots y_m$ where $y_1 = y$ and $y_{j \neq 1} = f(x_j)$ and $x_j \leftarrow \{0,1\}^n$. Again, this will not work since A could potentially invert only a small fraction of y_1's (but, say, all $y_2, \ldots y_m$'s). As we show below, letting $y_i = y$, where $i \leftarrow [m]$ is a random "position" will, however, work.

More precisely, define the algorithm $\mathcal{A}_0 : \{0,1\}^n \to \{0,1\}^n_\perp$, which will attempt to use $\mathcal{A}'$ to invert f, as per the figure below.

ALGORITHM 32.1: $A_0(f, y)$ WHERE $y \in \{0,1\}^n$

1: Pick a random $i \leftarrow [1, m]$.
2: For all $j \neq i$, pick a random $x_j \leftarrow \{0,1\}^n$, and let $y_j = f(x_j)$.
3: Let $y_i \leftarrow y$.
4: Let $(z_1, z_2, \ldots, z_m) \leftarrow \mathcal{A}'(y_1, y_2, \ldots, y_m)$.
5: If $f(z_i) = y$, then output z_i; otherwise, fail and output $\perp$.

To improve our chances of inverting f, we will run $\mathcal{A}_0$ multiple times. To capture this, define the algorithm $\mathcal{A} : \{0,1\}^n \to \{0,1\}^n_\perp$ to run $\mathcal{A}_0$ with its input $2nm^2p(n)$ times, outputting the first non-$\perp$ result it receives. If all runs of $\mathcal{A}_0$ result in $\perp$, then $\mathcal{A}$ outputs $\perp$ as well.

Given this, call an element $x \in \{0,1\}^n$ "good" if $\mathcal{A}_0$ will successfully invert $f(x)$ with non-negligible probability:

$$\mathsf{Pr}\left[\mathcal{A}_0(f(x)) \neq \perp\right] \geq \frac{1}{2m^2p(n)};$$

otherwise, call x "bad."

Note that the probability that $\mathcal{A}$ fails to invert $f(x)$ on a good x is small:

$$\mathsf{Pr}\left[\mathcal{A}(f(x)) \text{ fails} \mid x \text{ good}\right] \leq \left(1 - \frac{1}{2m^2p(n)}\right)^{2m^2np(n)} \approx e^{-n}.$$

We claim that there are a significant number of good elements—enough for $\mathcal{A}$ to invert f with sufficient probability to contradict the weakly one-way assumption on f. In particular, we claim there are at least $2^n\left(1 - \frac{1}{2q(n)}\right)$ good elements in $\{0,1\}^n$. If this holds, then

$$\begin{aligned}
&\mathsf{Pr}\left[\mathcal{A}(f(x)) \text{ fails}\right] \\
&\quad = \mathsf{Pr}\left[\mathcal{A}(f(x)) \text{ fails} \mid x \text{ good}\right] \cdot \mathsf{Pr}\left[x \text{ good}\right] + \mathsf{Pr}\left[\mathcal{A}(f(x)) \text{ fails} \mid x \text{ bad}\right] \cdot \mathsf{Pr}\left[x \text{ bad}\right] \\
&\quad \leq \mathsf{Pr}\left[\mathcal{A}(f(x)) \text{ fails} \mid x \text{ good}\right] + \mathsf{Pr}\left[x \text{ bad}\right] \\
&\quad \leq \left(1 - \tfrac{1}{2m^2p(n)}\right)^{2m^2np(n)} + \tfrac{1}{2q(n)} \\
&\quad \approx e^{-n} + \tfrac{1}{2q(n)} \\
&\quad < \tfrac{1}{q(n)}.
\end{aligned}$$

This contradicts the assumption that f is $q(n)$-weak.

It remains to be shown that there are at least $2^n\left(1 - \frac{1}{2q(n)}\right)$ good elements in $\{0,1\}^n$. Assume that there are more than $2^n\left(\frac{1}{2q(n)}\right)$ bad elements. We will contradict fact (2.4) that with probability $\frac{1}{p(n)}$, $\mathcal{A}'$ succeeds in inverting $f'(x)$ on a random input x. To do so, we establish an upper bound on the probability by splitting it into two quantities:

$$\begin{aligned}
&\mathsf{Pr}\left[x_i \leftarrow \{0,1\}^n; y_i = f'(x_i) : \mathcal{A}'(\vec{y}) \text{ succeeds}\right] \\
&\quad = \mathsf{Pr}\left[x_i \leftarrow \{0,1\}^n; y_i = f'(x_i) : \mathcal{A}'(\vec{y}) \text{ succeeds and some } x_i \text{ is bad}\right] \\
&\qquad + \mathsf{Pr}\left[x_i \leftarrow \{0,1\}^n; y_i = f'(x_i) : \mathcal{A}'(\vec{y}) \text{ succeeds and all } x_i \text{ are good}\right]
\end{aligned}$$

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

$$\begin{aligned}
&\mathsf{Pr}\left[x_i \leftarrow \{0,1\}^n; y_i = f'(x_i) : \mathcal{A}'(\vec{y}) \text{ succeeds and } x_j \text{ is bad}\right] \\
&\quad \leq \mathsf{Pr}\left[x_i \leftarrow \{0,1\}^n; y_i = f'(x_i) : \mathcal{A}'(\vec{y}) \text{ succeeds} \mid x_j \text{ is bad}\right] \\
&\quad \leq m \cdot \mathsf{Pr}\left[\mathcal{A}_0(f(x_j)) \text{ succeeds} \mid x_j \text{ is bad}\right] \\
&\quad \leq \tfrac{m}{2m^2p(n)} = \tfrac{1}{2mp(n)}.
\end{aligned}$$

So taking a union bound, we have

$$\begin{aligned} &\Pr\left[x_i \leftarrow \{0,1\}^n; y_i = f'(x_i) : \mathcal{A}'(\vec{y}) \text{ succeeds and some } x_i \text{ is bad}\right] \\ &\le \sum_j \Pr\left[x_i \leftarrow \{0,1\}^n; y_i = f'(x_i) : \mathcal{A}'(\vec{y}) \text{ succeeds and } x_j \text{ is bad}\right] \\ &\le \frac{m}{2mp(n)} = \frac{1}{2p(n)}. \end{aligned}$$

Also,

$$\begin{aligned} &\Pr\left[x_i \leftarrow \{0,1\}^n; y_i = f'(x_i) : \mathcal{A}'(\vec{y}) \text{ succeeds and all } x_i \text{ are good}\right] \\ &\le \Pr\left[x_i \leftarrow \{0,1\}^n : \text{all } x_i \text{ are good}\right] \\ &< \left(1 - \frac{1}{2q(n)}\right)^m = \left(1 - \frac{1}{2q(n)}\right)^{2nq(n)} \approx e^{-n}. \end{aligned}$$

Hence, $\Pr\left[x_i \leftarrow \{0,1\}^n; y_i = f'(x_i) : \mathcal{A}'(\vec{y}) \text{ succeeds}\right] < \frac{1}{2p(n)} + e^{-n} < \frac{1}{p(n)}$, thus contradicting (2.4). □

2.6 Collections of One-Way Functions

In the last two sections, we have come to suitable definitions for strong and weak one-way functions. These two definitions are concise and elegant, and can nonetheless be used to construct generic schemes and protocols. However, the definitions are more suited for research in complexity-theoretic aspects of cryptography.

For more practical cryptography, we introduce a slightly more flexible definition that combines the practicality of a weak OWF with the security of a strong OWF. In particular, instead of requiring the function to be one-way on a randomly chosen string, we define a domain and a domain sampler for *hard-to-invert* instances. Because the inspiration behind this definition comes from "candidate one-way functions," we also introduce the concept of a *collection* of functions; one function per input size.

Definition 34.1 (Collection of OWFs). A *collection of one-way functions* is a family $\mathcal{F} = \{f_i : \mathcal{D}_i \to \mathcal{R}_i\}_{i \in I}$ satisfying the following conditions:

1. It is easy to sample a function, i.e. there exists a p.p.t. Gen such that $Gen(1^n)$ outputs some $i \in I$.
2. It is easy to sample a given domain, i.e. there exists a p.p.t that on input i returns a uniformly random element of $\mathcal{D}_i$
3. It is easy to evaluate, i.e. there exists a p.p.t. that on input $i, x \in \mathcal{D}_i$ computes $f_i(x)$.

4. It is hard to invert, i.e. for any p.p.t. $\mathcal{A}$ there exists a negligible function ϵ such that

$$\Pr\left[i \leftarrow Gen; x \leftarrow \mathcal{D}_i; y \leftarrow f_i(x); z = \mathcal{A}(1^n, i, y) : f(z) = y\right] \leq \epsilon(n)$$

Despite our various relaxations, the existence of a collection of one-way functions is equivalent to the existence of a strong one-way function.

Theorem 35.1. *There exists a collection of one-way functions if and only if there exists a single strong one-way function.*

Proof idea: If we have a single one-way function f, then we can choose our index set ot be the singleton set $I = \{0\}$, choose $\mathcal{D}_0 = \mathbb{N}$, and $f_0 = f$.

The difficult direction is to construct a single one-way function given a collection $\mathcal{F}$. The trick is to define $g(r_1, r_2)$ to be $i, f_i(x)$ where i is generated using r_1 as the random bits and x is sampled from $\mathcal{D}_i$ using r_2 as the random bits. The fact that g is a strong one-way function is left as an excercise. □

2.7 Basic Computational Number Theory

Before we can study candidate collections of one-way functions, it serves us to review some basic algorithms and concepts in number theory and group theory.

2.7.1 Modular Arithmetic

We state the following basic facts about modular arithmetic:

Claim 35.1. *For $N > 0$ and $a, b \in \mathbb{Z}$,*

1. $(a \bmod N) + (b \bmod N) \bmod N = (a + b) \bmod N$
2. $(a \bmod N)(b \bmod N) \bmod N = ab \bmod N$

2.7.2 Euclid's algorithm

Euclid's algorithm appears in text around 300B.C.; it is therefore well-studied. Given two numbers a and b such that $a \geq b$, Euclid's algorithm computes the greatest common divisor of a and b, denoted $\gcd(a, b)$. It is not at all obvious how this value can be efficiently computed, without say, the factorization of both numbers. Euclid's insight was to notice that any divisor of a and b will also be a divisor of b and $a - b$. The latter is both easy to compute and a *smaller* problem than the original one. The algorithm has since been updated to use $a \bmod b$ in place of $a - b$ to improve efficiency. An elegant version of the algorithm which we present here also computes values x, y such that $ax + by = \gcd(a, b)$.

Algorithm 3: ExtendedEuclid(a, b)

Input: (a, b) s.t $a > b \geq 0$
Output: (x, y) s.t. $ax + by = \gcd(a, b)$
if $a \bmod b = 0$ **then**
  Return $(0, 1)$
else
  $(x, y) \leftarrow$ `ExtendedEuclid` $(b, a \bmod b)$
  Return $(y, x - y(\lfloor a/b \rfloor))$

Note: by polynomial time we always mean polynomial in the size of the input, that is $poly(\log a + \log b)$

Proof. On input $a > b \geq 0$, we aim to prove that Algorithm 3 returns (x, y) such that $ax + by = \gcd(a, b) = d$ via induction. First, let us argue that the procedure terminates in polynomial time. The original analysis by Lamé is slightly better; for us the following suffices since each recursive call involves only a constant number of divisions and subtraction operations.

Claim 36.1. *If $a > b \geq 0$ and $a < 2^n$, then ExtendedEuclid(a, b) makes at most $2n$ recursive calls.*

Proof. By inspection, if $n \leq 2$, the procedure returns after at most 2 recursive calls. Assume for $a < 2^n$. Now consider an instance with $a < 2^{n+1}$. We identify two cases.

1. If $b < 2^n$, then the next recursive call on $(b, a \bmod b)$ meets the inductive hypothesis and makes at most $2n$ recursive calls. Thus, the total number of recursive calls is less than $2n + 1 < 2(n + 1)$.

2. If $b > 2^n$, we can only guarantee that first argument of the next recursive call on $(b, a \bmod b)$ is upper-bounded by 2^{n+1} since $a > b$. Thus, the problem is no "smaller" on face. However, we can show that the second argument will be small enough to satisfy the prior case:

$$\begin{aligned} a \bmod b &= a - \lfloor a/b \rfloor \cdot b \\ &< 2^{n+1} - b \\ &< 2^{n+1} - 2^n = 2^n \end{aligned}$$

Thus, 2 recursive calls are required to reduce to the inductive case for a total of $2 + 2n = 2(n + 1)$ calls.

□

Now for correctness, suppose that b divided a evenly (i.e., $a \bmod b = 0$). Then we have $\gcd(a, b) = b$, and the algorithm returns $(0, 1)$ which is correct by inspection. By the inductive hypothesis, assume that the recursive call returns (x, y) such that

$$bx + (a \bmod b)y = \gcd(b, a \bmod b)$$

First, we claim that

Claim 37.1. $\gcd(a, b) = \gcd(b, a \bmod b)$

Proof. Divide a by b and write the result as $a = qb + r$. Rearrange to get $r = a - qb$.

Observe that if d is a divisor of a and b (i.e. $a = a'd$ and $b = b'd$ for $a', b' \in \mathbb{Z}$) then d is also a divisor of r since $r = (a'd) - q(b'd) = d(a' - qb')$. Since this holds for all divisors of a and b, it follows that $\gcd(a, b) = \gcd(b, r)$. □

Thus, we can write

$$bx + (a \bmod b)y = d$$

and by adding 0 to the right, and regrouping, we get

$$\begin{aligned} d &= bx - b(\lfloor a/b \rfloor)y + (a \bmod b)y + b(\lfloor a/b \rfloor)y \\ &= b(x - (\lfloor a/b \rfloor)y) + ay \end{aligned}$$

which shows that the return value $(y, x - (\lfloor a/b \rfloor)y)$ is correct. □

The assumption that the inputs are such that $a > b$ is without loss of generality since otherwise the first recursive call swaps the order of the inputs.

2.7.3 Exponentiation modulo N

Given a, x, N, we now demonstrate how to efficiently compute $a^x \bmod N$. Recall that by *efficient*, we require the computation to take polynomial time in the size of the representation of a, x, N. Since inputs are given in binary notation, this requires our procedure to run in time $poly(\log(a), \log(x), \log(N))$.

The key idea is to rewrite x in binary as $x = 2^\ell x_\ell + 2^{\ell-1} x_{\ell-1} + \cdots + 2x_1 + x_0$ where $x_i \in \{0, 1\}$ so that

$$a^x \bmod N = a^{2^\ell x_\ell + 2^{\ell-1} x_{\ell-1} + \cdots + 2^1 x_1 + x_0} \bmod N$$

We show this can be further simplified as

$$a^x \bmod n = \prod_{i=0}^{\ell} x_i a^{2^i} \bmod N$$

Algorithm 4: ModularExponentiation(a, x, N)

Input: $a, x \in [1, N]$
$r \leftarrow 1$
while $x > 0$ **do**
  if x *is odd* **then**
    $r \leftarrow r \cdot a \bmod N$
  $x \leftarrow \lfloor x/2 \rfloor$
  $a \leftarrow a^2 \bmod N$
Return r

Theorem 38.1. *On input* (a, x, N) *where* $a, x \in [1, N]$*, Algorithm 8 computes* $a^x \bmod N$ *in time* $O(\log^3(N))$.

Proof. By the basic facts of modulo arithmetic from Claim **[?]**, we can rewrite $a^x \bmod N$ as $\prod_i x_i a^{2^i} \bmod N$.

Since multiplying (and squaring) modulo N take time $\log^2(N)$, each iteration of the loop requires $O(\log^2(N))$ time. Because $x < N$, and each iteration divides x by two, the loop runs at most $\log N$ times which establishes a running time of $O(\log^3(N))$. □

Later, after we have introduced Euler's theorem, we present a similar algorithm for modular exponentiation which removes the restriction that $x < N$. In order to discuss this, we must introduce the notion of Groups.

2.7.4 Groups

Definition 38.1. A group G is a set of elements G with a binary operator $\oplus : G \times G \rightarrow G$ that satisfies the following axioms:

1. Closure: For all $a, b \in G$, $a \oplus b \in G$,
2. Identity: There is an element i in G such that for all $a \in G$, $i \oplus a = a \oplus i = a$. This element i is called the *identity* element.

3. Associativity: For all a, b and c in G, $(a \oplus b) \oplus c = a \oplus (b \oplus c)$.

4. Inverse: For all $a \in G$, there is an element $b \in G$ such that $a \oplus b = b \oplus a = i$ where i is the *identity*.

Example: The Additive Group Mod N

There are many natural groups which we have already implicitly worked with. The additive group modulo N is denoted $(\mathbb{Z}_N, +)$ where $Z_N = \{0, 1, ..N - 1\}$ and $+$ is addition modulo N. It is straightforward to verify the four properties for this set and operation.

Example: The Multiplicative Group Mod N

The multiplicative group modulo $N > 0$ is denoted $(\mathbb{Z}_N^*, \times)$, where $Z_N^* = \{x \in [1, N - 1] \mid \gcd(x, N) = 1\}$ and $\times$ is multiplication modulo N.

Theorem 39.1. *$(\mathbb{Z}_N^*, \times)$ is a group*

Proof. It is easy to see that 1 is the identity in this group and that $(a * b) * c = a * (b * c)$ for $a, b, c \in \mathbb{Z}_N^*$. However, we must verify that the group is closed and that each element has an inverse.

Closure For the sake of contradiction, suppose there are two elements $a, b \in \mathbb{Z}_N^*$ such that $ab \notin \mathbb{Z}_N^*$. This implies that $\gcd(a, N) = 1$, $\gcd(b, N) = 1$, but that $\gcd(ab, N) = d > 1$. The latter condition implies that d has a non-trivial factor that divides both ab and N. Thus, d must also divide either a or b (verify as an exercise), which contradicts the assumption that $\gcd(a, N) = 1$ or $\gcd(b, N) = 1$.

Inverse Consider an element $a \in \mathbb{Z}_N^*$. Since $\gcd(a, N) = 1$, we can use Euclid's algorithm on (a, N) to compute values (x, y) such that $ax + Ny = 1$. Notice, this directly produces a value x such that $ax = 1 \bmod N$. Thus, every element $a \in \mathbb{Z}_N^*$ has an inverse which can be efficiently computed. □

Remark: The groups $(\mathbb{Z}_N, +)$ and $(\mathbb{Z}_N^*, \times)$ are also *abelian* or commutative groups in which $a \oplus b = b \oplus a$.

Definition 39.1. A *prime* is a positive integer that is divisible by only 1 and itself.

The number of unique elements in $\mathbb{Z}_N^*$ (often referred to as the *order* of the group) is denoted by the the Euler Totient function $\Phi(N)$.

$$\begin{array}{ll} \Phi(p) = p-1 & \text{if } p \text{ is prime} \\ \Phi(N) = (p-1)(q-1) & \text{if } N = pq \text{ and } p, q \text{ are primes} \end{array}$$

The first case follows because all elements less than p will be relatively prime to p. The second case requires some simple counting (show this).

The structure of these multiplicative groups results in some very special properties which we can exploit throughout this course. One of the first properties is the following identity first proven by Euler in 1736.

Theorem 40.1 (Euler). $\forall a \in \mathbb{Z}_N^*, a^{\Phi(N)} \equiv 1 \bmod N$

Proof. Consider the set $A = \{ax \mid x \in \mathbb{Z}_N^*\}$. Since $\mathbb{Z}_N^*$ is a group, it follows that $A \subseteq \mathbb{Z}_N^*$ since every element $ax \in \mathbb{Z}_N^*$. Now suppose that $|A| < |\mathbb{Z}_N^*|$. By the pidgeonhole principle, this implies that there exist two group element $i, j \in \mathbb{Z}_N^*$ such that $i \neq j$ but $ai = aj$. Since $a \in \mathbb{Z}_N^*$, there exists an inverse a^{-1} such that $aa^{-1} = 1$. Multiplying on both sides we have $a^{-1}ai = a^{-1}aj \implies i = j$ which is a contradiction. Thus, $|A| = |\mathbb{Z}_N^*|$ which implies that $A = \mathbb{Z}_N^*$.

Because the group is abelian (i.e., commutative), we can take products and substitute the defintion of A to get

$$\prod_{x \in \mathbb{Z}_N^*} x = \prod_{y \in A} y = \prod_{x \in \mathbb{Z}^* n} ax$$

The product further simplifies as

$$\prod_{x \in \mathbb{Z}_N^*} x = a^{\Phi(n)} \prod_{x \in \mathbb{Z}_N^*} x$$

Finally, since the closure property guarantees that $\prod_{x \in \mathbb{Z}_N^*} x \in \mathbb{Z}_N^*$ and since the inverse property guarantees that this element has an inverse, we can multiply the inverse on both sides to obtain

$$1 = a^{\Phi(n)}$$

□

Corollary 40.1 (Fermat's little theorem). $\forall a \in \mathbb{Z}_p^*, a^{p-1} \equiv 1 \bmod p$

Corollary 40.2. $a^x \bmod N = a^{x \bmod \Phi(N)} \bmod N$. *Thus, given* $\Phi(N)$*, the operation* $a^x \bmod N$ *can be computed efficiently in* $\mathbb{Z}_N$ *for any* x.

Thus, we can compute a value such as $2^{2^x} \bmod N$ quickly.

2.7.5 Primality Testing

An important task in generating the parameters of a many cryptographic scheme will be the identification of a suitably large prime number. Eratosthenes (276-174BC), the librarian of Alexandria, is credited with devising a very simple sieving method to enumerate all primes. However, this method is not practical for choosing a large (i.e., 1000 digit) prime.

Instead, recall that Fermat's Little Theorem establishes that $a^{p-1} = 1 \bmod p$ for any $a \in \mathbb{Z}_p$ whenever p is prime. It turns out that when p is not prime, then a^{p-1} is usually *not* equal to 1. The first fact and second phenomena form the basic idea behind the Miller-Rabin primality test: to test p, pick a random $a \in \mathbb{Z}_p$, and check whether $a^{p-1} = 1 \bmod p$. (Notice that efficient modular exponentiation is critical for this test.) Unfortunately, the second phenomena is *on rare occasion* false. Despite there rarity (starting with $561, 1105, 1729, \ldots,$, there are only 255 such cases less than 10^8!), there are infact an infinite number of counter examples collectively known as the *Carmichael* numbers. Thus, for correctness, we must test for these rare cases. This second check amounts to verifying that none of the *intermediate* powers of a encountered during the modular exponentiation computation of a^{n-1} are non-trivial square-roots of 1. This suggest the following approach.
For positive n, write $n = u2^j$ where u is odd. Define the set

$$L_N = \left\{ \alpha \in \mathbb{Z}_N \mid \alpha^{N-1} = 1 \text{ and if } \alpha^{u2^{j+1}} = 1 \text{ then } \alpha^{u2^j} = 1 \right\}$$

The first condition is based on Fermat's Little theorem, and the second one eliminates errors due to Carmichael numbers.

Theorem 41.1. *If N is an odd prime, then $|L_N| = N - 1$. If $N > 2$ is composite, then $|L_N| < (N-1)/2$.*

We will not prove this theorem here. See **[?]** for a formal proof. The proof idea is as follows. If n is prime, then by Fermat's Little Theorem, the first condition will always hold, and since 1 only has two square roots modulo p (namely, $1, -1$), the second condition holds as well. If N is composite, then either there will be some α for which α^{N-1} is not equal to 1 or the process of computing α^{N-1} reveals a square root of 1 which is different from 1 or -1—recall that when N is composite, there are at least 4 square roots of 1. More formally, the proof works by first arguing that all of the $\alpha \notin L_N$ form a *proper* subgroup of $\mathbb{Z}_N^*$. Since the order of a subgroup must divide the order of the group, the size of a proper subgroup must therefore be less than $(N-1)/2$.

This suggests the following algorithm for testing primality:

Algorithm 5: Miller-Rabin Primality Test

Handle base case $N = 2$
for t *times* **do**
  Pick a random $\alpha \in \mathbb{Z}_N$
  if $\alpha \notin L_N$ **then** Output "composite"
Output "prime"

Observe that testing whether $\alpha \in L_N$ can be done by using a repeated-squaring algorithm to compute modular exponentiation, and adding internal checks to make sure that no non-trivial roots of unity are discovered.

Theorem 42.1. *If N is composite, then the Miller-Rabin test outputs "composite" with probability $1 - 2^{-t}$. If N is prime, then the test outputs "prime."*

We mention that a more complicated (and significantly slower), but *deterministic* polynomial time algotithm, for primality testing was recently shown by Agrawal, Saxena and Kayal.

2.7.6 Selecting a Random Prime

Our algorithm for finding a random n-bit prime will be simple: we will repeatedly sample a n-bit number and then check whether it is prime.

Algorithm 6: SamplePrime(n)

repeat
  $x \xleftarrow{r} \{0,1\}^n$
  if *Miller-Rabin*(x) = *"prime"* **then** Return x
until *done*

There are two mathematical facts which make this simple scheme work:

1. There are many primes
2. It is easy to determine whether a number is prime

By Theorem 27.1, the probability that a uniformly-sampled n-bit integer is prime is greater than $(N/\log N)/N = \frac{c}{n}$. Thus, the expected number of guesses in the guess-and-check method is polynomial in n. Since the running time of the Miller-Rabin algorithm is also polynomial in n, the expected running time to sample a random prime using the guess-and-check approach is polynomial in n.

2.8 Factoring-based Collection

Under the factoring assumption, we can prove the following result, which establishes our first realistic collection of one-way functions:

Theorem 43.1. *Let* $\mathcal{F} = \{f_i : \mathcal{D}_i \to \mathcal{R}_i\}_{i \in I}$ *where*

$$\begin{aligned} I &= \mathbb{N} \\ \mathcal{D}_i &= \{(p,q) \mid p, q \text{ prime, and } |p| = |q| = \frac{i}{2}\} \\ f_i(p,q) &= p \cdot q \end{aligned}$$

Assuming the Factoring Assumption holds, then $\mathcal{F}$ *is a collection of OWF.*

Proof. We can clearly sample a random element of $\mathbb{N}$. It is easy to evaluate f_i since multiplication is efficiently computable, and the factoring assumption says that inverting f_i is hard. Thus, all that remains is to present a method to efficiently sample two random prime numbers. This follows from the section below. Thus all four conditions in the definition of a one-way collection are satisfied. □

2.9 Discrete Logarithm-based Collection

Another often used collection is based on the discrete logarithm problem in the group $\mathbb{Z}_p^*$.

2.9.1 Discrete logarithm modulo p

An instance of the discrete logarithm problem consists of two elements $g, y \in \mathbb{Z}_p^*$. The task is to find an x such that $g^x = y \bmod p$. In some special cases (e.g., $g = 1$), it is easy to either solve the problem or declare that no solution exists. However, when g is a *generator* or Z_p^*, then the problem is believed to be hard.

Definition 43.1 (Generator of a Group). A element g of a multiplicative group G is a generator if the set $\{g, g^2, g^3, \ldots\} = G$. We denote the set of all generators of a group G by Gen_G.

Conjecture 43.1 (Discrete Log Assumption). *For every non-uniform p.p.t. algorithm $\mathcal{A}$, there exists a negligible function ϵ such that*

$$\Pr\left[p \xleftarrow{r} \Pi_n; g \xleftarrow{r} Gen_p; x \xleftarrow{r} \mathbb{Z}_p; y \leftarrow g^x \bmod p : \mathcal{A}(y) = x\right] < \epsilon(n)$$

Theorem 44.1. *Let* $\mathbf{DL} = \{f_i : \mathcal{D}_i \to \mathcal{R}_i\}_{i \in I}$ *where*

$$\begin{aligned} I &= \{(p, g) \mid p \in \Pi_k, g \in Gen_p\} \\ \mathcal{D}_i &= \{x \mid x \in \mathbb{Z}_p\} \\ \mathcal{R}_i &= \mathbb{Z}_p^* \\ f_{p,g}(x) &= g^x \bmod p \end{aligned}$$

*Assuming the Discrete Log Assumption holds, the family of functions **DL** is a collection of one-way functions.*

Proof. It is easy to sample the domain D_i and to evaluate the function $f_{p,g}(x)$. The discrete log assumption implies that $f_{p,g}$ is hard to invert. Thus, all that remains is to prove that I can be sampled efficiently. Unfortunately, given only a prime p, it is not known how to efficiently choose a generator $g \in Gen_p$. However, it is possible to sample both a prime *and* a generator g at the same time. One approach proposed by Bach and later adapted by Kalai is to sample a k-bit integer x in factored form (i.e., sample the integer and its factorization at the same time) such that $p = x + 1$ is prime. Given such a pair $p, (q_1, \ldots, q_k)$, one can use a central result from group theory to test whether an element is a generator.

Another approach is to pick primes of the form $p = 2q + 1$. These are known as Sophie Germain primes or "safe primes." The standard method for sampling such a prime is simple: first pick a prime q as usual, and then check whether $2q + 1$ is also prime. Unfortunately, even though this procedure always terminates in practice, its basic theoretical properties are unknown. That is to say, it is unknown even (a) whether there are an infinite number of Sophie Germain primes, (b) and even so, whether such a procedure continues to quickly succeed as the size of q increases. Despite these unknowns, once such a prime is chosen, testing whether an element $g \in \mathbb{Z}_p^*$ is a generator consists of checking $g \neq \pm 1 \bmod p$ and $g^q \neq 1 \bmod p$. □

As we will see later, the collection **DL** is also a special collection of one-way functions in which each function is a permutation.

2.10 RSA collection

Another popular assumption is the RSA-assumption.

Conjecture 45.0 (RSA Assumption). *Given* (N, e, y) *such that* $N = pq$ *where* $p, q \in \Pi_n$, $\gcd(e, \Phi(N)) = 1$ *and* $y \in \mathbb{Z}_N^*$, *the probability that any non-uniform p.p.t. algorithm* A *is able to produce* x *such that* $x^e = y \bmod N$ *is a negligible function* $\epsilon(n)$.

$$\Pr[p, q \xleftarrow{r} \Pi_n;\ n \leftarrow pq;\ e \xleftarrow{r} \mathbb{Z}_{\Phi(N)}^*;\ y \leftarrow \mathbb{Z}_N^* : A(N, e, y) = x \text{ s.t. } x^e = y \bmod N] < \epsilon(n)$$

Theorem 45.1 (RSA Collection). *Let* $\boldsymbol{RSA} = \{f_i : \mathcal{D}_i \to \mathcal{R}_i\}_{i \in I}$ *where*

$$\begin{aligned} I &= \{(N, e) \mid N = pq \text{ s.t. } p, q \in \Pi_n \text{ and } e \in \mathbb{Z}_{\Phi(N)}^*\} \\ \mathcal{D}_i &= \{x \mid x \in \mathbb{Z}_N^*\} \\ \mathcal{R}_i &= \mathbb{Z}_N^* \\ f_{N,e}(x) &= x^e \bmod N \end{aligned}$$

Assuming the RSA Assumption holds, the family of functions ***RSA*** *is a collection of one-way functions.*

Proof. The set I is easy to sample: generate two primes p, q, multiply then to generate N, and use the fact that $\Phi(N) = (p-1)(q-1)$ to sample a random element from $\mathbb{Z}_{\Phi(N)}^*$. Likewise, the set D_i is also easy to sample and the function $f_{N,e}$ requires only one modular exponentiation to evaluate. It only remains to show that $f_{N,e}$ is difficult to invert. Notice, however, that this does not *directly* follow from the our hardness assumption (as it did in previous examples). The RSA assumption states that it is difficult to compute the eth root of a *random* group element y. On the other hand, our collection first picks the root and then computes $y \leftarrow x^e \bmod N$. One could imagine that picking an element that is *known* to have an eth root makes it easier to find such a root. We will show that this is not the case—and we will do so by showing that the function$f_{N,e}(x) = x^e \bmod N$ is in fact a permutation of the elements of $\mathbb{Z}_N^*$. This would then imply that the distributions $\{x, e \xleftarrow{r} \mathbb{Z}_N^* : (e, x^e \bmod N)\}$ and $\{y, e \xleftarrow{r} \mathbb{Z}_N^* : (e, y)\}$ are identical, and so an algorithm that inverts $f_{N,e}$ would also succeed at breaking the RSA-assumption. □

Theorem 45.2. *The function* $f_{N,e}(x) = x^e \bmod N$ *is a permutation of* $\mathbb{Z}_N^*$ *when* $e \in \mathbb{Z}_{\Phi(N)}^*$.

Proof. Since e is an element of the group $Z^*_{\Phi(N)}$, let d be its inverse (recall that every element in a group has an inverse), i.e. $ed = 1 \bmod \Phi(N)$. Consider the inverse map $g_{N,e}(x) = x^d \bmod N$. Now for any $x \in \mathbb{Z}^*_N$,

$$\begin{aligned} g_{N,e}(f_{N,e}(x)) &= g_{N,e}(x^e \bmod N) = (x^e \bmod N)^d \\ &= x^{ed} \bmod N \\ &= x^{c\Phi(N)+1} \bmod N \end{aligned}$$

for some constant c. Since Euler's theorem establishes that $x^{\Phi(N)} = 1 \bmod N$, then the above can be simplfied as

$$x^{c\Phi(N)} \cdot x \bmod N = x \bmod N$$

Hence, RSA is a permutation. □

This phenomena suggests that we formalize a new, stronger class of one-way functions.

2.11 One-way Permutations

Definition 46.1 (One-way permutation). A collection $\mathcal{F} = \{f_i : \mathcal{D}_i \to R_i\}_{i \in I}$ is a collection of *one-way permutations* if $\mathcal{F}$ is a collection of *one-way functions* and for all $i \in I$, we have that f_i is a permutation.

A natural question is whether this extra property comes at a price—that is, how does the RSA-assumption that we must make compare to a natural assumption such as factoring. Here, we can immediately show that RSA is at least as strong an assumption as Factoring.

Theorem 46.1. *The RSA-assumption implies the Factoring assumption.*

Proof. We prove by contrapositive: if factoring is possible in polynomial time, then we can break the RSA assumption in polynomial time. Formally, assume there an algorithm A and polynomial function $p(n)$ so that A can factor $N = pq$ with probability $1/p(n)$, where p and q are random n-bits primes. Then there exists an algorithm A', which can invert $f_{N,e}$ with probability $1/p(n)$, where $N = pq$, $p,\ q \leftarrow \{0,1\}^n$ primes, and $e \leftarrow \mathbb{Z}^*_{\Phi(N)}$.

The algorithm feeds the factoring algorithm A with exactly the same distribution of inputs as with the factoring assumption. Hence in the first step A will return the correct prime factors with probability $1/p(n)$. Provided that the factors are correct, then we can compute the inverse of y in the same

Algorithm 7: Adversary $A'(N, e, y)$

Run $(p, q) \leftarrow A(N)$ to recover prime factors of N
if $N \neq pq$ **then** abort
Compute $\Phi(N) \leftarrow (p-1)(q-1)$
Compute the inverse d of e in $\mathbb{Z}^*_{\Phi(N)}$ using Euclid
Output $y^d \bmod N$

way as we construct the inverse map of $f_{N,e}$. And this always succeeds with probability 1. Thus overall, A' succeeds in breaking the RSA-assumption with probability $1/p(n)$. Moreover, the running time of A' is essentially the running time of A plus $O(\log^3(n))$. Thus, if A succeeds in factoring in polynomial time, then A' succeeds in breaking the RSA-assumption in roughly the same time. □

Unfortunately, it is not known whether the converse it true—i.e., whether the factoring assumption also implies the RSA-assumption.

2.12 Trapdoor Permutations

The proof that RSA is a permutation actually suggests another special property of that collection: if the factorization of N is unknown, then inverting $f_{N,e}$ is considered infeasiable; however if the factorization of N *is known*, then it is no longer hard to invert. In this sense, the factorization of N is a *trapdoor* which enables $f_{N,e}$ to be inverted.

This spawns the idea of trapdoor permutations, first conceived by Diffie and Hellman.

Definition 47.1 (Trapdoor Permutations). A *collection of trapdoor permutations* if a family $\mathcal{F} = \{f_i : \mathcal{D}_i \rightarrow \mathcal{R}_i\}_{i \in \mathcal{I}}$ satisfying the following properties:

1. $\forall i \in \mathcal{I}$, f_i is a permutation,
2. *It is easy to sample function:* $\exists$ p.p.t. Gen s.t. $(i, t) \leftarrow Gen(1^n),\ i \in \mathcal{I}$ (t is trapdoor info),
3. *It is easy to sample the domain:* $\exists$ p.p.t. machine that given input $i \in \mathcal{I}$, samples uniformly in $\mathcal{D}_i$.
4. f_i *is easy to evaluate:* $\exists$ p.p.t. machine that given input $i \in \mathcal{I}, x \in \mathcal{D}_i$, computes $f_i(x)$.

5. f_i *is hard to invert:* $\forall$ p.p.t. $\mathcal{A}$, $\exists$ negligible ϵ s.t.

$$\Pr[(i,t) \leftarrow Gen(1^n); x \in \mathcal{D}_i; y = f(x); z = A(1^n, i, y) : f(z) = y] \leq \epsilon(k)$$

6. f_i *is easy to invert with trapdoor information:* $\exists$ p.p.t. machine that given input (i,t) from Gen and $y \in \mathcal{R}_i$, computes $f^{-1}(y)$.

Now by slightly modifying the definition of the family **RSA**, we can easily show that it is a collection of trapdoor permutations.

Theorem 48.1. *Let **RSA** be defined as per Theorem 45.1 with the exception that*

$$[(N,e),(d)] \leftarrow Gen(1^n)$$
$$f^{-1}_{N,d}(y) = y^d \bmod N$$

where $N = pq$, $e \in \mathbb{Z}^*_{\Phi(N)}$ *and* $ed = 1 \bmod \Phi(N)$. *Assuming the RSA-assumption, the collection* RSA *is a collection of trapdoor permutations.*

The proof is an exercise.

2.13 A Universal One Way Function

As we have mentioned in previous sections, it is not known whether one-way functions exist. Although we have presented specific assumptions which have lead to specific constructions, a much weaker assumption is to assume only that *some* one-way function exists (without knowing exactly which one). The following theorem gives a single constructible function that is one-way if this weaker assumption is true.

Theorem 48.2. *If there exists a one-way function, then the following polynomial-time computable function* $f_{\text{universal}}$ *is also a one-way function.*

Proof. We will construct function $f_{\text{universal}}$ and show that it is weakly one-way. We can then apply the hardness amplification construction from §2.3 to get a strong one-way function.

The idea behind the construction is that $f_{\text{universal}}$ should combine the computations in all efficient functions in such a way that inverting $f_{\text{universal}}$ allows us to invert all other functions. One approach is to interpret the input y to $f_{\text{universal}}$ as a machine-input pair $\langle M, x\rangle$, and then let the output of $f_{\text{universal}}(y)$ be $M(x)$. The problem with this approach is that $f_{\text{universal}}$ will not be computable in polynomial time, since M may or may not even terminate on input

x. We can overcome this problem by only running $M(x)$ for a number of steps related to $|y|$.

ALGORITHM 48.1: $f_{\text{UNIVERSAL}}(y)$: A UNIVERSAL ONE-WAY FUNCTION

Interpret y as $M, x\rangle$ where and $|M| = \log(|y|)$
Run M on input x for $|y|^3$ steps
if M terminates **then**
  Output $M(x)$
else
  Output $\perp$
end if

In words, this function interprets the first $\log|y|$ bits of the input y as a machine M, and the remaining bits are considered input x. We assume a standard way to encode Turing machines with appropriate padding. We claim that this function is weakly one-way. Clearly $f_{\text{universal}}$ is computable in $O(n^3)$ time, and thus it satisfies the "easy" criterion for being one-way. To show that it satisfies the "hard" criterion, we must assume that there exists some function g that is strongly one-way. By the following lemma, we can assume that g runs in $O(n^2)$ time.

Lemma 49.1. *If there exists a strongly one-way function g, then there exists a strongly one-way function g' that is computable in time $O(n^2)$.*

Proof. Suppose g runs in $O(n^c)$ for some $c > 2$. (If not, then the lemma already holds.) Let $g'(\langle a, b\rangle) = \langle a, g(b)\rangle$, where $|a| \approx n^c$ and $|b| = n$. Then if we let $m = |\langle a, b\rangle| = O(n^c)$, the function g' is computable in time

$$\underbrace{|a|}_{\text{copying a}} + \underbrace{|b|^c}_{\text{computing } g} + \underbrace{O(m^2)}_{\text{parsing}} < 2m + O(m^2) = O(m^2)$$

Moreover, g' is still one-way, since we can easily reduce inverting g to inverting g'. □

Now, if $f_{\text{universal}}$ is not weakly one-way, then there exists a machine $\mathcal{A}$ such that for every polynomial q and for infinitely many input lengths n,

$$\Pr\left[y \leftarrow \{0,1\}^n; \mathcal{A}(f(y)) \in f^{-1}(f(y))\right] > 1 - 1/q(n)$$

In particular, this holds for $q(n) = n^3$. Denote the event that $\mathcal{A}$ inverts as Invert.

Let M_g be the smallest machine which computes function g. Since M_g is a uniform algorithm it has some constant description size $|M_g|$. Thus, on a random n-bit input $y = \langle M, x\rangle$, the probability that machine $M = M_g$ (with appropriate padding of 0) is $2^{-\log n} = 1/n$. In other words

$$\Pr\left[y \xleftarrow{r} \{0,1\}^n \ : y = \langle M_g, x\rangle\right] \geq \frac{1}{n}$$

Denote this event as event PickG. We can now combine the above two equations to conclude that $\mathcal{A}$ inverts an instance of g with noticeable probability. By the Union Bound, either $\mathcal{A}$ fails to invert or the instance fails to be g with probability at most

$$\Pr[!\mathsf{Invert} \vee !\mathsf{PickG}] \leq (1/n^3) + (1 - 1/n) < \frac{n^3 - 1}{n^3}$$

Therefore, $\mathcal{A}$ must invert a hard instance of g with probability

$$\Pr[\mathsf{Invert}\ \mathsf{and}\ \mathsf{PickG}] \geq \frac{1}{n^3}$$

which contradicts the assumption that g is strongly one-way; therefore $f_{\text{universal}}$ must be weakly one-way. □

This theorem gives us a function that we can safely assume is one-way (because that assumption is equivalent to the assumption that one-way functions exist). However, it is extremely impractical to compute. First, it is difficult to compute because it involves interpreting random turing machines. Second, it will require very large key lengths before the hardness kicks in. A very open problem is to find a "nicer" universal one way function (e.g. it would be very nice if f_{mult} is universal).

Chapter 3

Indistinguishability & Pseudo-Randomness

Recall that one main drawback of the One-time pad encryption scheme—and its simple encryption operation $Enc_k(m) = m \oplus k$—is that the key k needs to be as long as the message m. A natural approach for making the scheme more efficient would be to start off with a short random key k and then try to use some *pseudo-random generator* g to expand it into a longer "random-looking" key $k' = g(k)$, and finally use k' as the key in the One-time pad.

Can this be done? We start by noting that there can not exist pseudo-random generators g that on input k generate a *perfectly random* string k', as this would contradict Shannon's theorem (show this). However, remember that Shannon's lower bound relied on the premise that the adversary Eve is computationally unbounded. Thus, if we restrict our attention to efficient adversaries, it might be possible to devise pseudo-random generators that output strings which are "sufficiently" random-looking for our encryption application.

To approach this problem, we must first understand what it means for a string to be "sufficiently random-looking" to a polynomial time adversary. Possible answers include:

— Roughly as many 0 as 1.

— Roughly as many 00 as 11

— Each particular bit is "roughly" unbiased.

— Each sequence of bits occurs with "roughly" the same probability.

— Given any prefix, it is hard to guess the next bit.

— Given any prefix, it is hard to guess the next sequence.

All of the above answers are examples of specific *statistical tests*—and many many more such test exist in the litterature. For specific simulations, it may be enough to use strings that pass some specific statistical tests. However, for cryptography, we require the use of string that passes *all* (efficient) statistical tests. At first, it seems quite overwhelming to test a candidate pseudo-random generator against *all* efficient tests. To do so requires some more abstract concepts which we now introduce.

3.1 Computational Indistinguihability

We introduce the notion of *computational indistinguishability* to formalize what it means for two probability distributions to "look" the same in the eyes of a computationally bounded adversary. This notion is one of the corner stones of modern cryptography. To begin our discussion, we present two games that illustrate important ideas in the notion if indistinguishability.

Game 1 Flip the page and spend no more than two seconds looking at Fig. 3.1 that appears on the next page. Do the two boxes contain the same arrangement of circles?

Now suppose you repeat the experiment but spend 10 seconds instead of two. Imagine spending 10 minutes instead of 10 seconds. If you are only given a short amount of time, the two boxes appear indistinguishable from one another. As you take more and more time to analyze the images, you are able to tease apart subtle differences between the left and right. Generalizing, even if two probability distrubutions are completely disjoint, it may be that an observer who is only given limited processing time cannot distinguish between the two distributions.

Game 2 A second issue concerns the size of a problem instance. Consider the following sequence of games parameterized by the value n in Fig. 3.2. The point of the game is to determine if the number of overlapping boxes is even or odd. An example of each case is given on the extreme left. The parameter n indicates the number of boxes in the puzzle. Notice that small instances of the puzzle are easy to solve, and thus "odd" instances are easily distinguishable from "even" ones. However, by considering a sequence of puzzles, as n increases, a human's ability to solve the puzzle correctly rapidly approaches $1/2$—i.e., no better than guessing.

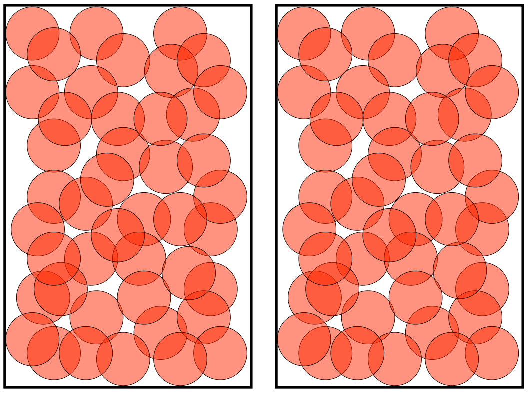
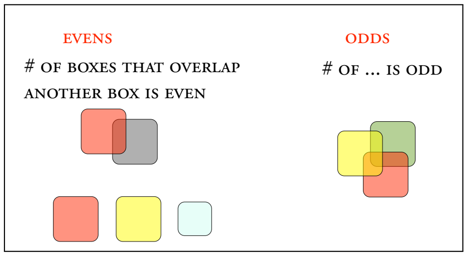
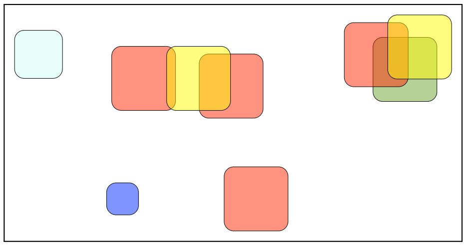


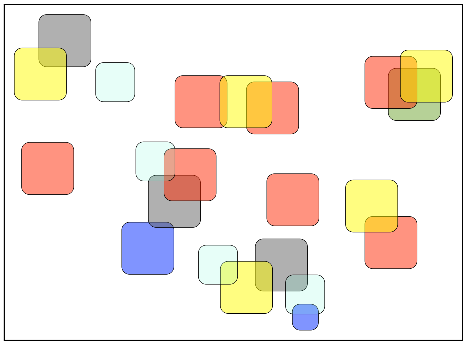
Figure 3.1: Are the two boxes the same or different?



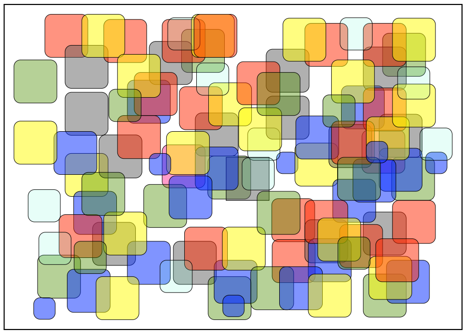
(a) Description of Game



(b) $n = 9$: Even or Odd?



(c) $n = 22$: Even or Odd?



(d) $n > 80$: Even or Odd?

Figure 3.2: A game parameterized by n

As our treatment is asymptotic, the actual formalization of this notion considers sequences—called *ensembles*—of probability distributions (or growing output length).

Definition 53.1 (Ensembles of Probability Distributions). A sequence $\{X_n\}_{n\in\mathbb{N}}$ is called a *probability ensemble* if for each $n \in \mathbb{N}$, X_n is a probability distribution over $\{0,1\}^*$.

Normally, ensembles are indexed by the natural numbers $n \in \mathbb{N}$. Thus, for the rest of this book, unless otherwise specified, we use $\{X_n\}_n$ to represent such an ensemble.

Definition 54.1 (Computational Indistinguishability). Let $\{X_n\}_n$ and $\{Y_n\}_n$ be ensembles of probability distributions where X_n, Y_n are probability distributions over $\{0,1\}^{l(n)}$ for some polynomial $l(\cdot)$. We say that $\{X_n\}_n$ and $\{Y_n\}_n$ are *computationally indistinguishable* (abbr. $\{X_n\}_n \approx \{Y_n\}_n$) if for all non-uniform p.p.t D (called the "distinguisher"), there exists a negligible function $\epsilon(\cdot)$ such that $\forall n \in \mathbb{N}$

$$\left|\Pr\left[t \leftarrow X_n, D(t) = 1\right] - \Pr\left[t \leftarrow Y_n, D(t) = 1\right]\right| < \epsilon(n).$$

In other words, two (ensembles of) probability distributions are computationally indistinguishable if no efficient distinguisher D can tell them apart better than with a negligible advantage.

To simplify notation, we say that D *distinguishes the distributions* X_n and Y_n with probability ϵ if

$$\left|\Pr\left[t \leftarrow X_n, D(t) = 1\right] - \Pr\left[t \leftarrow Y_n, D(t) = 1\right]\right| > \epsilon.$$

Additionally, we say D *distinguishes the probability ensembles* $\{X_n\}_n$ and $\{Y_n\}_n$ with probability $\mu(\cdot)$ if $\forall n \in \mathbb{N}$, D distinguishes X_n and Y_n with probability $\mu(n)$.

3.1.1 Properties of Computational Indistinguishability

We highlight some important (and natural) properties of the notion of indistinguishability. This properties will be used over and over again in the remainder of the course.

Closure under efficient opertations

The first property formalizes the statement "If two distributions look the same, then they look the same no matter how you process them" (as long as the processing is efficient). More formally, if two distributions are indistinguishable, then they remain indistinguishable also if one applies a p.p.t. computable operation to them.

Lemma 54.1 (Closure under Efficient Operations Lemma). *Let $\{X_n\}_n$,$\{Y_n\}_n$ be ensembles of probability distributions where X_n, Y_n are probability distributions over $\{0,1\}^{\ell(n)}$ for some polynomial $\ell(\cdot)$, and let M be a n.u. p.p.t. machine. If $\{X_n\}_n \approx \{Y_n\}_n$, then $\{M(X_n)\}_n \approx \{M(Y_n)\}_n$.*

Proof. Suppose there exists a non-uniform p.p.t. D and non-negligible function $\mu(n)$ s.t D distinguishes $\{M(X_n)\}_n$ and $\{M(Y_n)\}_n$ with probability $\mu(n)$. That is,

$$\begin{aligned} &|\Pr[t \leftarrow M(X_n) : D(t) = 1] - \Pr[t \leftarrow M(Y_n) : D(t) = 1]| &> \mu(n) \\ \Rightarrow &|\Pr[t \leftarrow X_n : D(M(t)) = 1] - \Pr[t \leftarrow Y_n : D(M(t)) = 1]| &> \mu(n). \end{aligned}$$

In that case, the non-uniform p.p.t machine $D'(\cdot) = D(M(\cdot))$ also distinguishes $\{X_n\}_n$, $\{Y_n\}_n$ with probability $\mu(n)$, which contradicts that the assumption that $\{X_n\}_n \approx \{Y_n\}_n$. □

Transitivity - The Hybrid Lemma

We next show that the notion of computational indistinguishability is transitive; namely, if $\{A_n\}_n \approx \{B_n\}_n$ and $\{B_n\}_n \approx \{C_n\}_n$, then $\{A_n\}_n \approx \{C_n\}_n$. In fact, we prove a generalization of this statement which considers $m = poly(n)$ distributions.

Lemma 55.1 (Hybrid Lemma). *Let $X^1, X^2, \cdots, X^m$ be a sequence of probability distributions. Assume that the machine D distinguishes X^1 and X^m with probability ϵ. Then there exists some $i \in [1, \cdots, m-1]$ s.t. D distinguishes X^i and X^{i+1} with probability $\frac{\epsilon}{m}$.*

Proof. Assume D distinguishes X^1, X^m with probability ϵ. That is,

$$|\Pr\left[t \leftarrow X^1 : D(t) = 1\right] - \Pr[t \leftarrow X^m : D(t) = 1]| > \epsilon$$

Let $g_i = \Pr\left[t \leftarrow X^i : D(t) = 1\right]$. Thus, $|g_1 - g_m| > \epsilon$. This implies,

$$\begin{aligned} &|g_1 - g_2| + |g_2 - g_3| + \cdots + |g_{m-1} - g_m| \\ &\quad \geq |g_1 - g_2 + g_2 - g_3 + \cdots + g_{m-1} - g_m| \\ &\quad = |g_1 - g_m| > \epsilon. \end{aligned}$$

Therefore, there must exist i such that $|g_i - g_{i+1}| > \frac{\epsilon}{m}$. □

Remark 55.1 (A geometric interpretation). Note that the probability with which D outputs 1 induces a metric space over probability distributions over strings t. Given this view the hybrid lemma is just a restatement of the traingle inequality over this metric spaces; in other words, if the distance between two consequetive points—representing probability distributions—-is small, then the distance between the extremal points is small too.

Note that because we lose a factor of m when we have a sequence of m distributions, the hybrid lemma can only be used to deduce transitivity when m is polynomially related to the security parameter n. (In fact, it is easy to construct a "long" sequence of probability distributions which are all indistinguishable, but where the extremal distributions are distinguishable.)

Distinguishing versus Predicting

The notion of computational indistinguishability requires that no efficient distinguisher algorithm can tell apart two distributions with more than a negligible advantage. This clearly means that there does not exist a machine that can *predict* what distributions a sample came from, with probability $\frac{1}{2} + \frac{1}{poly(n)}$; any such predictor would be a valid distinguisher (show this!). As the following useful lemma shows, the converse also holds: if you cannot predict what distribution a sample comes from, with probability significantly better than $\frac{1}{2}$, then the distributions must be indistinguishable.

Lemma 56.1 (The Prediction Lemma). *Let $\{X_n^0\}_n$,$\{X_n^1\}_n$ be ensembles of probability distributions where X_n^0, X_n^1 are probability distributions over $\{0,1\}^{\ell(n)}$ for some polynomial $\ell(\cdot)$, and let D be a n.u. p.p.t. machine that distinguishes between $\{X_n^0\}_n$ and $\{X_n^1\}_n$ w.p. $\mu(\cdot)$ for infinitely many $n \in N$. Then there exists a n.u. p.p.t. A such that*

$$\Pr\left[b \leftarrow \{0,1\}; t \leftarrow X_n^b : A(t) = b\right] \geq \frac{1}{2} + \frac{\mu(n)}{2}.$$

for infinitely many $n \in N$.

Proof. Consider some non-uniform p.p.t. D that distinguishes between X_n^0 and X_n^1 w.p. $\mu(n)$ for infinitely many $n \in \mathbb{N}$. We can assume, without loss of generality, that D outputs 1 with higher probability when receiving a sample from X_n^1 than when receiving a sample from X_n^0, i.e.,

$$\Pr\left[t \leftarrow X_n^1 \ : \ D(t) = 1\right] - \Pr\left[t \leftarrow X_n^0 \ : \ D(t) = 1\right] > \mu(n) \qquad (3.1)$$

This is without loss of generality since we could always replace D with $D'(\cdot) = 1 - D(\cdot)$; one of these distinguishers will work for infinitely many $n \in N$. We show that D, in fact, is also a "predictor":

$$\begin{aligned}
&\Pr\left[b \leftarrow \{0,1\}; t \leftarrow X_n^b : D(t) = b\right] \\
&\quad = \frac{1}{2}\Pr\left[t \leftarrow X_n^1 \;:\; D(t) = 1\right] + \frac{1}{2}\Pr\left[t \leftarrow X_n^0 \;:\; D(t) \neq 1\right] \\
&\quad = \frac{1}{2}\Pr\left[t \leftarrow X_n^1 \;:\; D(t) = 1\right] + \frac{1}{2}\left(1 - \Pr\left[t \leftarrow X_n^0 \;:\; D(t) = 1\right]\right) \\
&\quad = \frac{1}{2} + \frac{1}{2}\left(\Pr\left[t \leftarrow X_n^1 \;:\; D(t) = 1\right] - \Pr\left[t \leftarrow X_n^0 \;:\; D(t) = 1\right]\right) \\
&\quad \geq \frac{1}{2} + \frac{\mu(n)}{2}
\end{aligned}$$

□

Example

Let $\{X_n\}_n$, $\{Y_n\}_n$ and $\{Z_n\}_n$ be pairwise *indistinguishable* probability ensembles, where X_n, Y_n, and Z_n are distributions over $\{0,1\}^n$. Assume further that we can efficiently sample from all three ensembles. Consider the n.u. p.p.t. machine $M(t)$ that samples $y \leftarrow Y_n$ where $n = |t|$ and outputs $t \oplus y$. Since $\{X_n\}_n \approx_c \{Z_n\}_n$, closure under efficient operations directly implies that

$$\{x \leftarrow X_n; y \leftarrow Y_n : x \oplus y\}_n \approx_c \{y \leftarrow Y_n; z \leftarrow Z_n : z \oplus y\}_n$$

3.2 Pseudo-randomness

Using the notion of computational indistinguishability, we next turn to defining pseudo-random distributions.

3.2.1 Definition of Pseudo-random Distributions

Let U_n denote the uniform distribution over $\{0,1\}^n$, i.e, $U_n = \{t \leftarrow \{0,1\}^n : t\}$. We say that a distribution is pseudo-random if it is indistinguishable from the uniform distribution.

Definition 57.1 (Pseudo-random Ensembles). The probability ensemble $\{X_n\}_n$, where X_n is a probability distribution over $\{0,1\}^{l(n)}$ for some polynomial $l(\cdot)$, is said to be *pseudorandom* if $\{X_n\}_n \approx \{U_{l(n)}\}_n$.

Note that this definition effectively says that a pseudorandom distribution needs to pass *all* efficiently computable statistical tests that the uniform distribution would have passesd; otherwise the statistical test would distinguish the distributions.

Thus, at first sight it might seem very hard to check or prove that a distribution is pseudorandom. As it turns out, there are *complete* statistical test; such a test has the property that if a distribution passes only that test, it will also pass all other efficient tests. We proceed to present such a test.

3.2.2 A complete statistical test: The next-bit test

We say that a distribution passes the *next-bit test* if no efficient adversary can, given any prefix of a sequence sampled from the distribution, predict the next bit in the sequence with probability significantely better than $\frac{1}{2}$ (recall that this was one of the test originally suggested in the introduction of this chapter).

Definition 58.1. An ensemble of probability distributions $\{X_n\}_n$ where X_n is a probability distribution over $\{0,1\}^{\ell(n)}$ for some polynomial $l(n)$ is said to pass the *Next-Bit Test* if for every non-uniform p.p.t. A, there exists a negligible function $\epsilon(n)$ s.t $\forall n \in N$ and $\forall i \in [0, \cdots, \ell(n)]$, it holds that

$$\mathsf{Pr}\left[t \leftarrow X_n : A(1^n, t_1 t_2 \ldots t_i) = t_{i+1}\right] < \frac{1}{2} + \epsilon(n).$$

Here, t_i denotes the i'th bit of t.

Remark 58.1. Note that we provide A with the additional input 1^n. This is simply allow A to have size and running-time that is polynomial in n and not simply in the (potentially) short prefix $t_0 \ldots t_i$.

Theorem 58.1 (Completeness of Next-Bit Test). *If a probability ensemble $\{X_n\}_n$ passes the next-bit test then $\{X_n\}_n$ is pseudo-random.*

Proof. Assume for the sake of contradiction that there exists a non-uniform p.p.t. distinguisher D, and a polynomial $p(\cdot)$ such that for infinitely many $n \in \mathbb{N}$, D distinguishes X_n and $U_{\ell(n)}$ with probability $\frac{1}{p(n)}$. We contruct a machine A that predicts the next bit of X_n for every such n. Define a sequence of *hybrid distributions* as follows.

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

Note that $H_n^0 = U_{\ell(n)}$ and $H_n^{\ell(n)} = X_n$. Thus, D distinguishes between H_n^0 and $H_n^{\ell(n)}$ with probability $\frac{1}{p(n)}$. It follows from the hybrid lemma that there exists some $i \in [0, \ell(n)]$ such that D distinguishes between H_n^i and H_n^{i+1} with probability $\frac{1}{p(n)\ell(n)}$. Recall, that the only difference between H^{i+1} and H^i is that in H^{i+1} the $(i+1)^{\text{th}}$ bit is x_{i+1}, whereas in H^i it is u_{i+1}. Thus, intuitively, D—given only the prefix $x_1 \ldots x_i$—can tell apart x_{i+1} from a uniformly chosen bit. This is turn means that D also can tell apart x_{i+1} from $\bar{x}_{i+1}$. More formally, consider the distribution $\tilde{H}_n^i$ defined as follows:

$$\tilde{H}_n^i = \left\{x \leftarrow X_n \;:\; u \leftarrow U_{\ell(n)} \;:\; x_0 x_1 \ldots x_{i-1} \bar{x}_i u_{i+1} \ldots u_{l(m)}\right\}$$

Note that H_n^i can be viewed as $\frac{1}{2}H_n^{i+1} + \frac{1}{2}\tilde{H}_n^{i+1}$. Substituting this identity into the last of term

$$\left|\Pr\left[t \leftarrow H_n^{i+1} \;:\; D(t) = 1\right] - \Pr\left[t \leftarrow H_n^i \;:\; D(t) = 1\right]\right|$$

yields

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

which simplifies to

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

Since D distinguishes H_n^i and H_n^{i+1} with probability $\frac{1}{p(n)\ell(n)}$, it distinguishes H_n^{i+1} and $\tilde{H}_n^{i+1}$ with probability $\frac{2}{p(n)\ell(n)}$. By the prediction lemma, there thus exists a machine A such that

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

where we let $H_n^{i+1,1}$ denote H_n^{i+1} and $H_n^{i+1,0}$ denote $\tilde{H}_n^{i+1}$. (i.e., A predicts whether a sample came from H_n^{i+1} or $\tilde{H}_n^{i+1}$.) We can now use A to construct a machine A' predicts x_{i+1} (i.e., the $(i+1)^{\text{th}}$ bit in the pseudorandom sequence):

- A' on input $(1^n, t_1 t_2 \ldots t_i)$ picks $\ell(n) - i$ random bits $u_{i+1} \ldots u_{\ell(n)} \leftarrow U^{\ell(n)-1}$, and lets $g \leftarrow A(t_1 \ldots t_i u_{i+1} \ldots u_{\ell(n)})$.
- If $g = 1$, it outputs u_{i+1}; otherwise it outputs $\bar{u}_{i+1} = 1 - u_{i+1}$.

Note that,

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

This concludes the proof Theorem 58.1. □

3.3 Pseudo-random generators

We now turn to definitions and constructions of pseudo-random generators.

3.3.1 Definition of a Pseudo-random Generators

Definition 60.1 (Pseudo-random Generator). A function $G : \{0,1\}^* \to \{0,1\}^*$ is a *Pseudo-random Generator (PRG)* if the following holds.

1. (efficiency): G can be computed in p.p.t.
2. (expansion): $|G(x)| > |x|$
3. (pseudo-randomness): The ensemble $\{x \leftarrow U_n : G(x)\}_n$ is pseudorandom.

3.3.2 An Initial Construction.

To provide some intuition for our construction, we start by considering a simplified construction (originally suggested by Adi Shamir). The basic idea is to iterate a one-way permutation and then output, in reverse order, all the intermediary values. More precisely, let f be a one-way permutation, and define the generator $G(s) = f^n(s) \parallel f^{n-1}(s) \parallel \ldots \parallel f(s) \parallel s$. (the $\parallel$ symbol here stands for string concatentation.)

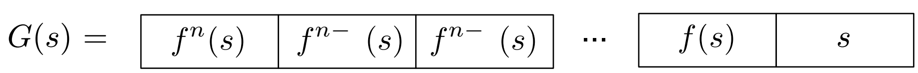


Figure 3.3: Shamir's proposed PRG

The idea behind the scheme is that given some prefix of the output of the generator, computing the next block is equivalent to inverting the one-way permutation f. Indeed, this scheme results in a sequence of unpredictable *numbers*, but not necessarily unpredictable *bits*. In particular, a one-way permutation may never "change" the first two bits of its input, and thus those corresponding positions will always be predictable.

The reason we need f to be a permutation, and not a general one-way function, is two-fold. First, we need the domain and range to be the same number of bits. Second, and more importantly, we require that the output of $f^k(x)$ be uniformly distributed if x is uniformly distributed. This holds if f is a permutation, but is not the case for a general one-way function.

As we shall see, this construction can be modified to generate also unpredictable bits. This is obtained through the concept of a *hard-core bit* which we introduce below.

3.3.3 Hard-core bits

Intuitively, a predicate b is *hard-core* for a OWF f if $b(x)$ cannot be predicted significantly better than with probability $1/2$, even given $f(x)$. In other words, although a OWF might leak many bits of its inverse, it does not leak the hard-core bits—in fact, it essentially does not leak *anything* about the hard-core bits (i.e., they are as unpredictable as if they were random).

Definition 61.1 (Hard-core Predicate). A predicate $h : \{0,1\}^* \to \{0,1\}$ is a hard-core predicate for $f(x)$ if h is efficiently computable given x, and $\forall$ nonuniform p.p.t. adversaries A, there exists a negligible ϵ so that $\forall k \in \mathbb{N}$

$$\Pr[x \leftarrow \{0,1\}^k \,:\, A(1^n, f(x)) = h(x)] \leq \frac{1}{2} + \epsilon(n)$$

The least significant bit of the RSA one-way function is known to be hard-core (under the RSA assumption). That is, given n,e, and $f_{RSA}(x) = x^e \bmod n$, there is no efficient algorithm that predicts LSB(x) given just $f_{RSA}(x)$. A few other examples include:

- The function half$_n(x)$ which is equal to 1 iff $0 \leq x \leq \frac{n}{2}$ is also hardcore for RSA, under the RSA assumption.
- The function half$_{p-1}(x)$ is a hardcore predicate for exponentiation to the power x mod p, under the DL assumption. (See §3.4.1 for this proof.)

We now show how hard-core predicates can be used to construct a PRG.

3.3.4 Constructions of a PRG

The idea is to use our initial generator (which only outputs unpredictable numbers, and not bits), but instead of outputting all intermediary values, we simply output a "hard-core" bit of each of them. We start by providing a construction of a PRG that only expands with one bits; the full construction is found in Corollary 63.1.

Theorem 61.1. *Let f be a one-way permutation, and h a hard-core predicate for f. Then $G(s) = f(s) \parallel h(s)$ is a PRG.*

Proof. Assume for contradiction that there exists a nonuniform p.p.t. adversary A and a polynomial $p(n)$ such that for infinitely many n, there exists an i such that A predicts the i^{th} bit with probability $\frac{1}{p(n)}$. Since the first n bits of $G(s)$ are a permutation of a uniform distribution (and thus also uniformly distributed), A must predict bit $n+1$ with advantage $\frac{1}{p(n)}$. Formally,

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

This contradicts the assumption that b is hard-core for f. We conclude that G is a PRG. □

3.3.5 Expansion of a PRG

The construction above only extends an n-bit seed to $n+1$ output bits. The following theorem show how a PRG that extends only 1 bit, can be used to create a PRG that extends a n-bit seed to $poly(n)$ output bits.

Lemma 62.1. *Let $G : \{0,1\}^n \rightarrow \{0,1\}^{n+1}$ be a PRG. For any polynomial ℓ, define $G' : \{0,1\}^n \rightarrow \{0,1\}^{\ell(n)}$ as follows (see Fig. 3.4):*

$$\begin{aligned} G'(s) &= b_1 \dots b_{\ell(n)} \textit{ where} \\ X_0 &\leftarrow s \\ X_{i+1} \parallel b_{i+1} &\leftarrow G(X_i) \end{aligned}$$

Then G' is a PRG.

Proof. Consider the following recursive definition of $G'(s) = G^m(s)$:

$$\begin{aligned} &G^0(s) = \varepsilon \\ &G^i(s) = b||G^{i-1}(x) \text{ where } x||b \leftarrow G(s) \end{aligned}$$

where ε denotes the empty string. Now, assume for contradiction that there exists a distinguisher D and a polynomial $p(\cdot)$ such that for infinitely many n, D distinguishes $\{U_{m(n)}\}_n$ and $\{G'(U_n)\}_n$ with probability $\frac{1}{p(n)}$.

Define the hybrid distributions $H_n^i = U_{m(n)-i}||G^i(U_n)$, for $i = 1 \dots m(n)$. Note that $H_n^0 = U_{m(n)}$ and $H_n^{m(n)} = G^{m(n)}(U_n)$. Thus, D distinguishes H_n^0 and $H_n^{m(n)}$ with probability $\frac{1}{p(n)}$. By the Hybrid Lemma, for each n, there

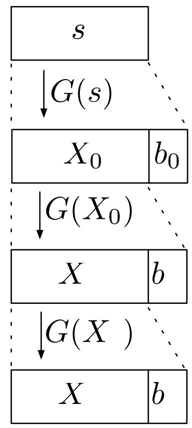


Figure 3.4: Illustration of the PRG G' that expands a seed of length n to $\ell(n)$. The function G is a PRG that expands by only 1 bit.

exist some i such that D distinguishes H_n^i and H_n^{i+1} with probability $\frac{1}{m(n)p(n)}$. Recall that,

$$\begin{aligned}
H_n^i &= U_{m-i} || G^i(U_n) \\
&= \{b_{prev} \leftarrow U_{m-i-1};\ b \leftarrow \{0,1\};\ b_{next} \leftarrow G^i(U_n) :\ b_{prev}||b||b_{next}\} \\
H_n^{i+1} &= U_{m-i-1} || G^{i+1}(U_n) \\
&= \{b_{prev} \leftarrow U_{m-i-1};\ x||b \leftarrow G(U_n);\ b_{next} = G^i(x) :\ b_{prev}||b||b_{next}\}
\end{aligned}$$

Consider the n.u. p.p.t $M(y)$ which samples and outputs as follows:

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

(Note that $M(y)$ is non-uniform as for each input length n, it needs to know the appropriate i.) Note that $M(U_{n+1}) = H_n^i$ and $M(G(U_n)) = H_n^{i+1}$. Since (by the PRG property of G) $\{U_{n+1}\}_n \approx \{G(U_n)\}_n$, it follows by closure under efficient operations that $\{H_n^i\}_n \approx \{H_n^{i+1}\}_n$, which is a contradiction. □

By combining Theorem 61.1 and Lemma 62.1, we get the final construction of a PRG.

Corollary 63.1. *Let f be a OWP and h a hard core bit for f. Then*

$$G(x) = h(x) \parallel h(f(x)) \parallel h(f^{(2)}(x)) \parallel \ldots \parallel h(f^{(n)}(x))$$

is a PRG.

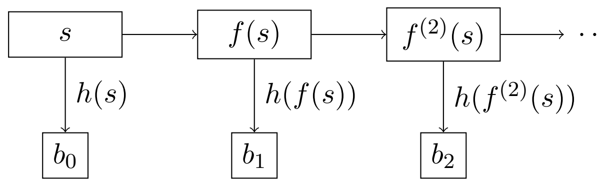


Figure 3.5: Illustration of a PRG based on a one-way permutation f and its hard-core bit h.

Proof. Let $G'(x) = f(x) \parallel h(x)$. By Theorem 61.1 G' is a PRG. Applying Lemma 62.1 to G' shows that G also is a PRG. □

Remark 63.1. Note that the above PRG can be computed in an "on-line" fashion. Namely, we only need to remember x_i to compute the continuation of the output. This makes it possible to compute an *arbitrary* long pseudo-random sequence using only a short seed of a fixed length. (In other words, we do not need to know an upper-bound on the length of the output when starting to generate the pseudo-random sequence.)

Remark 64.1. Furthermore, note that the PRG construction can be easily adapted to work also with a collection of OWP, and not just a OWP. If $\{f_i\}$ is a collection of OWP, simply consider G defined as follows:

$$G(r_1, r_2) = h_i(f_i(x)) \parallel h_i(f_i^{(2)}(x)) \parallel \dots$$

where r_1 is used to sample i and r_2 is used to sample x.

3.3.6 Concrete examples of PRGs

By using our concrete candidates of OWP (and their corresponding hard-core bits), we get the following concrete instantiations of PRGs.

- **Modular Exponentiation** (Blum-Micali PRG)
  - Use seed to generate p, g, x where p is a prime > 2, g is a generator for $\mathbb{Z}_p^*$, and $x \in \mathbb{Z}_p^*$.
  - Output $\mathit{half}_{p-1}(x) \parallel \mathit{half}_{p-1}(g^x \bmod p) \parallel \mathit{half}_{p-1}(g^{g^x} \bmod p) \parallel \dots$
- **RSA** (RSA PRG)
  - Use seed to generate p, q, e where p, q are random n-bit primes p, q, and e is a random element in Z_N^* where $N = pq$.

– Output $LSB(x) \parallel LSB(x^e \bmod N) \parallel LSB((x^e)^e \bmod N) \parallel \ldots$ where $LSB(x)$ is the least significant bit of x.

In all the above PRGs, we can in fact output $\log k$ bits at each iteration, while still remaining provably secure. Moreover, it is conjectured that it is possible to output $\frac{k}{2}$ bits at each iteration and still remain secure (but this has not been proven).

3.4 Hard-Core Bits from Any OWF

We have previously shown that if f is a one-way permutation and h is a hard-core predicate for f, then the function

$$G(s) = f(s) \parallel h(s)$$

is a pseudo-random generator. One issue, however, is how to find a hard-core predicate for a given one-way permutation. We have illustrated examples for some of the well-known permutations. Here, we show that every one-way function (permutation resp.) can be transformed into another one-way function (permutation resp.) which has a hard-core bit. Combined with our previous results, this shows that a PRG can be constructed from any one-way permutation.

As a warm-up, we show that half_{p-1} is a hardcore predicate for exponentiation mod p (assuming the DL assumption).

3.4.1 A Hard-core Bit from the Discrete Log problem

Recall that $\mathrm{half}_n(x)$ is equal to 1 iff $0 \leq x \leq \frac{n}{2}$.

Lemma 65.1. *Under the discrete log assumption (43.1), the function $half_{p-1}$ is a hard-core predicate for the exponentiation function $f_{p,g}(x) = g^x \mod p$ from the discrete-log collection DL.*

Proof. First, note that it is easy to compute $\mathrm{half}_{p-1}()$ given x. Suppose, for the sake of contradiction, that there exists some p.p.t. algorithm A and a polynomial $s(n)$ such that for infinitely many n

$$\Pr\left[f_{p,g} \leftarrow DL(n); x \leftarrow Z_p \,:\, A(1^n, f_{p,g}(x)) = \mathrm{half}_{(p-1)}(x)\right] > \frac{1}{2} + \frac{1}{s(n)}$$

It follows using an averaging argument (see the Proof of Theorem 68.1 for more details) that for a fraction $\frac{1}{2s(n)}$ of p, g, A predicts the hard-core bit with probabiliy $\frac{1}{2} + \frac{1}{2s(n)}$ only over the choice of x.

We show how to use the algorithm A to construct a new algorithm B which solves the discrete logarithm problem for those p, g and therefore violates the discrete log assumption. Let us first assume that A is always correct. Later we will remove this assumption. The algorithm B works as follows:

Algorithm 8: Solving Discrete Log using A

Input: (g, p, y)
Set $y_k \leftarrow y$ and $k = |p|$
while $k > 0$ **do**
    if y_k *is a square mod* p **then** $x_k \leftarrow 0$
    else
        $x_k \leftarrow 1$
        $y_k \leftarrow y_k/g \mod p$ to make a square
    Compute the square root $y_{k-1} \leftarrow \sqrt{y_k} \mod p$
    Run $b \leftarrow A(y_{k-1})$
    if $b = 0$ **then** $y_{k-1} \leftarrow -y_{k-1}$
    Decrement k
Return x

We first note that for every prime p, the square root operation can be performed efficiently. Here, we shall prove this for a special case of primes of the form $4k + 3$. The procedure can be generalized to handle any odd prime. (Show this!)

Theorem 66.1. *If $p = 3 + 4k$ and y is a square* $\text{mod} p$*, then* $\sqrt{y} = \pm y^{k+1}$.

Proof. If y is a square, then let $y = a^2 \mod p$. Thus,

$$(y^{k+1})^2 = a^{2(k+1)2} = a^{4k+4} = a^{p+1} = a^2 = y$$

□

Now if g is a generator, and y is square $y = g^{2x}$, notice that y has two square roots: g^x and $g^{x+p/2}$. (Recall that $g^{p/2} = -1 \bmod p$.) The first of these two roots has a smaller discrete log than y, and the other has a larger one. If it is possible to determine which of the two roots is the smaller root—say by using the adversary A to determine whether the exponent of the root is in the "top half" $[1, p/2]$ or not—then we can iteratively divide, take square-roots, and then choose the root with smaller discrete log until we eventually end up at 1. This is in fact the procedure given above.

Unfortunately, we are not guaranteed that A always outputs the correct answer, but only that A is correct noticably more often than not. In particular, A is correct with probability $\frac{1}{2}+\epsilon$ where $\epsilon = s(n)$. To get around this problem, we can use self-reducibility in the group $\mathbb{Z}_p$. In particular, we can choose ℓ random values $r_1, \ldots, r_\ell$, and randomize the values that we feed to A in line 8. Since A must be noticibly more correct than incorrect, we can use a "majority vote" to get an answer that is correct with high probability.

To formalize this intuition, we first demonstrate that the procedure above can be used to solve the discrete log for instances in which the discrete log is in the range $[1, 2, \ldots, \epsilon/4 \cdot p]$. As we will show later, such an algorithm suffices to solve the discrete log problem, since we can guess and map any instance into this range.

Consider the following alternative test in place of line 8. Let us now an-

8 $lo \leftarrow 0$
9 **for** $i = 1, 2, \ldots, \ell$ **do**
10 $\quad r_i \leftarrow \mathbb{Z}_p$
11 $\quad z_i \leftarrow y_{k-1} g^{r_i}$
12 $\quad b_i \leftarrow A(z_i)$
13 $\quad$ Increment lo if $(b_i = 0 \wedge r_i < p/2)$ or $(b_i = 1 \wedge r_i \geq p/2)$
14 Set $b = 0$ if $lo > \ell/2$ and 1 otherwise

alyze these lines when the discrete log of y_k is in the range $[1, 2, \ldots, s]$ where $s = \epsilon/4 \cdot p$. It follows that either the discrete log of $y_{k-1} \in [1, 2, \ldots, s/2]$ or in $[p/2, \ldots p/2 + s/2]$. In other words, the square root will either be slighly greater than 0, or slightly greater than $p/2$. Let the event noinc be the event that the counter lo is not incremented in line 13. By the Union Bound, we have that

$$\Pr\left[\mathsf{noinc} \mid low(y_{k-1})\right] \leq \Pr[A(z_i) \text{ errs}] + \Pr[r_i \in [p/2 - s, p/2] \cup [p - s, p - 1]]$$

The last term on the right arises from the error of re-randomizing. That is, when $r_i \in [p/2 - s, p/2]$, then the discrete log of z_i will be greater than $p/2$. Therefore, even if $\mathcal{A}$ answers correctly, the test on line 13 will not increment the counter lo. Although this error in unavoidable, since r_i is chosen randomly, we have that

$$\Pr[\mathsf{noinc} \mid low(y_{k-1})] \leq \frac{1}{2} - \epsilon + 2(\epsilon/4) = \frac{1}{2} - \frac{\epsilon}{2}$$

Conversely, the probability that lo is incremented is therefore greater than $1/2 + \epsilon/2$. By setting the number of samples $\ell = (1\epsilon)^2$, then by the corollary of the Chernoff bound given in Appendix **??**, then b will be correct correct with probability very close to 1.

□

In the above proof, we rely on specific properties of the OWP f. We proceed to show how the existence of OWPs implies the exitence of OWPs with a hard-core bit.

3.4.2 A General Hard-core Predicate from Any OWF

Let $\langle x, r\rangle$ denote the inner produce of x and r, i.e., $\sum x_i r_i$ mod 2. (In other words, r decides which bits of x to take parity on.)

Theorem 68.1. *Let f be a OWF (OWP). Then $f'(x,r) = f(x), r$ (where $|x| = |r|$) is a OWF (OWP) and $b(x,r) = \langle x, r\rangle$ is a hardcore predicate for f.*

3.4.3 *Proof of Theorem 68.1

Proof. We show that if $\mathcal{A}$, given $f'(x,r)$ can compute $b(x,r)$ with probability non-negligibly better than 1/2, then there exists a p.p.t. adversary $\mathcal{B}$ that inverts f. More precisely, we use $\mathcal{A}$ to construct a machine $\mathcal{B}$ that on input $y = f(x)$ recovers x with non-negligible probability, which contradicts the one-wayness of f. The proof is fairly involved. To provide intuition, we first consider two simplified cases.

Oversimplified case: assume $\mathcal{A}$ always computes $b(x,r)$ correctly. (Note that this is oversimplified as we only know that $\mathcal{A}$ computes $b(x,r)$ with probability non-negligibly better than 1/2.) In this case the following simple procedure recovers x: $\mathcal{B}$ on input y lets $x_i = \mathcal{A}(y, e_i)$ where $e_i = 00..010..$ is an n bit string with the only 1 being in position i, and outputs $x_1, x_2, \ldots, x_n$. This clearly works, since by definition $\langle x, e_i\rangle = x_i$ and by our assumption $\mathcal{A}(f(x), r) = \langle x, r\rangle$.

Less simplified case: assume $\mathcal{A}$ computes $b(x,r)$ w.p. $\frac{3}{4} + \frac{1}{poly(n)}$. Assume that $\mathcal{A}$ given a random $y = f(x)$ and random r, computes $b(x,r)$ w.p. $\frac{3}{4} + \epsilon$, where $\epsilon = \frac{1}{poly(n)}$. The above algorithm of simply querying $\mathcal{A}$ with y, e_i no longer work for two reasons:

1. $\mathcal{A}$ might not work for all y's,

2. even if $\mathcal{A}$ predicts $b(y, r)$ with high probabiliy for a given y, but a random r, it might still fail on the particular $r = e_i$.

To get around the first problem, we show that for a "reasonable" fraction of x's, $\mathcal{A}$ does work w.h.p. Let $S = \{x | \Pr[r \leftarrow \{0,1\}^n : \mathcal{A}(f(x), r) = b(x, r)] > \frac{3}{4} + \frac{\epsilon}{2}\}$. It is easy to see that $\Pr[x \in S] \geq \frac{\epsilon}{2}$; if not,

$$\begin{aligned}
&\Pr[x, r \leftarrow \{0,1\}^n : \mathcal{A}(f(x), r) = b(x, r)] \\
&\quad \leq \Pr[x \in S] \times 1 + \Pr[x \notin S] \times \Pr[\mathcal{A}(f(x), r) = b(x, r) | x \notin S] \\
&\quad < \frac{\epsilon}{2} + \frac{3}{4} + \frac{\epsilon}{2}
\end{aligned}$$

which is a contradiction.

The second problem is more subtle. To get around it, we "obfuscate" the queries y, e_i and rely on the linearity of $\langle a, b \rangle$. The following simple fact is useful.

Fact 69.1. $\langle a, b \oplus c \rangle = \langle a, b \rangle \oplus \langle a, c \rangle$ *mod* 2

Proof. $\langle a, b \oplus c \rangle = \Sigma a_i(b_i + c_i) = \Sigma a_i b_i + \Sigma a_i c_i = \langle a, b \rangle + \langle a, c \rangle \mod 2$ □

Now, rather than asking $\mathcal{A}$ to recover $\langle x, e_i \rangle$, we instead pick a random string r and ask $\mathcal{A}$ to recover $\langle x, r \rangle$ and $\langle x, r + e_1 \rangle$, and compute the XOR of the answers. If $\mathcal{A}$ correctly answers both queries we have recovered the i'th bit of x. More precisely, $\mathcal{B}(y)$ proceeds as follows:

ALGORITHM 69.1: $\mathcal{B}(y)$

$m \leftarrow \text{poly}(1/\epsilon)$
for $i = 1, 2, \ldots, n$ **do**
  for $j = 1, 2, \ldots, m$ **do**
    Pick random $r \leftarrow \{0,1\}^n$
    Set $r' \leftarrow e_i \oplus r$
    Compute a guess g_j for x_i as $\mathcal{A}(y, r) \oplus \mathcal{A}(y, r')$
  end for
  Set x_i to the majority of the guesses $g_1, \ldots, g_m$.
end for
Output $x_1, \ldots, x_n$.

Note that for a "good" x (i.e., $x \in S$) it holds that:

- w.p. at most $\frac{1}{4} - \frac{\epsilon}{2}$, $\mathcal{A}(y, r) \neq b(x, r)$
- w.p. at most $\frac{1}{4} - \frac{\epsilon}{2}$, $\mathcal{A}(y, r') \neq b(x, r)$

It follows by the union bound that w.p. at least $\frac{1}{2}+\epsilon$ both answers of $\mathcal{A}$ are correct. Since $\langle y, r\rangle + \langle y, r'\rangle = \langle y, r \oplus r'\rangle = \langle y, e_i\rangle$, each guess g_i is correct w.p. $\frac{1}{2}+\epsilon$. Since we do $poly(1/\epsilon)$ independent guesses and finally take a majority vote, it follows using the Chernoff Bound that every bit is x_i computed by $\mathcal{A}$ is correct with high probability. Thus, for a non-negligible fraction of x's, $\mathcal{B}$ inverts f, which is a contradiction.

The general case. We proceed to the most general case. Here, we simply assume that $\mathcal{A}$, given random $y = f(x)$, random r computes $b(x, r)$ w.p. $\frac{1}{2}+\epsilon$ (where $\epsilon = \frac{1}{poly(n)}$). As before, let $S = \{x | \Pr[\mathcal{A}(f(x), r) = b(x, r)] > \frac{1}{2} + \frac{\epsilon}{2}\}$. It again follows that $\Pr[x \in S] \geq \frac{\epsilon}{2}$. To construct $\mathcal{B}$, assume for starters that $\mathcal{B}$ has access to "magical" oracle C that given $f(x)$ gives us samples

$$\langle x, r_1\rangle, r_1 \cdots \langle x, r_n\rangle, r_n$$

where $r_1, \ldots, r_n$ are independent and random. Consider the following procedure $\mathcal{B}(y)$:

ALGORITHM 70.1: $\mathcal{B}(y)$ FOR THE GENERAL CASE

$m \leftarrow \text{poly}(1/\epsilon)$
for $i = 1, 2, \ldots, n$ **do**
  $C(y) \rightarrow (b_1, r_1), \ldots, (b_m, r_m)$
  Let $r'_j = e_i \oplus r_j$
  Compute $g_j = b_j \oplus \mathcal{A}(y, r')$
  Let $x_i \leftarrow majority(g_1, \ldots, g_m)$
end for
Output $x_1, \ldots, x_n$.

Given an $x \in S$, it follows that each guess g_i is correct w.p. $\frac{1}{2} + \frac{\epsilon}{2} = \frac{1}{2} + \epsilon'$. We can now again apply the the Chernoff bound to show that x_i is wrong w.p. $\leq 2^{-\epsilon'^2 m}$. Thus, as long as $m >> \frac{1}{\epsilon'^2}$, we can recover all x_i. The only problem is that $\mathcal{B}$ uses the magical oracle C.

From here on, the idea is to eliminate C from the constructed machine. As an intermediate step, assume that C gives us samples $\langle x, r_1\rangle, r_1; \ldots; \langle x, r_n\rangle, r_n$ which are only *pairwise independent* (instead of being completely independent). It follows by the Pairwise-Independent Sampling inequality that each x_i is wrong w.p. $\leq \frac{1-4\epsilon'^2}{4m\epsilon'^2} \leq \frac{1}{m\epsilon'^2}$. By union bound, any of the x_i is wrong w.p. $\leq \frac{n}{m\epsilon'^2} \leq \frac{1}{2}$, when $m \geq \frac{2n}{\epsilon'^2}$. Thus, if we could get $\frac{2n}{\epsilon'^2}$ pairwise independent samples, we would be done. So, where can we get them from?

An initial attempt to remove C would be to pick $r_1, \ldots, r_m$ at random and *guess* $b_1, \ldots, b_m$ randomly. However, b_i would be correct only w.p. 2^{-m}. A better idea is to pick $log(m)$ samples $s_1, \ldots, s_{log(m)}$ and guessing $b'_1, \ldots, b'_{log(m)}$; here the guess is correct with probability $1/m$. Now, generate $r_1, r_2, \ldots, r_{m-1}$ as all possible sums (mod 2) of subsets of $s_1, \ldots, s_{log(m)}$, and $b_1, b_2, \ldots, b_m$ as the corresponding subsets of b'_i. That is,

$$\begin{aligned} r_i &= \sum_{j \in I_i} s_j \quad j \in I \text{ iff } i_j = 1 \\ b_i &= \sum_{j \in I_i} b'_j \end{aligned}$$

It is not hard to show that these r_i are pairwise independent samples (show this!). Yet w.p. $1/m$, all guesses for $b'_1, \ldots, b'_{log(m)}$ are correct, which means that $b_1, \ldots, b_{m-1}$ are also correct.

Thus, for a fraction of ϵ' of x' it holds that w.p. $1/m$ we invert f. w.p. $1/2$. That is, $\mathcal{B}$ inverts f w.p.

$$\frac{\epsilon'}{2m} = \frac{\epsilon'^3}{4n} = \frac{(\epsilon/2)^3}{4n}$$

when $m = \frac{2n}{\epsilon^2}$. This contradicts the one-wayness of f.

3.5 Secure Encryption

We next use the notion of indistinguishability to provide a computational definition of security of encryption schemes. As we shall see, the notion of a PRG will be instrumental in the construction of encryption schemes which permit the use of a *short key* to encrypt a long message.

The intuition behind the definition of secure encryption is simple: instead of requiring that encryptions of any two messages are identically distributed (as in the definition of perfect secrecy), the computational notion of secure encryption requires only that encryptions of any two messages are indistinguishable.

Definition 71.1 (Secure Encryption). The encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is said to be *single-message secure* if $\forall$ non uniform p.p.t D, there exists a negligible function $\epsilon(\cdot)$ such that for all $n \in \mathbb{N}, m_0, m_1 \in \{0,1\}^n$, D distinguishes between the the following distributions with probability at most $\epsilon(n)$:

- $\{k \leftarrow \mathsf{Gen}(1^n) : \mathsf{Enc}_k(m_0)\}$
- $\{k \leftarrow \mathsf{Gen}(1^n) : \mathsf{Enc}_k(m_1)\}$

3.6 An Encryption Scheme with Short Keys

Recall that perfectly secure encryption schemes require a key that is at least long as the message to be encrypted. In this section we show how a short key can be use to get a secure encryption scheme. The idea is to simply use the one-time pad encryption scheme, but where the pad is only pseudorandomness (instead of being truly random.) Since we know how to take a small seed and construct a long pseudorandom sequence (i.e., using a PRG), we can encrypt long messages with a short key.

More precisely, consider the following encryption scheme. Let $G(s)$ be a length-doubling pseudo-random generator.

ALGORITHM 72.1: ENCRYPTION SCHEME FOR n-BIT MESSAGE

$\mathsf{Gen}(1^n)$: $k \leftarrow U_{n/2}$

$\mathsf{Enc}_k(m)$: Output $m \oplus G(k)$

$\mathsf{Dec}_k(c)$: Output $c \oplus G(k)$

Theorem 72.1. $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *from Algorithm 72.1 is single-message secure.*

Proof. Assume for contradiction that there a distinguisher D and a polynomial $p(n)$ such that for infinitely many n, there exists messages m_n^0, m_n^1 such that D distinguishes between

- $\{k \leftarrow \mathsf{Gen}(1^n) : \mathsf{Enc}_k(m_0)\}$
- $\{k \leftarrow \mathsf{Gen}(1^n) : \mathsf{Enc}_k(m_1)\}$

with probability $1/p(n)$. Consider the following hybrid distributions:

- H_n^1 (Encryption of m_n^0): $\{s \leftarrow Gen(1^n) : m_n^0 \oplus G(s)\}$.
- H_n^2 (OTP with m_n^1): $\{r \leftarrow U_n : m_n^0 \oplus r\}$.
- H_n^3 (OTP with m_n^1): $\{r \leftarrow U_n : m_n^1 \oplus r\}$.
- H_n^4 (Encryption of m_n^1): $\{s \leftarrow Gen(1^n) : m_n^1 \oplus G(s)\}$.

By construction D distringuishes H_n^1 and H_n^4 with probability $1/p(n)$ for infinitely many n. It follows by the hybrid lemma that D also distinguishes two consequetive hybrids with probability $1/4p(n)$ (for infinitely many n). We show that this is a contradiction.

- Consider the n.u. p.p.t. machine $M^i(x) = m^i_{|x|} \oplus x$[1] and the distribution $X_n = \{s \leftarrow U_{\frac{n}{2}} : G(s)\}$. By definition, $H^1_n = M^0(X_n)$, $H^4_n = M^1(X_n)$ and $H^2_n = M^0(U_n)$, $H^3_n = M^1(U_n)$. But since $\{X_n\}_n \approx \{U_n\}_n$ (by the PRG property of G) it follows by closure under efficient operations that $\{H^1_n\}_n \approx \{H^2_n\}_n$ and $\{H^3_n\}_n \approx \{H^4_n\}_n$.

- Additionally, by the perfect secrecy of the OTP, H^2_n and H^3_n are identically distributed.

Thus, all consequetive hybrid distributions are indistinguishable, which is a contradiction. □

3.7 Multi-message Secure Encryption

As suggested by the name, *single-message* secure encryption (Definition 71.1), only considers the security of an encryption scheme that is used to encrypt a single message. In general, we would like to encrypt many messages, and still require that the adversary cannot learn anything about the messages.

The following definition extends single-message security to multi-message security. The definition is identical, with the only exception being that we require that the encryptions of any two *vectors* or messages are indistinguishable.

Definition 73.1 (Multi-message Secure Encryption). An encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is said to be *multi-message secure* if $\forall$ non uniform p.p.t D, $\forall$ polynomial $q(n)$, there exists a negligible function $\epsilon(\cdot)$ such that for all $n \in N, m_0, m_1, \ldots, m_{q(n)}, m'_0, m'_1, \ldots, m'_{q(n)} \in \{0,1\}^n$, D distinguishes between the the following distributions with probability at most $\epsilon(n)$:

- $\{k \leftarrow \mathsf{Gen}(1^n) : \mathsf{Enc}_k(m_0), \mathsf{Enc}_k(m_1), \ldots \mathsf{Enc}_k(m_{q(n)})\}$
- $\{k \leftarrow \mathsf{Gen}(1^n) : \mathsf{Enc}_k(m'_0), \mathsf{Enc}_k(m'_1), \ldots \mathsf{Enc}_k(m'_{q(n)})\}$

It is easy to see that the single-message secure encryption scheme from the previous section (i.e., $Enc_k(m) = m \oplus G(s)$, where G is a PRG) is not multi-message secure. More generally,

Theorem 73.1. *There does not exist deterministic stateless multi-message secure secure encryption schemes.*

[1]Note that M^i is *non-uniform* as for each input lenght n, it has the message m^i_n hard-coded.

Proof. For any two messages m_0, m_1, consider the encryptions (c_0, c_1), (c_0', c_1') of the messages (m_0, m_0) and (m_0, m_1). If the encryption scheme is deterministic and stateless, $c_0 = c_1$, but $c_0' \neq c_1$. □

So, to get a multi-messages secure scheme (that is stateless) we need to develop a probabilistic scheme. One idea for such a scheme is to pick a random string r, then output $r||m \oplus f(r)$ for some function f. Ideally, we'd like the output of f to be a random string as well. One way to get such an f might be to have a long pseudorandom sequence of length on the order of $n2^n$. Then f could use r as an index into this sequence and return the n bits at r. But no pseudorandom generator can produce an exponential number of bits; our construction only works for pseudorandom generators with polynomial expansion.

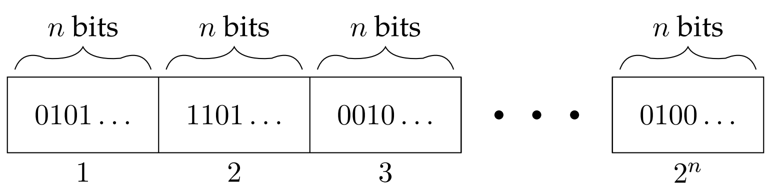
If we were to use a pseudorandom generator, then r could be at most $O(\log n)$ bits long, so even if r is chosen randomly, we would end up choosing two identical values of r with reasonable probability; this scheme would not be multi-message secure, though a stateful scheme that keeps track of the values of r used could be. The ideas is to introduce a new type of "pseudorandom" function that allows us to index an exponentially long pseudorandom string.

3.8 Pseudorandom Functions

Before defining pseudorandom function, we first recall the definition of a random function.

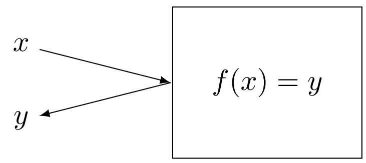
3.8.1 Random Functions

The scheme $r||m \oplus f(r)$ would be multi-message secure if f were a random function. We can describe a random functions in two different ways: a combinatorial description—as a random function table—and compuational description—as a machine that randomly chooses outputs given inputs and keeps track of its previous answers. In the combinatorial description, the random function table can be view as a long array that stores the values of f. So, $f(x)$ returns the value at position nx.



Note that the description length of a random function is $n2^n$, so there are 2^{n2^n} random functions from $\{0,1\}^n \to \{0,1\}^n$. Let RF_n be the distribution that picks a function mapping $\{0,1\}^n \to \{0,1\}^n$ uniformly at random.

A computational description of a random function is instead as follows: a random function is a machine that upon receiving input x proceeds as follows. If it has not seen x before, it chooses a value $y \leftarrow \{0,1\}^n$ and returns y; it then records that $f(x) = y$. If it has seen x before, then it looks up x, and outputs the same value as before.



It can be seen that both of the above descriptions of a random functions give rise to identical distributions.

The problem with random functions is that (by definition) they have a long description length. So, we cannot employ a random function in our encryption scheme. We will next define a *pseudorandom* function, which mimics a random function, but has a short description.

3.8.2 Definition of Pseudorandom Functions

Intuitively, a pseudorandom function (PRF) "looks" like a random function to any n.u. p.p.t. adversary. In defining this notion, we consider an adversary that gets *oracle* access to either the PRF, or a truly random function, and is supposed to decide which one it is interacting with. More precisely, an oracle Turing machine M is a Turing machine that has been augmented with a component called an *oracle*: the oracle receives requests from M on a special tape and writes its responses to a tape in M. We now extend the notion of indistinguishability of distributions, to indistinguishability of distributions of oracles.

Definition 75.1 (Oracle Indistinguishability). Let $\{O_n\}_{n\in N}$ and $\{O'_n\}_n$ be ensembles of probability distributions where O_n, O'_n are probability distributions over functions $\{0,1\}^{l_1(n)} \to \{0,1\}^{l_2(n)}$ for some polynomials $l_1(\cdot), l_2(\cdot)$. We say that $\{O_n\}_n$ and $\{O'_n\}_n$ are *computationally indistinguishable* (abbr. $\{O'_n\}_n \approx \{O'_n\}_{n\in N}$) if for all non-uniform p.p.t oracles machines D, there exists a negligible function $\epsilon(\cdot)$ such that $\forall n \in N$

$$\left|\mathsf{Pr}\left[F \leftarrow O_n, D^{F(\cdot)}(1^n) = 1\right] - \mathsf{Pr}\left[F \leftarrow O'_n, D^{F(\cdot)}(1^n) = 1\right]\right| < \epsilon(n).$$

It is easy to verify that oracle indistinguishability satisfies "closure under efficient operations", the Hybrid Lemma, and the Prediction Lemma.

We turn to define pseudorandom functions.

Definition 76.1 (Pseudo-random Function). A family of functions $\{f_s : \{0,1\}^{|s|} \to \{0,1\}^{|s|}\}_{s \in \{0,1\}^*}$ is *pseudo-random* if

- (Easy to compute): $f_s(x)$ is p.p.t computable (given s and x)
- (Pseudorandom): $\{s \leftarrow \{0,1\}^n : f_s\}_n \approx \{F \leftarrow \mathsf{RF}_n : F\}_n$.

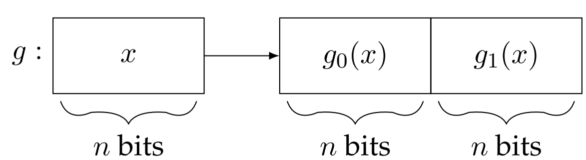
Remark 76.1. Note that in the definition of a PRF, it is critical that the seed s to the PRF is not revealed; otherwise it is easy to distinguish f_s from a random function: simply ask the oracle a random query x and check whether the oracle's reply equals $f_s(x)$.

Remark 76.2. Also note that the number of pseudorandom functions is much smaller than the number of random function (for the same input lenghts); indeed all pseudorandom functions have a short description, whereas random functions in general do not.

We have the following theorem.

Theorem 76.1. *If there exists a pseudorandom generator, then there exists a pseudorandom function.*

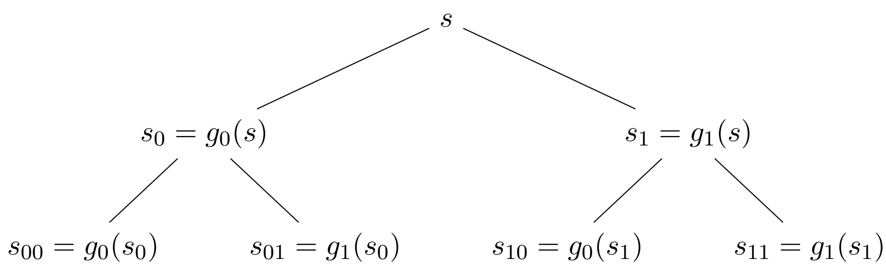
Proof. We have already shown that any pseudorandom generator g is sufficient to construct a pseudorandom generator g' that has polynomial expansion. So, without loss of generality, let g be a length-doubling pseudorandom generator.



Then we define f_s as follows to be a pseudorandom function:

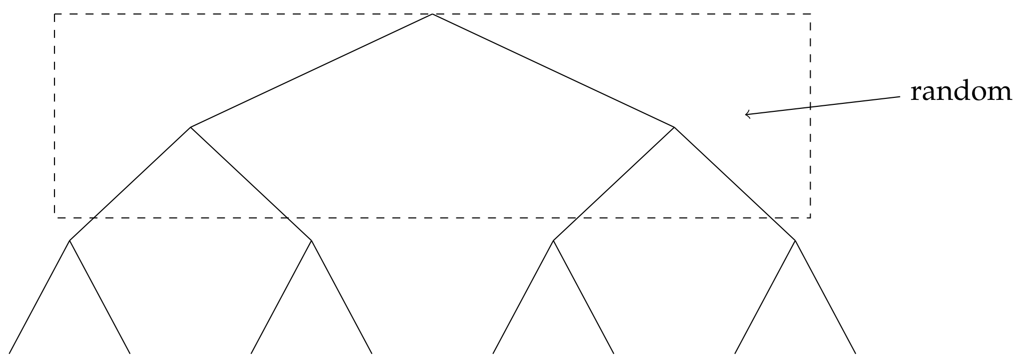
$$f_s(b_1 b_2 \ldots b_n) = g_{b_n}(g_{b_{n-1}}(\cdots(g_{b_1}(s))\cdots))$$

f keeps only one side of the pseudorandom generator at each of n iterations. Thus, the possible outputs of f for a given input form a tree; the first three levels are shown in the following diagram. The leaves of the tree are the output of f.



The intuition about why f is a pseudorandom function is that a tree of height n contains 2^n leaves, so exponentially many values can be indexed by a single function with n bits of input. Thus, each unique input to f takes a unique path through the tree. The output of f is the output of a pseudorandom generator on a random string, so is random.

One approach to the proof is to look at the leaves of the tree. Build a sequence of hybrids by successively replacing each leaf with a random distribution. This approach does not work, since the hybrid lemma does not apply when there are exponentially many hybrids. Instead, we form hybrids by replacing successive levels of the tree: hybrid HF_n^i is formed by picking all levels through the ith uniformly at random, then applying the tree construction as before.



Note that $\mathsf{HF}_n^1 = \{s \leftarrow \{0,1\}^n : f_s(\cdot)\}$ (picking only the seed at random), which is the distribution defined originally. Further, $\mathsf{HF}_n^n = \mathsf{RF}_n$ (picking the leaves at random).

Thus, if $\mathcal{D}$ can distinguish $F \leftarrow \mathsf{RF}_n$ and f_s for a randomly chosen s, then $\mathcal{D}$ distinguishes $F_1 \leftarrow \mathsf{HF}_n^1$ and $F_n \leftarrow \mathsf{HF}_n^n$ with probability ϵ. By the hybrid lemma, there exists some i such that $\mathcal{D}$ distinguishes HF_n^i and HF_n^{i+1} with probability ϵ/n.

The difference between HF_n^i and HF_n^{i+1} is that level $i+1$ in HF_n^i is $g(U_n)$, whereas in HF_n^{i+1}, level $i+1$ is U_n. Afterwards, both distributions continue to use g to construct the tree.

To finish the proof, we will construct one more set of hybrid distributions. Recall that there is some polynomial $p(n)$ such that the number of queries made by $\mathcal{D}$ is bounded by $p(n)$. So, we can now apply the first hybrid idea suggested above: define hybrid HHF_n^j that picks F from HF_n^i, and answer the first j new queries using F, then answer the remaining queries using HF_n^{i+1}.

But now there are only $p(n)$ hybrids, so the hybrid lemma applies, and $\mathcal{D}$ can distinguish HHF_n^j and HHF_n^{j+1} for some j with probability $\epsilon/(np(n))$. But HHF_n^j and HHF_n^{j+1} differ only in that HHF_n^{j+1} answers its $j+1$st query with the output of a pseudorandom generator on a randomly chosen value, whereas HHF_n^j answers its $j+1$st query with a randomly chosen value. As queries to HHF_n^j can be emulated in p.p.t (we here rely on the equivalence between the combinatorial and the computational view of a random function; we omit the details), it follows by closure under efficient operations that $\mathcal{D}$ contradicts the pseudorandom property of g. □

3.9 Construction of Multi-message Secure Encryption

The idea behind our construction is to use a pseudorandom function in order to pick a separate random pad for every message. In order to make decryption possible, the ciphertext contains the input on which the pseudo-random function is evaluated.

ALGORITHM 78.1: MANY-MESSAGE ENCRYPTION SCHEME

Assume $m \in \{0,1\}^n$ and let $\{f_k\}$ be a PRF family.

$\mathsf{Gen}(1^n) : k \leftarrow U_n$

$\mathsf{Enc}_k(m)$: Pick $r \leftarrow U_n$. Output $(r, m \oplus f_k(r))$

$\mathsf{Dec}_k((r,c))$: Output $c \oplus f_k(r)$

Theorem 78.1. *(*Gen, Enc, Dec*) is a many-message secure encryption scheme.*

Proof. Assume for contradiction that there exists a n.u. p.p.t. distinguisher D, and a polynomial $p(\cdot)$ such that for infinitely many n, there exists messages $\bar{m} = \{m_0, m_1 \ldots, m_{q(n)}\}$ and $\bar{m'} = \{m'_0, m'_1 \ldots, m'_{q(n)}\}$ such that D distinguishes between encryptions of $\bar{m}$ and $\bar{m'}$ w.p. $\frac{1}{p(n)}$. (To simplify notation, here—and in subsequent proofs—we sometimes omit n from our notation and let m_i denote m_n^i, whenever n is clear from the context). Consider the following sequence of hybrid distributions: (As above, it should be clear that H_i denotes H_n^i).

- H_1 – real encryptions of $m_0, m_1 \ldots, m_{q(n)}$

  $$\{s \leftarrow \{0,1\}^n, r_0, \ldots, r_{q(n)} \leftarrow \{0,1\}^n : r_0||m_0 \oplus f_s(r_0), \ldots m_{q(n)} \oplus f_s(r_{q(n)})\}$$

  This is precisely what the adversary sees when receiving the encryptions of $m_0, \ldots m_{q(n)}$.

- H_2 – using a truly random function instead of f on $m_0, m_1, \ldots, m_{q(n)}$

  $$\{R \leftarrow RF_n; r_0, \ldots, r_{q(n)} \leftarrow \{0,1\}^n : r_0||m_0 \oplus R(r_0), \ldots m_{q(n)} \oplus R(r_{q(n)})\}$$

- H_3 – using OTP on $m_0, m_1, \ldots, m_{q(n)}$

  $$\{p_0 \ldots p_{q(n)} \leftarrow \{0,1\}^n; r_0, \ldots, r_{q(n)} \leftarrow \{0,1\}^n : r_0||m_0 \oplus p_0, \ldots m_{q(n)} \oplus p_{q(n)}\}$$

- H_4 – using OTP on $m'_0, m'_1, \ldots, m'_{q(n)}$

  $$\{p_0 \ldots p_{q(n)} \leftarrow \{0,1\}^n; r_0, \ldots, r_{q(n)} \leftarrow \{0,1\}^n : r_0||m'_0 \oplus p_0, \ldots m'_{q(n)} \oplus p_{q(n)}\}$$

- H_5 – using a truly random function instead of f on $m'_0, m'_1, \ldots, m'_{q(n)}$

  $$\{R \leftarrow \{\{0,1\}^n \rightarrow \{0,1\}^n\}; r_0, \ldots, r_{q(n)} \leftarrow \{0,1\}^n : r_0||m'_0 \oplus R(r_0), \ldots m'_{q(n)} \oplus R(r_{q(n)})\}$$

- H_6 – real encryptions of $m'_0, m'_1 \ldots, m'_{q(n)}$

  $$\{s \leftarrow \{0,1\}^n, r_0, \ldots, r_{q(n)} \leftarrow \{0,1\}^n : r_0||m'_0 \oplus f_s(r_0), \ldots m'_{q(n)} \oplus f_s(r_{q(n)})\}$$

By the Hybrid Lemma, D distinguishes between to adjacent hybrid distributions with inverse polynomial probability (for infinitely many n). We show that this is a contradiction:

- First, note that D distinguish between H_1 and H_2 only with negligible probability; otherwise (by closure under efficient operations) we contradict the pseudorandomness property of $\{f_s\}_n$.

  The same argument applies for H_6 and H_5.

- H_2 and H_3 are "almost" identical except for the case when $\exists i, j$ such that $r_i = r_j$, but this happens only with probability

$$\frac{\binom{q(n)}{2}}{2^{-n}},$$

which is negligible; thus, D can distinguishes between H_2 and H_3 only with negligible probability. The same argument applies for H_4 and H_5.

- Finally, H_3 and H_4 are identical by the perfect secrecy of the OTP.

This contradicts that D distinguishes two adjacent hybrids. □

3.10 Public Key Encryption

So far, our model of communication allows the encryptor and decryptor to meet in advance and agree on a secret key which they later can use to send private messages. Ideally, we would like to drop this requirement of meeting in advance to agree on a secret key. At first, this seems impossible. Certainly the decryptor of a message needs to use a secret key; otherwise, nothing prevents the eavesdropper from running the same procedure as the decryptor to recover the message. It also seems like the encryptor needs to use a key because otherwise the key cannot help to decrypt the cyphertext.

The flaw in this argument is that the encrypter and the decryptor need not share the *same* key, and in fact this is how public key cryptography works. We split the key into a secret decryption key sk and a public encryption key pk. The public key is published in a secure repository, where anyone can use it to encrypt messages. The private key is kept by the recipient so that only she can decrypt messages sent to her.

We define a public key encryption scheme as follows:

Definition 80.1 (Public Key Encryption Scheme). A triple $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a public key encryption scheme if

1. $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ is a p.p.t. algorithm that produces a key pair (pk, sk)
2. $c \leftarrow \mathsf{Enc}_{pk}(m)$ is a p.p.t. algorithm that given pk and $m \in \{0,1\}^n$ produces a ciphertext c.
3. $m \leftarrow \mathsf{Dec}_{sk}(c)$ is a p.p.t. algorithm that given a ciphertext c and secret key sk produces a message $m \in \{0,1\}^n \cup \perp$.
4. There exists a polynomial-time algorithm M that on input $1^n, i$ outputs the ith length n message (if such a message exists).

5. For all $n \in N, m \in \{0,1\}^n$

$$\mathsf{Pr}\left[(pk, sk) \leftarrow \mathsf{Gen}(1^n) : \mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) = m\right] = 1$$

We allow the decryption algorithm to produce a special symbol $\perp$ when the input ciphertext is "undecipherable." The security property for public-key encryption can be defined using an experiment similar to the ones used in the definition for secure private key encryption.

Definition 81.1 (Secure Public Key Encryption). The encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is said to be *secure* if $\forall$ non uniform p.p.t D, there exists a negligible function $\epsilon(\cdot)$ such that for all $n \in N, m_0, m_1 \in \{0,1\}^n$, D distinguishes between the the following distributions with probability at most $\epsilon(n)$:

- $\{(pk, sk) \leftarrow \mathsf{Gen}(1^n) : (pk, \mathsf{Enc}_{pk}(m_0))\}$
- $\{(pk, sk) \leftarrow \mathsf{Gen}(1^n) : (pk, \mathsf{Enc}_{pk}(m_1))\}$

With this definitions, there are some immediate impossibility results:

Perfect secrecy Perfect secrecy is not possible (even for small message spaces) since an unbounded adversary could simply encrypt every message in $\{0,1\}^n$ with every random string and compare with the challenge ciphertext to learn the underlying message.

Deterministic encryption It is also impossible to have a deterministic encryption algorithm because otherwise an adversary could simply encrypt and compare the encryption of m_0 with the challenge ciphertext to distinguish the two experiments.

Remark 81.1. As with the case of private-key encryption, we can extend the definition to multi-message security. Fortunately, for the case of public-key encryption, multi-message security is equivalent to single-messages security. This follows by a simple application of the hybrid lemma, and closure under efficient operations; the key point here is that we can efficiently generate encryptions of any message, without knowing the secret key (this was not possible, in the case of private-key encryption). We leave it as an exercise to the reader to complete the proof.

Remark 81.2. We can consider a weaker notion of "single-bit" secure encryption where we only require that encryptions of 0 and 1 are indistinguishable. Any single-bit secure encryption can be turned into a secure encryption scheme by simply encryption each bit of the message using the single-bit secure encryption; the security of the new scheme follows directly from the multi-message security (which is equivalent to traditional security) of the single-bit secure encryption scheme.

3.10.1 Constructing a PK encryption system

Trapdoor permutations seem to fit the requirements for a public key cryptosystem. We could let the public key be the index i of the function to apply, and the private key be the trapdoor t. Then we might consider $Enc_i(m) = f_i(m)$, and $Dec_{i,t}(c) = f_i^{-1}(c)$. This makes it easy to encrypt, and easy to decrypt with the public key, and hard to decrypt without. Using the RSA function defined in Theorem 45.1, this construction yields the commonly used RSA cryptosystem.

However, according to our definition, this construction does not yield a secure encryption scheme. In particular, it is deterministic, so it is subject to comparison attacks. A better scheme (for single-bit messages) is to let $\mathsf{Enc}_i(x) = \{r \leftarrow \{0,1\}^n : \langle f_i(r), b(r) \oplus m\rangle\}$ where b is a hardcore bit for f. As we show, the scheme is secure, as distinguishing encryptions of 0 and 1 essentially requires predicting the hardcore bit of a one-way permutation.

ALGORITHM 82.1: 1-BIT SECURE PUBLIC KEY ENCRYPTION

$\mathsf{Gen}(1^n) : (f_i, f_i^{-1}) \leftarrow \mathsf{Gen}_T(1^n)$. Output $(pk, sk) \leftarrow ((f_i, b_i), (b_i, f_i^{-1}))$.

$\mathsf{Enc}_{pk}(m)$: Pick $r \leftarrow \{0,1\}^n$. Output $(f_i(r),\ b_i(r) \oplus m)$.

$\mathsf{Dec}_{sk}(c_1, c_2)$: Compute $r \leftarrow f_i^{-1}(c_1)$. Output $b_i(r) \oplus c_2$.

Here, $(f_i, f^{-1})_{i \in I}$ is a family of one-way trapdoor permutations and b_i is the hard-core bit corresponding to f_i. Let Gen_T be the p.p.t that samples a trapdoor permutation index from I.

Theorem 82.1. *If trapdoor permutations exist, then scheme 82.1 is a secure single-bit public-key encryption system.*

Proof: As usual, assume for contradiction that there exists a n.u. p.p.t. D and a polynomial $p(\cdot)$, such that D distinguishes $\{(pk, sk) \leftarrow \mathsf{Gen}(1^n) : (pk, \mathsf{Enc}_{pk}(0))\}$ and $\{(pk, sk) \leftarrow \mathsf{Gen}(1^n) : (pk, \mathsf{Enc}_{pk}(1))\}$ w.p. $\frac{1}{p(n)}$ for infinitely many n. By the prediction lemma, there exist a machine A such that

$$\Pr\left[m \leftarrow \{0,1\}; (pk, sk) \leftarrow \mathsf{Gen}(1^n) \ : \ D(pk, \mathsf{Enc}_{pk}(m)) = m\right] \geq \frac{1}{2} + \frac{1}{2p(n)}$$

We can now use A to construct a machine A' that predicts the hard-core predicate $b(\cdot)$:

- A' on input (pk, y) picks $c \leftarrow \{0,1\}$, $m \leftarrow A(pk, (y, c))$, and outputs $c \oplus m$.

Note that,

$$\begin{aligned}
&\Pr\left[(pk, sk) \leftarrow \mathsf{Gen}(1^n); r \leftarrow \{0,1\}^n \;:\; A'(pk, f_{pk}(r)) = b(r)\right] \\
&\quad = \Pr\left[(pk, sk) \leftarrow \mathsf{Gen}(1^n); r \leftarrow \{0,1\}^n; c \leftarrow \{0,1\} \;:\; A(pk, (f_{pk}(r), c)) \oplus c = b(r)\right] \\
&\quad = \Pr\left[(pk, sk) \leftarrow \mathsf{Gen}(1^n); r \leftarrow \{0,1\}^n; m \leftarrow \{0,1\} \;:\; A(pk, (f_{pk}(r), m \oplus b(r))) = m\right] \\
&\quad = \Pr\left[m \leftarrow \{0,1\}; (pk, sk) \leftarrow \mathsf{Gen}(1^n) \;:\; A(pk, \mathsf{Enc}_{pk}(m)) = m\right] \\
&\quad \geq \frac{1}{2} + \frac{1}{2p(n)}.
\end{aligned}$$

□

3.11 El-Gamal Public Key Encryption scheme

The El-Gamal public key encryption scheme is a popular and simple public key encryption scheme that is far more efficient than the one just presented. However, this efficiency requires us to make a new complexity assumption called the Decisional Diffie-Hellman Assumption (DDH).

Conjecture 83.1 (Decisional Diffie-Hellman assumption (DDH)). *For all p.p.t $\mathcal{A}$, the following two distributions are computationally indistinguishable*

$$\left\{p \leftarrow \tilde{\Pi}_n, y \leftarrow Gen_q, a, b \leftarrow \mathbb{Z}_q : p, y, y^a, y^b, y^{ab}\right\}_n \approx \left\{p \leftarrow \tilde{\Pi}_n, y \leftarrow Gen_q, a, b, z \leftarrow \mathbb{Z}_q : p, y, y^a, y^b, y^z\right\}_n$$

Here the term $\tilde{\Pi}_n$ refers to a special subset of primes called the *safe primes*:

$$\tilde{\Pi}_n = \{p \mid p \in \Pi_n \text{ and } p = 2q+1, q \in \Pi_{n-1}\}$$

The corresponding q is called a *Sophie Germain prime*. We use such a multiplicative group $G = \mathbb{Z}_p$ because it has a special structure that is convenient to work with. First, G has a subgroup G_q of order q, and since q is prime, G_q will be a cyclic group. Thus, it is easy to pick a generator of the group G_q (since every element is a generator). When $p = 2q + 1$, then the subgroup G_q consists of all of the squares modulo p. Thus, choosing a generator involves simply picking a random element $a \in G$ and computing a^2. Note that all of the math is still done in the "big" group, and therefore modulo p.

It is crucial for the DDH assumption that the group within which we work is a *prime-order* group. In a prime order group, all elements except the identity have the same order. On the other hand, in groups like G, there are elements of order $2, q$ and $2q$, and it is easy to distinguish between these cases. For example, if one is given a tuple $T = (p, y, g, h, f)$ and one notices that both g, h are of order q but f is of order $2q$, then one can immediately determine that tuple T is not a DDH-tuple.

Notice that the DDH assumption implies the discrete-log assumption (Assumption 43.1) since after solving the discrete log twice on the first two components, it is easy to distinguish whether the third component is y^{ab} or not.

We now construct a Public Key Encryption scheme based on the DDH assumption.

ALGORITHM 84.1: EL-GAMAL SECURE PUBLIC KEY ENCRYPTION

$\mathsf{Gen}(1^n)$: Pick a safe prime $p = 2q + 1$ of length n. Choose a random element $g \in \mathbb{Z}_p$ and compute $h \leftarrow g^2 \mod p$. Choose $a \leftarrow \mathbb{Z}_q$. Output the public key pk as $pk \leftarrow (p, h, h^a \bmod p)$ and sk as $sk \leftarrow (p, h, a))$.

$\mathsf{Enc}_{pk}(m)$: Choose $b \leftarrow \mathbb{Z}_q$. Output $(h^b, h^{ab} \cdot m \bmod p)$.

$\mathsf{Dec}_{sk}(c = (c_1, c_2)$: Output $c_2/c_1^a \bmod p$.

Roughly speaking, this scheme is secure assuming the DDH assumption since h^{ab} is indistinguishable from a random element and hence, by closure under efficient operations, $h^{ab} \cdot m$ is indistinguishable from a random element too. We leave the formal proof as an exercise to the reader.

Theorem 84.1. *Under the DDH Assumption, the El-Gamal encryption scheme is secure.*

3.12 A Note on Complexity Assumptions

Throughout this semester, we have built a hierarchy of constructions. At the bottom of this hierarchy are computationally difficult problems such as one-way functions, one-way permutations, and trapdoor permutations. Our efficient constructions of these objects were further based on specific number-theoretic assumptions, including factoring, RSA, discrete log, and decisional Diffie-Hellman.

Using these hard problems, we constructed several primitives: pseudorandom generators, pseudorandom functions, and private-key encryption schemes. Although our constructions were usually based on one-way permutations, it is possible to construct these using one-way functions. Further, one-way functions are a minimal assumption, because the existence of any of these primitives implies the existence of one-way functions.

Public-key encryption schemes are noticeably absent from the list of primitives above. Although we did construct two schemes, it is unknown how to base such a construction on one-way functions. Moreover, it is known to be impossible to create a black-box construction from one-way functions.

Chapter 4

Knowledge

In this chapter, we investigate what it means for a conversation to "convey" knowledge.

4.1 When Does a Message Convey Knowledge

Our investigation is based on a behavioristic notion of knowledge which views knowledge as an ability to complete a task. A conversation therefore conveys knowledge when the conversation allows the recipient to complete a "new" task that the recipient could not complete before. To quantify the knowledge inherent in a message m, it is therefore sufficient to quantify how much easier it becomes to compute some new function given m.

To illustrate the idea, lets consider the simplest case when Alice sends a single message to Bob, i.e., non-interactive conversations. As before, to describe such phenomena, we must consider a sequence of conversations of increasing size parameterized by n.

Imagine Alice always sends the same message 0^n to Bob. Alice's message is deterministic and it has a short description; Bob can easily produce the message 0^n himself. Thus, this message does not convey any knowledge to Bob.

Now suppose that f is a one-way function, and consider the case when Alice sends Bob the message consisting of "the preimage of the preimage ... (n times) of 0." Once again, the string that Alice sends is deterministic and has a short description. However, in this case, it is not clear that Bob can produce the message himself because producing the message might require a lot of computation (or a very large circuit). This leads us to a first approximate notion of knowledge. The amount of knowledge conveyed in a message can be quantified by considering the running time and size of a Turing machine that generates the message. With this notion, we can say that any message which can be generated by a constant-sized Turing machine that runs in polynomial-time in n conveys no knowledge since Bob can generate that message himself. These choices can be further refined, but are reasonable for our current purposes.

So far the messages that Alice sends are deterministic; our theory of knowledge should also handle the case when Alice uses randomness to select her message. In this case, the message that Alice sends is drawn from a probability distribution. To quantify the amount of knowledge conveyed by such a message, we again consider the complexity of a Turing machine that can produce the same distribution of messages as Alice. In fact, instead of requiring the machine to produce the identical distribution, we may be content with a machine that produces messages chosen from a computationally indistinguishable one. This leads to the following informal notion:

"Alice conveys zero knowledge to Bob if Bob can produce a distribution of messages that is computationally indistinguishable from the distribution of messages that Alice would send."

Shannon's theory of information is certainly closely related to our current discussion; briefly, the difference between information and knowledge in this context is the latter's focus on the *computational* aspects, i.e. running time and circuit size. Thus, messages that convey zero information may actually convey knowledge.

4.2 A Knowledge-Based Notion of Secure Encryption

As a first case study of our behavioristic notion of knowledge, we can recast the theory of secure encryption in terms of knowledge. (In fact, this was historically the first approach taken by Goldwasser and Micali.) A good notion for encryption is to argue that an encrypted message conveys zero knowledge to an eavesdropper. In other words, we say that an encryption scheme is secure if the cipertext does not allow the eavesdropper to compute any new (efficiently computable) function about the plaintext message with respect to what she could have computed without the ciphertext message.

The following definition of zero-knowledge encryption[1] captures this very intuition. This definition requires that there exists a simulator algorithm S which produces a string that is indistinguishable from a ciphertext of any message m.

[1] This is a variant of the well-known notion of semanical security.

Definition 88.1 (Zero-Knowledge Encryption). A private-key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is *zero-knowledge encryption scheme* if there exists a p.p.t. simulator algorithm S such that $\forall$ non uniform p.p.t D, $\exists$ a negligible function $\epsilon(n)$, such that $\forall m \in \{0,1\}^n$ it holds that D distinguishes the following distributions with probability at most $\epsilon(n)$

- $\{k \leftarrow \mathsf{Gen}(1^n) : \mathsf{Enc}_k(m)\}$
- $\{S(1^n)\}$

Remark 89.1. Note that we can strengthen the definition to require that the above distributions are *identical*; we call the resulting notion *perfect zero-knowledge*.

Remark 89.2. A similar definition can be used for public-key encryption; here we instead that D cannot distinguish the following two distributions

- $\{pk, sk \leftarrow \mathsf{Gen}(1^n) : pk, \mathsf{Enc}_{pk}(m)\}$
- $\{pk, sk \leftarrow \mathsf{Gen}(1^n) : pk, S(pk, 1^n)\}$

As we show below, for all "interesting" encryption schemes the notion of zero-knowledge encryption is equivalent to the indistinguishability-based notion of secure encryption. We show this for the case of private-key encryption, but it should be appreciated that the same equivalence (with essentially the same proof) holds also for the case of public-key encryption. (Additionally, the same proof show that perfect zero-knowledge encryption is equivalent to the notion of perfect secrecy.)

Theorem 89.1 (Equivalence of Secure and Zero-Knowledge Encryption). *Let* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be an encryption scheme such that* $\mathsf{Gen}, \mathsf{Enc}$ *are both p.p.t, and there exists a polynomial-time machine* M *such that for every* n, $M(n)$ *outputs a messages in* $\{0,1\}^n$. *Then* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is secure if and only if it is zero-knowledge.*

Proof. We prove each direction separately.

Security implies ZK. Intuitively, if was possible to extract some "knowledge" from the encrypted message, then there would be a way to distinguish between encryptions of two different messages. More formally, suppose that $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is secure. Consider the following simulator $S(1^n)$:

1. Pick a message $m \in \{0,1\}^n$ (recall that by our asumptions, this can be done in p.p.t.)
2. Pick $k \leftarrow \mathsf{Gen}(1^n)$, $c \leftarrow \mathsf{Enc}_k(m)$.

3. Output c.

It only remains to show that the output of S is indistinguishable from the encryption of any message. Assume for contradiction that there exist a n.u. p.p.t. distinguisher D and a polynomial $p(\cdot)$ such that for infinitely many n, there exist some message m'_n such that D distinguishes

- $\{k \leftarrow \mathsf{Gen}(1^n) : \mathsf{Enc}_k(m_n)\}$
- $\{S(1^n)\}$

with probability $p(n)$. Since $\{S(1^n)\} = \{k \leftarrow \mathsf{Gen}(1^n); m'_n \leftarrow M(1^n) : \mathsf{Enc}_k(m'_n)\}$, it follows that there exists messages m_n and m'_n such that their encryptions can be distinguished with inverse polynomial probability; this contradict the security of $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

ZK implies Security. Suppose for contradiction that $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is zero-knowledge, but there exists a n.u. p.p.t distringuisher D and a polynomial $p(n)$, such that for infinitely many n there exist messages m_n^1 and m_n^2 such that D distinguishes

- $H_n^1 = \{k \leftarrow \mathsf{Gen}(1^n) : \mathsf{Enc}_k(m_n^1)\}$
- $H_n^2 = \{k \leftarrow \mathsf{Gen}(1^n) : \mathsf{Enc}_k(m_n^2)\}$

with probability $p(n)$. Let S denote the zero-knowledge simulator for $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, and define the hybrid distribution H_3:

- $H_n^3 = \{S(1^n)\}$

By the hybrid lemma, D distinguishes between either H_n^1 and H_n^2 or between H_n^2 and H_n^3, with probability $\frac{1}{2p(n)}$ for infinitely many n; this is a contradiction.

4.3 Zero-Knowledge Interactions

So far, we have only worried about an honest Alice who wants to talk to an honest Bob, in the presence of a malicious Eve. We will now consider a situation in which neither Alice nor Bob trust each other.

Suppose Alice (the prover) would like to convince Bob (the verifier) that a particular string x is in a language L. Since Alice does not trust Bob, Alice wants to perform this proof in such a way that Bob learns nothing else except that $x \in L$. In particular, it should not be possible for Bob to later prove that

$x \in L$ to someone else. For instance, it might be useful in a cryptographic protocol for Alice to show Bob that a number N is the product of exactly two primes, but without revealing anything about the two primes.

It seems almost paradoxical to prove something in such a way that the theorem proven cannot be established subsequently. However, *zero-knowledge proofs* can be used to achieve exactly this.

Consider the following toy example involving the popular "Where's Waldo?" children's books. Each page is a large complicated illustration, and somewhere in it there is a small picture of Waldo, in his sweater and hat; the reader is invited to find him. Sometimes, you wonder if he is there at all.

The following protocol allows a prover to convince a verifier that Waldo is in the image without revealing any information about where he is in the image: Take a large sheet of newsprint, cut a Waldo-sized hole, and overlap it on the "Where's waldo" image, so that Waldo shows through the hole. This shows he is somewhere in the image, but there is no extra contextual information to show where.

A slightly more involved example follows. Suppose you want to prove that two pictures or other objects are distinct without revealing anything about the distinction. Have the verifier give the prover one of the two, selected at random. If the two really are distinct, then the prover can reliably say "this one is object 1", or "this is object 2". If they were identical, this would be impossible.

The key insight in both examples is that the verifier generates a puzzle related to the original theorem and asks the prover to solve it. Since the puzzle was generated by the verifier, the verifier already knows the answer—the only thing that the verifier does learn is that the puzzle *can* be solved by the prover, and therefore the theorem is true.

4.4 Interactive Protocols

To begin the study of zero-knowledge proofs, we must first formalize the notion of interaction. The first step is to consider *Interactive Turing Machine* instead of simply turing machine. Briefly, an interactive Turing machine (ITM) is a Turing machine with a read-only *input* tape, a read-only *auxiliary input* tape, a read-only *random* tape, a read/write *work-tape*, a read-only communication tape (for receiving messages) a write-only communication tape (for sending messages) and finally an *output* tape. The content of the input (respectively auxiliary input) tape of an ITM A is called *the input* (respectively *auxiliary input*) *of* A and the content of the output tape of A, upon halting, is called *the output of* A.

A protocol (A, B) is a pair of ITMs that share communication tapes so that the (write-only) send-tape of the first ITM is the (read-only) receive-tape of the second, and vice versa. The computation of such a pair consists of a sequence of rounds $1, 2, ...$ In each round only one ITM is active, and the other is idle. A round ends with the active machine either halting —in which case the protocol ends— or by it entering a special *idle* state. The string m written on the communication tape in a round is called the *message sent* by the active machine to the idle machine.

In this chapter, we consider protocols (A, B) where both ITMs A, B receive the *same* string as input (but not necessarily as auxiliary input); this input string will be denoted the *common input* of A and B. We make use of the following notation for protocol executions.

Executions, transcripts and views. Let M_A, M_B be vectors of strings $M_A = \{m_A^1, m_A^2, ...\}$, $M_B = \{m_B^1, m_B^2, ...\}$ and let $x, r_1, r_2, z_1, r_2 \in \{0,1\}^*$. We say that the pair $((x, z_1, r_1, M_A), (x, z_2, r_2, M_B))$ is an execution of the protocol (A, B) if, running ITM A on common input x, auxiliary input z_1 and random tape r_1 with ITM B on x, z_2 and r_2, results in m_A^i being the i'th message received by A and in m_B^i being the i'th message received by B. We also denote such an execution by $A_{r_1}(x, z_1) \leftrightarrow B_{r_2}(x, z_2)$.

In an execution $((x, z_1, r_1, M_A), (x, z_2, r_2, M_B)) = (V_A, V_B)$ of the protocol (A, B), we call V_A the *view of A* (in the execution), and V_B the *view of B*. We let $\mathsf{view}_A[A_{r_1}(x, z_1) \leftrightarrow B_{r_2}(x, z_2)]$ denote A's view in the execution $A_{r_1}(x, z_1) \leftrightarrow B_{r_2}(x, z_2)$ and $\mathsf{view}_B[A_{r_1}(x, z_1) \leftrightarrow B_{r_2}(x, z_2)]$ B's view in the same execution. (We occasionally find it convenient referring to an execution of a protocol (A, B) as a *joint view* of (A, B).)

In an execution $((x, z_1, r_1, M_A), (x, z_2, r_2, M_B))$, the pair (M_A, M_B) is called the transcript of the execution.

Outputs of executions and views. If e is an execution of a protocol (A, B) we denote by $\mathsf{out}_X(e)$ the output of X, where $X \in \{A, B\}$. Analogously, if v is the view of A, we denote by $\mathsf{out}(v)$ the output of A in v.

Random executions. We denote by $A(x, z_1) \leftrightarrow B(x, z_2)$, the probability distribution of the random variable obtained by selecting each bit of r_1 (respectively, each bit of r_2, and each bit of r_1 and r_2) randomly and independently, and then outputting $A_{r_1}(x, z_1) \leftrightarrow B_{r_2}(x, z_2)$. The corresponding probability distributions for view and out are analogously defined.

Time Complexity of ITMs. We say that an ITM A has time-complexity $t(n)$, if for every ITM B, every common input x, every auxiliary inputs z_a, z_b, it holds that $A(x, z_a)$ *always* halts within $t(|x|)$ steps in an interaction with $B(x, z_b)$, regardless of the content of A and B's random tapes). Note that time complexity is defined as an upperbound on the running time of A *independently* of the content of the messages it receives. In other words, the time complexity of A is the *worst-case* running time of A in *any* interaction.

4.5 Interactive Proofs

Given the above notation we start by considering *interactive proofs* in which a prover wishes to a verifier that a statement is true (without the additional property of zero-knowledge). Roughly speaking, we require the following two properties from an interactive proof system: it should be possible for a prover to convince a verifier of a true statment, but it should not be possible for a malicious prover to convince a verifier of a false statement.

Definition 93.1 (Interactive Proof). A pair of interactive machines (P, V) is an interactive proof system for a language L if V is a p.p.t. machine and the follwing properties hold.

1. (Completeness) For every $x \in L$, there exists a witness string $y \in \{0,1\}^*$ such that for every auxiliary string z:

$$\Pr\left[\mathsf{out}_V[P(x, y) \leftrightarrow V(x, z)] = 1\right] = 1$$

2. (Soundness) There exists some negligible function ϵ such that for all $x \notin L$ and for all prover algorithms P^*, and all auxiliary strings $z \in \{0,1\}^*$,

$$\Pr\left[\mathsf{out}_{V^*}[P^*(x) \leftrightarrow V(x, z)] = 0\right] > 1 - \epsilon(|x|)$$

Remark 93.1. Note that the prover in the definition of an interactive proof need not be efficient. (Looking forward, we shall later consider a definition which requires the prover to be efficient.)

Remark 93.2. Note that we do not provide any auxilary input to the "malicious" prover strategy P^*; this is without loss of generality as we consider *any* prover strategy; in particular, this prover strategy could have the auxilary input hard-coded.

Remark 93.3. Note that we can relax the definition and replace the $1 - \epsilon(|x|)$ with some constant (e.g., $\frac{1}{2}$); more generally we say that an interactive proof has soundness error $s(n)$ if it satisfies the above definitiob, but with $1 - \epsilon(|x|)$ replaced by $1 - s(n)$.

The class of languages having an interactive proofs is denoted **IP**. It trivially holds that **NP** $\subset IP$—the prover can simply provide the NP witness to the verifier, and the verifier checks if it is a valid witness. Perhaps surprisingly, there are languages that are not known to be in NP that also have interactive proofs: as shown by Shamir, every language in **PSPACE**—i.e., the set of languages that can be recognized in polynomial space—has an interactive proof; in fact, **IP** $=$ **PSPACE**. Below we provide an example of an interactive proof for a language that is not known to be in **NP**. More precisely, we show an interactive proof for the Graph Non-isomorphism Language.

An Interactive Proof Graph Non-isomorphism

A graph $G = (V, E)$ consists of a set of vertices V and a set of edges E which consists of pairs of verticies. Typically, we use n to denote the number of verticies in a graph, and m to denote the number of edges. Recall that two graphs $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ are isomorphic if there exists a permutation σ over the verticies of V_1 such that $V_2 = \{\sigma(v_1) \mid v_1 \in V_1\}$ and $E_2 = \{(\sigma(v_1), \sigma(v_2)) \mid (v_1, v_2) \in E_1\}$. In other words, permuting the verticies of G_1 and maintaining the permuted edge relations results in the graph G_2. We will often write $\sigma(G_1) = G_2$ to indicate that graphs G_1 and G_2 are isomorphic via the permutation σ. Similarly, two graphs are non-isomorphic if there exists no permutation σ for which $\sigma(G_1) = G_2$. (See Fig. 4.1 for examples.)

Notice that the language of isomorphic graphs is in NP since the permutation serves as a witness. Let L_{niso} be the language of pairs of graphs (G_0, G_1) that have the same number of verticies but are not isomorphic. This language $L_{niso} \in co - NP$ and is not known to be in NP. Consider the following protocol 94.1 which proves that two graphs are non-isomorphic.

94.1: PROTOCOL FOR GRAPH NON-ISOMORPHISM

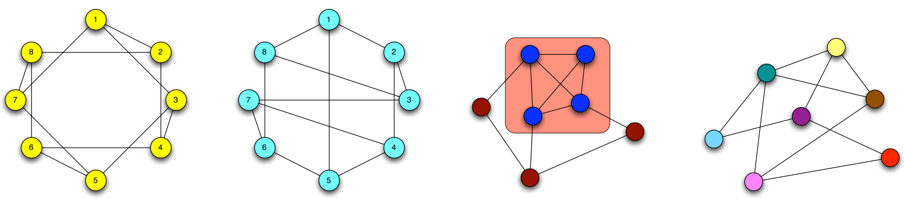


Figure 4.1: (a) Two graphs that are isomorphic, (b) Two graphs that are non-isomorphic. Notice that the highlighted 4-clique has no corresponding 4-clique in the extreme right graph.

Input:	$x = (G_0, G_1)$ where $\|G_i\| = n$
$V \xrightarrow{H} P$	The verifier, $V(x)$, chooses a random bit $b \in \{0,1\}$, chooses a random permutation $\sigma \in S_n$, computes $H \leftarrow \sigma(G_b)$, and finally sends H to the prover.
$V \xleftarrow{b'} P$	The prover computes a b' such that H and $G_{b'}$ are isomorphic and sends b' to the verifier.
$V(x, H, b, b')$	The verifier accepts and outputs 1 if $b' = b$ and 0 otherwise.
	Repeat the procedure $\|G_1\|$ times.

Proposition 95.1. *Protocol 94.1 is an interactive proof for L_{niso}.*

Proof. Completeness follows by inspection: If G_1 and G_2 are not isomorphic, then the Prover (who runs in exponential time in this protocol) will always succeed in finding b' such that $b' = b$. For soundness, the prover's chance of succeeding in one iteration of the basic protocol is $1/2$. This is because when G_1 and G_2 are isomorphic, then H is independent of the bit b. Since each iteration is independent of all prior iterations, the probability that a cheating prover succeeds is therefore upper-bounded by 2^{-n}. □

4.5.1 Interactive proofs with Efficient Provers

The Graph Non-isomorphism protocol required an exponential-time prover. (Indeed, a polynomial time prover would imply that $L_{niso} \in$ **NP**.) In cryptography, we will be more interested in protocols where the prover is efficient. To do so we are required to restrict our attention to languages in **NP**; the prover strategy is not required to be efficient given a **NP**-witness y to the statement x that it attempts to prove. See Appendix B for a formal definition of **NP** languages and witness relations.

Definition 96.0 (Interactive Proof with Efficient Provers). An interactive proof system (P, V) is said to have an *efficient prover* with respect to the witness relation R_L if P is p.p.t. and the completeness condition holds for every $y \in R_L(x)$.

Remark 96.1. Note that although we require that the honest prover strategy P is efficient, the soundness condition still requires that not even an all powerful prover strategy P^* can cheat the verifier V. A more relaxed notion—called an *interactive argument* considers only P^*'s that are n.u. p.p.t.

Although we have already shown that the Graph Isomorphism Language has an Interactive Proof, we now present a new protocol which will be useful to us later. Since we want an efficient prover, we provide the prover the witness for the theorem $x \in L_{iso}$, i.e., we provide the permutation to the prover.

An Interactive Proof for Graph Isomorphism

96.1: PROTOCOL FOR GRAPH ISOMORPHISM

Input:	$x = (G_0, G_1)$ where $\lvert G_i \rvert = n$
P's **witness**:	σ such that $\sigma(G_0) = G_1$
$V \xleftarrow{H} P$	The prover chooses a random permutation π, computes $H \leftarrow \pi(G_0)$ and sends H.
$V \xrightarrow{b} P$	The verifier picks a random bit b and sends it.
$V \xleftarrow{\gamma} P$	If $b = 0$, the prover sends π. Otherwise, the prover sends $\gamma = \pi \cdot \sigma^{-1}$.
V	The verifier outputs 1 if and only if $\gamma(G_b) = H$.
P, V	Repeat the procedure $\lvert G_1 \rvert$ times.

Proposition 96.1. *Protocol 96.1 is an interactive proof for L_{niso}.*

Proof. If the two graphs G_1, G_2 are isomorphic, then the verifier always accepts because $\pi(H) = G_1$ and $\sigma(\pi(H)) = \sigma(G_1) = G_2$. If the graphs are not isomorphic, then no malicious prover can convince V with probability greater than $\frac{1}{2}$: if G_1 and G_2 are not isomorphic, then H can be isomorphic to at most one of them. Thus, since b is selected at random after H is fixed, then with probability $\frac{1}{2}$ it will be the case that H and G_i are not isomorphic.

This protocol can be repeated many times (provided a fresh H is generated), to drive the probability of error as low as desired. □

As we shall see, the Graph-Iso protocol is in fact also zero-knowledge.

4.6 Zero-Knowledge Proofs

In addition to being an interactive proof, the protocol 96.1 also has the property that the verifier "does not learn anything" beyond the fact that G_0 and G_1 are isomorphic. In particular, the verifier does not learn anything about the permutation σ! As discussed in the introduction, by "did not learn anything," we mean that the verifier is not able to perform any extra tasks after seeing a proof that $(G_0, G_1) \in L_{iso}$. As with zero-knowledge encryption, we can formalize this idea by requiring there to be a simulator algorithm that produces "interactive transcripts" that are identical to the transcripts that the verifier encounters during the actual execution of the interactive proof protocol.

Definition 97.1 (Honest-verifier Zero-knowledge). Let (P, V) be an efficient interactive proof for the language $L \in$ **NP** with witness relation R_L. (P, V) is said to be *honest verifier zero-knowledge* if there exists a p.p.t. simulator S such that for every n.u. p.p.t distinguisher D, there exists a negligible function $\epsilon(\cdot)$ such that for every $x \in L, y \in R_L(x), z \in \{0,1\}^*$, D distinguishes the following distributions with probability at most $\epsilon(n)$.

- $\{\mathsf{view}_V[P(x,y) \leftrightarrow V(x,z)]\}$
- $\{S(x,z)\}$

Intuitively, the definition says whatever V "saw" in the interactive proof could have been generated by V himself by simply running the algorithm $S(x,z)$. The auxiliary input z to V denotes any a-priori information V has about x; as such the definition requires that V does not learn anything "new" (even considering this a-priori information).

This definition is, however, not entirely satisfactory. It ensures that when the verifier V follows the protocol, it gains no additional information. But what if the verifier is malicious and uses some other machine V^*. We would still like V to gain no additional information. To achieve this we modify the definition to require the existence of a simulator S for every, possibly malicious, efficient verifier strategy V^*. For technical reasons, we additionally slighty weaken the requirement on the simulator S and only require it to be an *expected p.p.t*—namely a machine whose *expected running-time* (where expectation is taken only over the internal randomness of the machine) is polynomial.[2]

Definition 98.1 (Zero-knowledge). Let (P, V) be an efficient interactive proof for the language $L \in \mathbf{NP}$ with witness relation R_L. (P, V) is said to be *zero-knowledge* if for every p.p.t adversary V^* there exists an expected p.p.t. simulator S such that for every n.u. p.p.t distinguisher D, there exists a negligible function $\epsilon(\cdot)$ such that for every $x \in L, y \in R_L(x), z \in \{0,1\}^*$, D distinguishes the following distributions with probability at most $\epsilon(n)$.

- $\{\mathsf{view}_{V^*}[P(x,y) \leftrightarrow V^*(x,z)]\}$
- $\{S(x,z)\}$

Remark 98.1. Note that here only consider p.p.t adversaries V^* (as opposed to *non-uniform* p.p.t adversaries). This only makes our definition stronger: V^* can anyway receive any non-uniform "advice" as its auxiliary input; in contrast, we can now require that the simulator S is only $p.p.t$ (but is of course also given the auxiliary input of V^*.) Thus, our definition says that even if V^* is non-uniform, the simulator only needs to get the same non-unform advice.

Remark 98.2. As for the case of zero-knowledge encryption, we can strengthen the definition to require the above to distributions to be identically distributed; in this case the interactive proof is called perfect zero-knowledge.

Remark 98.3. An alternate formalization more directly considers what V^* "can do", instead of what V^* "sees". That is, we require that whatever V^* can do after the interactions, V^* could have already done it before it. This is formalized by simply exchanging view_{V^*} to out_{V^*} in the above definition. We leave it as an exercise to the reader to verify that the definitions are equivalent.

Given the above formalism, we can now show that the Graph-isomorphism protocol is zero-knowledge.

Theorem 98.1. *Protocol 96.1 is a perfect zero-knowledge interactive proof for the Graph-isomorphism language (for some canonical witness relation).*

Proof. We have already demonstrated completeness and soundness in Proposition 96.1. We show how to construct an expected p.p.t. simulator for every p.p.t verifier V^*. $S(x,z)$ makes use of V^* and proceeds as described in Figure

[2] In essence, this relaxation will greatly facilitate the construction of zero-knowledge protocols

99.1. For simplicity, we here only provide a simulator for a single iteration of the Graph Isomorphism protocol; the same technique easily extends to the iterated version of the protocol as well. In fact, as we show in Section **??**, this holds for every zero-knowledge protocol: namely, the sequential repetition of any zero-knowledge protocol is still zero-knowledge.

99.1: SIMULATOR FOR GRAPH ISOMORPHISM

1. Randomly pick $b' \leftarrow \{0,1\}$, $\pi \leftarrow S_n$
2. Compute $H \leftarrow \pi(G_{b'})$.
3. Emulate the execution of $V^*(x,z)$ by feeding it H and truly random bits as its random coins; let b denote the response of V^*.
4. If $b = b'$ then output the view of V^*—i.e., the messages H, π, and the random coins it was feed. Otherwise, restart the emulation of V^* and repeat the procedure.

We need to show the following properties:

- the expected running time of S is polynomial,
- the output distribution of S is correctly distributed.

Towards this goal, we start with the following lemma.

Lemma 99.1. *In the execution of $S(x,z)$, H is identically distributed to $\pi(G_0)$, and* $\Pr[b' = b] = \frac{1}{2}$.

Proof. Since G_0 is an isomorphic copy of G_1, the distribution of $\pi(G_0)$ and $\pi(G_1)$ is the same for random π. Thus, the distribution of H is independent of b'. In particular, H has the same distribution as $\pi(G_0)$.

Furthermore, since V^* takes only H as input, its output, b, is also independent of b'. As b' is chosen at random from $\{0,1\}$, it follows that $\Pr[b' = b] = \frac{1}{2}$. □

From the lemma, we directly have that S has probability $\frac{1}{2}$ of succeeding in each trial. It follows that the expected number of trials before terminating is 2. Since each round takes polynomial time, S runs in expected polynomial time.

Also from the lemma, H has the same distribuion as $\pi(G_0)$. Thus, if we were always able to output the corresponding π, then the output distribution of S would be the same as in the actual protocol. However, we only output H if $b' = b$. Fortunetly, since H is independent from b', this does not change the output distribution. □

4.7 Zero-knowledge proofs for NP

We now show that every language in **NP** has a zero-knowledge proof system assuming the existence of one-way permutation. (In fact, using a a much more involved proof, it can be shown that general one-way function suffice.)

Theorem 100.1. *Assume the existence of one-way permutations. Then every languages in* **NP** *has a zero-knowledge proof.*

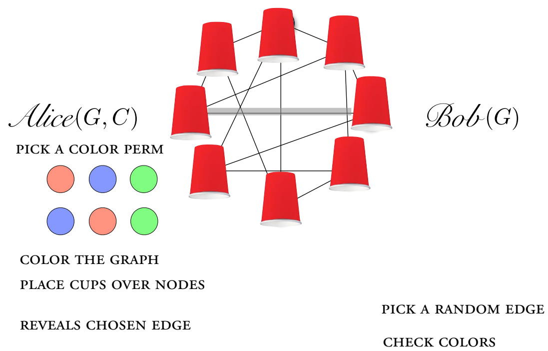
Proof. Our proof proceeds in two steps:

Step 1: Show a ZK proof (P', V') (with efficient provers) for an **NP**-complete language; the particular language we will consider is *Graph 3 Coloring*—namely the language of all graphs whose vertices can be colored using only three colors $1, 2, 3$ such that no to connected vertices have the same color.

Step 2: To get a zero-knowledge proof (P, V) for any **NP** language, proceed as follows: Given a language L, instance x and witness y, both P and V reduce x into an instance of a Graph 3-coloring x'; this can be done using Cook's reduction (the reduction is deterministic which means that both P and V will reach the same instance x). Additionally, Cook's reduction can be applied to the witness y yielding a witness y' for the instance x'. The parties then (P, V) on common input x', and the prover additionally uses y' as its auxiliary input.

It is easy to verify that the above protocol is a zero-knowledge proof (assuming that (P', V') is a zero-knowledge proof for Graph 3-coloring.) Thus it only remains to show a zero-knowledge proof for Graph 3-coloring.

To give some intuition, we start by proving a "physical" variant of the protocol. Given a graph $G = (V, E)$, where V is the vertice set, and E is the edge set, and a coloring C of the vertices V, the prover picks a random permutation π over the colors $\{1, 2, 3\}$ and physically colors the graph G with the permutated colors. It then covers each vertices with individual cups. The verifier is next asked to pick a random edge, and the prover is supposed to remove the two cups corresponding to the vertices of the edge, to show that the two vertices have different colors. If they don't the prover has been caught cheating, otherwise the interaction is repeated (each time letting the prover pick a new random permutation π.) As we shall see, if the procedure is repeated $O(n|E|)$, where $|E|$ is the number of edges, then the soundness error will be 2^{-n}. Additionally, in each round of the intereaction, the verifier



only learns something he knew before—two random (but different) colors! See Figure 4.7 for a pictoral description of the protocol.

To be able to digitally implement the above protocol, we need to have a way to implement the "cups". Intuitively, we require two properties from the cups: a) the verifier should not be able to see what is under the cup—i.e., they cups should be *hiding*, b) the prover should not be able to change what is under a cup—i.e, they should be *binding*. The cryptographic notion that allows to acheive this properties is a commitment schemes.

4.7.1 Commitment Schemes

Commitment schemes are usually referred to as the digital equivalent to a "physical" locked boxes. They consist of two phases:

Commit phase : Sender puts a value v in a locked box.

Reveal phase : Sender unlocks the box and reveals v.

We require that before the reveal phase the value v should remain hidden: this property is called *hiding*. Additionally, during the reveal phase, there should only exists a signle value that the commitment can be revealed to: this is called *binding*.

We provide a formalization of single-messages commitments where both the commit and the reveal phases only consist of a single message sent from the committer to the receiver.

Definition 101.1 (Commitment). A polynomial-time machine Com is called a commitment scheme it there exists some polynomial $\ell(\cdot)$ such that the following two properties hold:

1. (Binding): For all $n \in N$ and all $v_0, v_1 \in \{0,1\}^n$, $r_0, r_1 \in \{0,1\}^{l(n)}$ it holds that $\mathsf{Com}(v_0, r_0) \neq \mathsf{Com}(v_1, r_1)$.
2. (Hiding): For every n.u. p.p.t distinguisher D, there exists a negligible function ϵ such that for every $n \in N$, $v_0, v_1 \in \{0,1\}^n$, D distinguishes the following distributions with probability at most $\epsilon(n)$.
   - $\{r \leftarrow \{0,1\}^{l(n)} : \mathsf{Com}(v_0, r)\}$
   - $\{r \leftarrow \{0,1\}^{l(n)} : \mathsf{Com}(v_1, r)\}$

Remark 102.1. Just as the definition of multi-message secure encryption, we can define a notion of "multi-value security" for commitments. It directly follows by a simple hybrid argument that any commitment scheme is multi-value secure.

Theorem 102.1. *Assume the existence of one-way permutations. Then there exists a commitment scheme.*

Proof. We give a construction of a single-bit commitment scheme. A full-fledge commitment scheme to a value $v \in \{0,1\}^n$ can be obtained by individually committing to each bit of v; the security of the full-fledged construction follows as a simple application of the hybrid lemma (show this!).

Let f be a one-way permutation with a hard-core predicate h. Let $\mathsf{Com}(b, r) = f(r), b \oplus h(r)$. It directly follows from the construction that Com is binding. Hiding follows using identically the same proof as in the proof of Theorem 82.1. □

4.7.2 A Zero-knowledge Proof for Graph 3-Coloring

102.1: ZERO-KNOWLEDGE FOR GRAPH 3-COLORING

Common input:	$G = (V, E)$ where $\|V\| = n,\ \|E\| = m$
Prover input:	Witness $y = c_0, c_1, \ldots, c_m$
$P \to V$	Let π be a random permutation over $\{1,2,3\}$. For each $i \in [1, n]$, the prover sends a commitment to the color $\pi(c_i) = c'_i$.
$V \to P$	The verifier sends a randomly chosen edge $(i, j) \in E$
$P \to V$	The prover opens commitments c'_i and c'_j
V	V accepts the proof if and only if $c'_i \neq c'_j$
P, V	Repeat the procedure $n\|E\|$ times.

Proposition 102.1. *Protocol 102.1 is a zero-knowledge protocol for the language of 3-colorable graphs.*

Proof. The completeness follows by inspection. If G is not 3 colorable, then for each coloring $c_1, \ldots, c_m$, there exists at least one edge which has the same colors on both endpoints. Thus, soundness follows by the binding property of the commitment scheme: In each iteration, a cheating prover is caught with probability $1/|E|$. Since the protocol is repeated $|E|^2$ times, the probability of successfully cheating in all rounds is

$$\left(1 - \frac{1}{|E|}\right)^{n|E|} \approx e^{-n}$$

For the zero-knowledge property, the prover only "reveals" 2 random colors in each iteration. The hiding property of the commitment scheme intuitively guarantees that "everything else" is hidden.

To prove this formally requires a lot more work. We construct the simulator in a similar fashion to the graph isomorphism simulator. Again, for simplicity, we here only provide a simulator for a single iteration of the Graph 3-Coloring protocol. As previously mentioned, this is without loss of generality (see Section **??**).

103.1: SIMULATOR FOR GRAPH 3-COLORING

1. Pick a random edge $(i', j') \in E$ and pick random colors $c'_i, c'_j \in \{1, 2, 3\}, c'_i \neq c'_j$. Let $c'_k = 1$ for all other $k \in [m]/\{i', j'\}$
2. Just as the honest prover, commit to c'_i for all i and feed the commitments to $V^*(x, z)$ (while also providing it truly random bits as its random coins).
3. Let (i, j) denote the answer from V^*.
4. If $(i, j) = (i', j')$ reveal the two colors, and output the view of V^*. Otherwise, restart the process from the first step, but at most $n|E|$ times.
5. If, after $n|E|$ repetitions the simulation has not been sucessful, output `fail`.

By construction it directly follows that S is a p.p.t. We proceed to show that the simulator's output distribution is correctly distributed.

Proposition 103.1. *For every n.u. p.p.t distinguisher D, there exists a negligible function $\epsilon(\cdot)$ such that for every $x \in L, y \in R_L(x), z \in \{0,1\}^*$, D distinguishes the following distributions with probability at most $\epsilon(n)$.*

- $\{\textsf{view}_{V^*}[P(x, y) \leftrightarrow V^*(x, z)]\}$

- $\{S(x, z)\}$

Assume for contradiction that there exists some n.u. distinguisher D and a polynomial $p(\cdot)$, such that for infinitely many $x \in L, y \in R_L(x), z \in \{0,1\}^*$, D distinguishes

- $\{\mathsf{view}_{V^*}[P(x,y) \leftrightarrow V^*(x,z)]\}$
- $\{S(x, z)\}$

with probability $p(|x|)$. First consider the following hybrid simulator S' that, just as the prover, gets the real witness $y = c_1, \ldots, c_n$: S' proceeds exactly as S, except that instead of picking the colors c'_i and c'_j at random, it picks π at random and lets $c'_i = \pi(c_i)$ and $c'_j = \pi(c_j)$ (just as the prover). It directly follows that $\{S(x, z)\}$ and $\{S'(x, z, y)\}$ are identically distributed.

Next, consider the following hybrid simulator S'': S'' proceeds just as S, but just as the real prover commits to a random permutation of the coloring given in the witness y; except for that it does everything just like S—i.e., it picks i', j' at random and restarts if $(i, j) \neq (i', j')$. If we assume that S' never outputs `fail`, then clearly the following distributions are identical.

- $\{\mathsf{view}_{V^*}[P(x,y) \leftrightarrow V^*(x,z)]\}$
- $\{S''(x, z, y)\}$

However, as i, j and i', j' are independently chosen, S' fails with probability

$$\left(1 - \frac{1}{|E|}\right)^{n|E|} \approx e^{-n}$$

It follows that

- $\{\mathsf{view}_{V^*}[P(x,y) \leftrightarrow V^*(x,z)]\}$
- $\{S''(x, z, y)\}$

can be distinguished with probability at most $O(e^{-n}) < \frac{1}{2p(n)}$. By the hybrid lemma, D thus distinguishes $\{S'(x, z, y)\}$ and $\{S''(x, z)\}$ with probability $\frac{1}{2p(n)}$. Next consider a sequence $S_0, S_1, \ldots, S_{2n|E|}$ of hybrid simulators where S_k proceeds just as S' in the first k iterations, and like S'' in the remaining ones. Note that $\{S_0(x, z, y)\}$ is identically distributed to $\{S''(x, z, y)\}$ and $\{S_{2n|E|}(x, z, y)\}$ is identically distributed to $\{S'(x, z, y)\}$. By the hybrid lemma, there exist some k such that D distinguishes between

- $\{S_k(x,z,y)\}$
- $\{S_{k+1}(x,z,y)\}$

with probability $\frac{1}{2n|E|p(n)}$. (Recall that the only difference between S_k and S_{k+1} is that in the $k+1$th iteration, S_k commits to 1's, whereas S_{k+1} commits to the real witness.) Consider, next, another sequence of $6|E|$ hybrid simulators $\tilde{S}_0, \ldots, \tilde{S}_{6|E|}$ where $\tilde{S}_e$ proceeds just as S_k if in the $k+1$th iteration the index, of the edge (i', j') and permutation π, is smaller than e; otherwise, it proceeds just as S_{k+1}. Again, note that $\{\tilde{S}_0(x,z,y)\}$ is identically distributed to $\{S_{k+1}(x,z,y)\}$ and $\{\tilde{S}_{6|E|}(x,z,y)\}$ is identically distributed to $\{S_k(x,z,y)\}$. By the hybrid lemma there exists some $e = (\tilde{i}, \tilde{j}, \tilde{\pi})$ such that D distinguishes

- $\{\tilde{S}_e(x,z,y)\}$
- $\{\tilde{S}_{e+1}(x,z,y)\}$

with probability $\frac{1}{12n|E|^2p(n)}$. Note that the only difference between $\{\tilde{S}_e(x,z,y)\}$ and $\{\tilde{S}_e(x,z,y)\}$ is that in $\{\tilde{S}_{e+1}(x,z,y)\}$, if in the kth iteration $(i,j,\pi) = (\tilde{i}, \tilde{j}, \tilde{\pi})$, then V^* is feed commitments to $\pi(c_k)$ for all $k \notin \{i,j\}$, whereas in $\{\tilde{S}_{e+1}(x,z,y)\}$, it is feed commitments to 1. Since $\tilde{S}_e$ is computable in n.u. p.p.t, by closure under efficient operations, this contradicts the (multi-value) computational hiding property of the commitments. □

4.8 Proof of knowledge

4.9 Applications of Zero-knowledge

One of the most basic applications of zero-knowledge protocols are for secure identification to a server. A typical approach to identification is for a server and a user to share a secret password; the user sends the password to the server to identify herself. This approach has one major drawback: an adversary who intercepts this message can impersonate the user by simply "replaying" the password to another login session.

It would be much better if the user could prove identity in such a way that a *passive adversary* cannot subsequently impersonate the user. A slightly better approach might be to use a signature. Consider the following protocol in which the User and Server share a signature verification key V to which the User knows the secret signing key S.

1. The User sends the Server the "login name."

2. The server sends the User the string σ="Server name, r" where r is a randomly chosen value.
3. The user responds by signing the message σ using the signing key S.

Proving Your Identity with Leaving a Trace

Here, "User" is trying to prove to "Server" that she holds the private key S corresponding to public key V; r is a nonce chosen at random from $\{0,1\}^n$. We are implicitly assuming that the signature scheme resists chosen-plaintext attacks. Constraining the text to be signed in some way (requiring it to start with "server") helps.

This protocol has a subtle consequence. The server can prove that the user with public key V logged in, since the server has, and can keep, the signed message $\sigma = \{$"Server name", $r\}$. This property is sometimes undesirable. Imagine that the user is accessing a politically subversive website. With physical keys, there is no way to show afterwards whether the key was used, and it might be nice to have a digital version of that property.

A zero-knowledge protocol can solve this problem. Imagine that instead of sending a signature of the message, the User simply proves in zero-knowledge that it knows the key S corresponding to V. Certainly such a statement is in an NP language, and therefore the prior protocols can work. Moreover, the server now has no reliable way of proving to another party that the user logged in. In particular, no one would believe a server who claimed as such because the server could have easily created the "proof transcript" by itself by running the Simulator. In this way, zero-knowledge protocols provide a tangibly new property than simple "challenge-response" identity protocols.

Chapter 5

Authentication

5.1 Message Authentication

Suppose Bob receives a message addressed from Alice. How does Bob ensure that the message *received* is the same as the message *sent* by Alice? For example, if the message was actually sent by Alice, how does Bob ensure that the message was not tampered with by any malicious intermediary?

In day-to-day life, we use signatures or other physical methods to solve the forementioned problem. Historically, governments have used elaborate and hard-to-replicate seals, watermarks, special papers, holograms, etc. to address this problem. In particular, these techniques help ensure that only, say, the physical currency issued by the government is accepted as money. All of these techniques rely on the physical difficulty of "forging" an official "signature."

In this chapter, we will discuss digital methods which make it difficult to "forge" a "signature." Just as with encryption, there are two different approaches to the problem based on whether private keys are allowed: *message authentication codes* and *digital signatures*. Message Authentication Codes (MACs) are used in the private key setting. Only people who know the secret key can check if a message is valid. Digital Signatures extend this idea to the public key setting. Anyone who knows the public key of Alice can verify a signature issued by Alice, and only those who know the secret key can issue signatures.

5.2 Message Authentication Codes

Definition 107.1 (MAC). $(\mathsf{Gen}, \mathsf{Tag}, \mathsf{Ver})$ is a MAC over the message space $\{\mathcal{M}\}_n$ if the following hold:

- Gen is a p.p.t. algorithm that returns a key $k \leftarrow Gen(1^n)$.
- Tag is a p.p.t. algorithm that on input key k and message m outputs a *tag* $\sigma \leftarrow Tag_k(m)$.
- Ver is a deterministic polynomial-time algorithm that on input k, m and σ outputs *"accept"* or *"reject"*.
- For all $n \in N$, for all $m \in \mathcal{M}_n$,

$$\Pr[k \leftarrow \mathsf{Gen}(1^n) : \mathsf{Ver}_k(m, \mathsf{Tag}_k(m)) = \text{"accept"}] = 1$$

The above definition requires that verification algorithms always correctly "accepts" a valid signature.

The goal of an adversary is to forge a MAC. In this case, the adversary is said to forge a MAC if it is able to construct a tag σ' such that it is a valid signature for some message. We could consider many different adversaries with varying powers depending on whether the adversary has access to signed messages; whether the adversary has access to a signing oracle; and whether the adversary can pick the message to be forged. The strongest adversary is the one who has oracle access to Tag and is allowed to forge any chosen message.

Definition 108.1 (Security of a MAC). A message authentication code $(\mathsf{Gen}, \mathsf{Tag}, \mathsf{Ver})$ is *secure* if for all non-uniform p.p.t. adversaries A, there exists a negligible function $\epsilon(n)$ such that for all n,

$$\Pr[k \leftarrow \mathsf{Gen}(1^n); m, \sigma \leftarrow A^{\mathsf{Tag}_k(\cdot)}(1^n) : \\ A \text{ did not query } m \wedge \mathsf{Ver}_k(m, \sigma) = \text{"accept"}] \leq \epsilon(n)$$

We now show a construction of a MAC using pseudorandom functions.

108.1: MAC SCHEME

Let $F = \{f_s\}$ be a family of pseudorandom functions such that $f_s : \{0,1\}^{|s|} \to \{0,1\}^{|s|}$.

$\mathsf{Gen}(1^n)$**:** $k \leftarrow \{0,1\}^n$

$\mathsf{Tag}_k(m)$**:** Output $f_k(m)$

$\mathsf{Ver}_k(m, \sigma)$**:** Ouptut "accept" if and only if $f_k(m) = \sigma$.

Theorem 109.0. *If there exists a pseudorandom function, then the above scheme is a Message Authentication Code over the message space* $\{0,1\}_n$.

Proof. (Sketch) Consider the above scheme when a random function RF is used instead of the pseudorandom function F. In this case, A succeeds with a probability at most 2^{-n}, since A only wins if A is able to guess the n bit random string which is the output of $RF_k(m)$ for some new message m. From the security property of a pseudorandom function, there is no non uniform p.p.t. distinguisher which can distinguish the output of F and RF with a non negligible probability. Hence, we conclude that $(\mathsf{Gen}, \mathsf{Tag}, \mathsf{Ver})$ is secure. □

5.3 Digital Signature Schemes

With message authentication codes, both the signer and verifier need to share a secret key. In contrast, digital signatures mirror real-life signatures in that anyone who knows Alice (but not necessarily her secrets) can verify a signature generated by Alice. Moreover, digital signatures possess the property of *non-repudiability*, i.e., if Alice signs a message and sends it to Bob, then Bob can prove to a third party (who also knows Alice) the validity of the signature. Hence, digital signatures can be used as certificates in a public key infrastructure.

Definition 109.1 (Digital Signatures). $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ is a *digital signature scheme* over the message space $\{M_n\}_n$ if

- $\mathsf{Gen}(1^n)$ is a p.p.t. which on input n outputs a public key pk and a secret key sk: $pk, sk \leftarrow Gen(1^n)$.

- Sign is a p.p.t. which on input a secret key sk and message m outputs a signature σ: $\sigma \leftarrow Sign_s k(m)$.

- Ver is a deterministic p.t. algorithm which on input a public key pk, a message m and a signature σ returns either *"accept"* or *"reject"*.

- For all $n \in N$, for all $m \in \mathcal{M}_n$,

$$\Pr[pk, sk \leftarrow \mathsf{Gen}(1^n) : \mathsf{Ver}_{pk}(m, \mathsf{Sign}_{sk}(m)) = \text{"accept"}] = 1$$

The security of a digital signature can be defined in terms very similar to the security of a MAC. The adversary can make a polynomial number of queries to a signing oracle. It is not considerd a forgery if the adversary A produces a signature on a message m on which it has queried the signing oracle. Note that by definition of a public key infrastructure, the adversary has free oracle access to the verification algorithm $\mathsf{Ver}_{\mathrm{pk}}$.

Definition 110.0 (Security of Digital Signatures). $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ is secure if for all non-uniform p.p.t. adversaries A, there exists a negligible function $\epsilon(n)$ such that $\forall n \in N$,

$$\Pr[pk, sk \leftarrow \mathsf{Gen}(1^n); m, \sigma \leftarrow A^{\mathsf{Sign}_{sk}(\cdot)}(1^n) : \\ A \text{ did not query } m \ \wedge \mathsf{Ver}_{pk}(m, \sigma) = \text{ “accept”}] \leq \epsilon(n)$$

In contrast, a digital signature scheme is said to be *one-time secure* if Definition 109.2 is satisfied under the constraint that the adversary A is only allowed to query the signing oracle *once*. In general, however, we need a digital signature scheme to be many-message secure. The construction of the one-time secure scheme, however, gives insight into the more general construction.

5.4 A One-Time Digital Signature Scheme for $\{0,1\}^n$

To produce a many-message secure digital signature scheme, we first describe a digital signature scheme and prove that it is one-time secure for n-bit messages. We then extend the scheme to handle arbitrarily long messages. Finally, we take that scheme and show how to make it many-message secure.

Our one-time secure digital signature scheme is a triple $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$. Gen produces a secret key consisting of $2n$ random elements and a public key consisting of the image of the same $2n$ elements under a one-way function f.

110.1: ONE-TIME DIGITAL SIGNATURE SCHEME

$\mathsf{Gen}(1^n)$: For $i = 1$ to n, and $b = 0, 1$, pick $x_b^i \leftarrow U_n$. Output the keys:

$$\text{sk} = \begin{pmatrix} x_0^1 & x_0^2 & \cdots & x_0^n \\ x_1^1 & x_1^2 & & x_1^n \end{pmatrix}$$
$$\text{pk} = \begin{pmatrix} f(x_0^1) & f(x_0^2) & \cdots & f(x_0^n) \\ f(x_1^1) & f(x_1^2) & & f(x_1^n) \end{pmatrix}$$

$\mathsf{Sign}_{sk}(m)$: For $i = 1$ to n, $\sigma_i \leftarrow x_{m_i}^i$. Output $\sigma = (\sigma_1, \ldots, \sigma_n)$.

$\mathsf{Ver}_{pk}(\sigma, m)$**:** Output accept if and only if $f(\sigma_i) = f(x^i_{m_i})$ for all $i \in [1, n]$.

For example, to sign the message $m = 010$, $\mathsf{Sign}_{\mathrm{sk}}(m)$ returns x^1_0, x^2_1, x^3_0. From these definitions, it is immediately clear that $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ is a digital signature scheme. However, this signature scheme is not many-message secure because after two signature queries (on say, the message $0 \ldots 0$ and $1 \ldots 1$), it is possible to forge a signature on any message.

Nonetheless, the scheme is one-time secure. The intuition behind the proof is as follows. If after one signature query on message m, if A produces a pair m', σ' that satisfies $\mathsf{Ver}_{\mathrm{sk}}(m', \sigma') = \mathsf{accept}$ and $m \neq m'$, then A must be able to invert f on a new point. Thus A has broken the one-way function f.

Theorem 111.1. *If f is a one-way function, then* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ *is one-time secure.*

Proof. By contradiction. Suppose f is a one-way function, and suppose we are given an adversary A that succeeds with probability $\epsilon(n)$ in breaking the one-time signature scheme. We construct a new adversary B that inverts f with probability $\frac{\epsilon(n)}{\mathrm{poly(n)}}$.

B is required to invert a one-way function f, so it is given a string y and access to f, and needs to find $f^{-1}(y)$. The intuition behind the construction of B is that A on a given instance of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ will produce at least one value in its output that is the inverse of $f(x^i_j)$ for some x^i_j not known to A. Thus, if B creates an instance of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ and replaces one of the $f(x^i_j)$ with y, then there is some non-negligible probability that A will succeed in inverting it, thereby inverting the one-way function.

Let m and m' be the two messages chosen by A (m is A's request to the signing oracle, and m' is in A's output). If m and m' were always going to differ in a given position, then it would be easy to decide where to put y. Instead, B generates an instance of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ using f and replaces one of the values in pk with y. With some probability, A will choose a pair m, m' that differ in the position B chose for y. B proceeds as follows:

- Pick a random $i \in \{1, \ldots, n\}$ and $c \in \{0, 1\}$
- Generate pk, sk using f and replace $f(x^i_c)$ with y
- Internally run $m', \sigma' \leftarrow A(\mathrm{pk}, 1^n)$
  - A may make a query m to the signing oracle. B answers this query if m_i is $1 - c$, and otherwise aborts (since B does not know the inverse of y)

- if $m'_i = c$, output σ'_i, and otherwise output $\perp$

To find the probability that B is successful, first consider the probability that B aborts while running A internally; this can only occur if A's query m contains c in the ith bit, so the probability is $\frac{1}{2}$. This probability follows because B's choice of c is independent of A's choice of m (A cannot determine where B put y, since all the elements of pk, including y, are the result of applications of f to a random value). The probability that B chose a bit that differs between m and m' is greater than $\frac{1}{n}$ (since there must be at least one such bit), and A succeeds with probability ϵ.

Thus B returns $f^{-1}(y) = \sigma'_i$ and succeeds with probability greater than $\frac{\epsilon}{2n}$. The security of f implies that $\epsilon(n)$ must be negligible, which implies that $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ is one-time secure. □

Now, we would like to sign longer messages with the same length key. To do so, we will need a new tool: collision-resistant hash functions.

5.5 Collision-Resistant Hash Functions

Intuitively, a hash function is a function $h(x) = y$ such that the representation of y is smaller than the representation of x, so h compresses x. The output of hash function h on a value x is often called the *hash* of x. Hash functions have a number of useful applications in data structures. For example, the Java programming language provides a built-in method that maps any string to a number in $[0, 2^{32})$. The following simple program computes the hash for a given string.

```
public class Hash {

    public void main(String args[]) {
        System.out.println( args[0].hashCode() );
    }
}
```

By inspeciting the Java library, one can see that when run on a string s, the `hashCode` function computes and returns the value

$$T = \sum_i s[i] \cdot 31^{n-i}$$

where n is the length of the string and $s[i]$ is the ith character of s. This function has a number of positive qualities: it is easy to compute, and it is

n-wise independent on strings of length n. Thus, when used to store strings in a hash table, it performs very well.

For a hash function to be cryptographically useful, however, we require that it be hard to find two elements x and x' such that $h(x) = h(x')$. Such a pair is called a *collision*, and hash functions for which it is hard to find collisions are said to satisfy *collision resistance* or are said to be *collision-resistant*. Before we formalize collision resistance, we should note why it is useful: rather than signing a message m, we will sign the hash of m. Then even if an adversary A can find another signature σ on some bit string y, A will not be able to find any x such that $h(x) = y$, so A will not be able to find a message that has signature σ. Further, given the signature of some message m, A will not be able to find an m' that has $h(m) = h(m')$ (if A could find such an m', then m and m' would have the same signature).

With this in mind, it is easy to see that the Java hash function does not work well as a cryptographic hash function. For example, it is very easy to change the last two digits of a string to make a collision. (This is because the contribution of the last two symbols to the output is $31 * s[n-1] + s[n]$. One can easily find two pairs of symbols which contribute the same value here, and therefore when pre-pended with the same prefix, result in the same hash.)

5.5.1 A Family of Collision-Resistant Hash Functions

It is not possible to guarantee collision resistance against a non-uniform adversary for a single hash function h: since h compresses its input, there certainly exist two inputs x and x' that comprise a collision. Thus, a non-uniform adversary can have x and x' hard-wired into their circuits. To get around this issue, we must introduce a family of collision-resistant hash functions.

Definition 113.1. A set of functions $H = \{h_i : D_i \to R_i\}_{i \in I}$ is a *family of collision-resistant hash functions* (CRH) if:

- (ease of sampling) Gen runs in p.p.t: $\mathsf{Gen}(1^n) \in I$
- (compression) $|R_i| < |D_i|$
- (ease of evaluation) Given $x, i \in I$, the computation of $h_i(x)$ can be done in p.p.t.
- (collision resistance) $\forall$ non-uniform p.p.t A . $\exists$ negligible ϵ such that $\forall\, n \in \mathbb{N}$.

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

Note that compression is a relatively weak property and does not even guarantee that the output is compressed by one bit. In practice, we often require that $|h(x)| < \frac{|x|}{2}$. Also note that if h is collision-resistant, then h is one-way.[1]

5.5.2 Attacks on CRHFs

Collision-resistance is a stronger property than one-wayness, so finding an attack on a collision-resistant hash functions is easier than finding an attack on a one-way function. We now consider some possible attacks.

Enumeration. If $|D_i| = 2^d$, $|R_i| = 2^n$, and x, x' are chosen at random, what is the probability of a collision between $h(x)$ and $h(x')$?

In order to analyze this situation, we must count the number of ways that a collision can occur. Let p_y be the probability that h maps a element from the domain into $y \in R_i$. The probability of a collision at y is therefore p_y^2. Since a collision can occur at either y_1 or y_2, etc., the probability of a collision can be written as

$$\Pr[collision] = \sum_{y \in R_i} p_y^2$$

Since $\sum_{y \in R_i} p_y = 1$, by the Cauchy-Scwartz Theorem **??**, we have that

$$\sum_{y \in R_i} p_y^2 > \frac{1}{|R_i|}$$

The probability that x and x' are not identical is $\frac{1}{|D_i|}$. Combining these two shows that the total probability of a collision is greater than $\frac{1}{2^n} - \frac{1}{2^d}$. In other words, enumeration requires searching most of the range to find a collision.

Birthday attack. Instead enumerating pairs of values, consider a set of random values $x_1, \ldots, x_t$. Evaluate h on each x_i and look for a collision between any pair x_i and $x_{i'}$. By the linearity of expectations, the expected number of

[1] The question of how to construct a CRH from a one-way permutation, however, is still open. There is a weaker kind of hash function: the universal one-way hash function (UOWF). A UOWF satisfies the property that it is hard to find a collision for a particular message; a UOWF can be constructed from a one-way permutation.

collisions is the number of pairs multiplied by the probability that a random pair collides. This probability is

$$\binom{t}{2}\left(\frac{1}{|R_i|}\right) \approx \frac{t^2}{|R_i|}$$

so $O(\sqrt{|R_i|}) = O(2^{n/2})$ samples are needed to find a collision with good probability. In other words, the birthday attack only requires the attacker to do computation on the order of the square root of the size of the output space.[2] This attack is much more efficient than the best known attacks on one-way functions, since those attacks require enumeration.

Now, we would like to show that, given some standard cryptographic assumptions, we can produce a CRH that compresses by one bit. Given such a CRH, we can then construct a CRH that compresses more.[3]

115.1: COLLISION RESISTANT HASH FUNCTION

$\mathsf{Gen}(1^n)$: Outputs a triple (g, p, y) such that p is an n-bit prime, g is a generator for $\mathbb{Z}_p^*$, and y is a random element in $\mathbb{Z}_p^*$.

$h_{p,g,y}(x, b)$: Given any n-bit string x and bit b,

$$h_{p,g,y}(x, b) = y^b g^x \bmod p$$

Theorem 115.1. *Under the Discrete Logarithm assumption, construction 115.1 is a collision-resistant hash function that compresses by 1 bit.*

Proof. Notice that both Gen and h are efficiently computable, and h compresses by one bit (since the input is in $\mathbb{Z}_p^* \times \{0, 1\}$ and the output is in $\mathbb{Z}_p^*$). We need to prove that if we could find a collision, then we could also find the discrete logarithm of y.

To do so, suppose that A finds a collision with non-negligible probability ϵ. We construct a B that finds the discrete logarithm also with probability ϵ.

Note first that if $h_i(x, b) = h_i(x', b)$, then $y^b g^x \bmod p = y^b g^{x'} \bmod p$, so $g^x \bmod p = g^{x'} \bmod p$, so $x = x'$.

[2]This attack gets its name from the *birthday paradox*, which uses a similar analysis to show that with 23 randomly chosen people, the probability of two of them having the same birthday is greater than 50%.

[3]Suppose that h is a hash function that compresses by one bit. Note that the naïve algorithm that applies h k times to an $n + k$ bit string is not secure, although it compresses by more than 1 bit, because in this case m and $h(m)$ both hash to the same value.

So, for a collision to occur, one value of the second parameter of h must be 1 and the other value of the second parameter of h must be 0. That is, for any collision $(x, b) = (x', b')$, it holds that $b \neq b'$. Without loss of generality, assume that $b = 0$. Then,

$$g^x = yg^{x'} \bmod p$$

so

$$y = g^{x-x'} \bmod p$$

which allows B to find the discrete logarithm of y. $B(p, g, y)$ calls $A(p, g, y) \rightarrow (x, b), (x', b')$. If $b = 0$, then B returns $x - x'$, and otherwise it returns $x' - x$. □

Thus we have constructed a CRH that compresses by one bit. Note further that this reduction is actually an algorithm for computing the discrete logarithm that is better than brute force: since the Birthday Attack on a CRH only requires searching $2^{k/2}$ keys rather than 2^k, the same attack works on the discrete logarithm by applying the above algorithm each time. Of course, there are much better (even deterministic) attacks on the discrete logarithm problem.[4]

5.5.3 Multiple-bit Compression

Given a CRHF function that compresses by one bit, it is possible to construct a CRHF function that compresses by polynomially-many bits. The idea is to apply the simple one-bit function repeatedly.

5.6 A One-Time Digital Signature Scheme for $\{0, 1\}^*$

We now use a family of Collision-Resistant Hash Functions (CRHFs) to construct a one-time signature scheme for messages in $\{0, 1\}^*$. Digital signature schemes that operate on the hash of a message are said to be in the *hash-and-sign* paradigm.

Theorem 116.1. *If there exists a CRH from* $\{0, 1\}^* \longrightarrow \{0, 1\}^n$ *and there exists a one-way function (OWF), then there exists a one-time secure digital signature scheme for* $\{0, 1\}^*$.

[4] Note that there is also a way to construct a CRH from the Factoring assumption:

$$h_{N,y}(x, b) = y^b x^2 \bmod N$$

Here, however, there is a trivial collision if we do not restrict the domain : x and $-x$ map to the same value. For instance, we might take only the first half of the values in $\mathbb{Z}_p^*$.

We define a new one-time secure digital signature scheme $(\mathsf{Gen}', \mathsf{Sign}', \mathsf{Ver}')$ for $\{0,1\}^*$ by

117.0: ONE-TIME DIGITAL SIGNATURE FOR $\{0,1\}^*$

$\mathsf{Gen}'(1^n)$: Run the generator $(pk, sk) \leftarrow \mathsf{Gen}_{\mathrm{Sig}}(1^n)$ and sampling function $i \leftarrow \mathsf{Gen}_{\mathrm{CRH}}(1^n)$. Output $\mathrm{pk}' = (\mathrm{pk}, i)$ and $\mathrm{sk}' = (\mathrm{sk}, i)$.

$\mathsf{Sign}'_{sk}(m)$: Sign the hash of message m: ouptut $\mathsf{Sign}_{\mathrm{sk}}(h_i(m))$.

$\mathsf{Ver}'_{pk}(\sigma, m)$: Verify σ on the hash of m: Output $\mathsf{Ver}_{\mathrm{pk}}(h_i(m), \sigma)$

Proof. We will only provide a sketch of the proof here.

Let $\{h_i\}_{i \in I}$ be a CRH with sampling function $\mathsf{Gen}_{\mathrm{CRH}}(1^n)$, and let $(\mathsf{Gen}_{\mathrm{Sig}}, \mathsf{Sign}, \mathsf{Ver})$ be a one-time secure digital signature scheme for $\{0,1\}^n$ (as constructed in the previous sections.)

Now suppose that there is a p.p.t. adversary A that breaks $(\mathsf{Gen}', \mathsf{Sign}', \mathsf{Ver}')$ with non-negligible probability ϵ after only one oracle call m to Sign'. To break this digital signature scheme, A must output $m' \neq m$ and σ' such that $\mathsf{Ver}'_{\mathrm{pk}'}(m', \sigma') = \mathsf{accept}$ (so $\mathsf{Ver}_{\mathrm{pk}}(h_i(m'), \sigma') = \mathsf{accept}$). There are only two possible cases:

1. $h(m) = h(m')$.

   In this case, A found a collision (m, m') in h_i, which is known to be hard, since h_i is a member of a CRH.

2. A never made any oracle calls, or $h(m) \neq h(m')$.

   Either way, in this case, A obtained a signature σ' using $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ to a new message $h(m')$. But obtaining such a signature violates the assumption that $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ is a one-time secure digital signature scheme.

To make this argument more formal, turn the two cases above into two adversaries B and C. Adversary B tries to invert a hash function from the CRH, and C tries to break the digital signature scheme.

$B(1^n, i)$ operates as follows to find a collision for h_i.

- Generate keys $\mathrm{pk}, \mathrm{sk} \leftarrow \mathsf{Gen}_{\mathrm{Sig}}(1^n)$
- Call A to get $m', \sigma' \leftarrow A^{\mathsf{Sign}_{\mathrm{sk}}(h_i(\cdot))}(1^n, (\mathrm{pk}, i))$.

- Output m, m' where m is the query made by A (if A made no query, then abort).

$C^{\mathsf{Sign}_{\mathrm{sk}}(\cdot)}(1^n, \mathrm{pk})$ operates as follows to break the one-time security of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$.

- Generate index $i \leftarrow \mathsf{Gen}_{\mathrm{CRH}}(1^n)$
- Call A to get $m', \sigma' \leftarrow A(1^n, (\mathrm{pk}, i))$
  - When A calls $\mathsf{Sign}'_{(\mathrm{sk},i)}(m)$, query signing oracle $\mathsf{Sign}_{\mathrm{sk}}(h_i(m))$
- Output $h_i(m')$, σ'.

So, if A succeeds with non-negligible probability, then either B or C must succeed with non-negligible probability. □

5.7 *Signing Many Messages

Now that we have extended one-time signatures on $\{0,1\}^n$ to operate on $\{0,1\}^*$, we turn to increasing the number of messages that can be signed. The main idea is to generate new keys for each new message to be signed. Then we can still use our one-time secure digital signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$. The disadvantage is that the signer must keep state to know which key to use and what to include in a given signature.

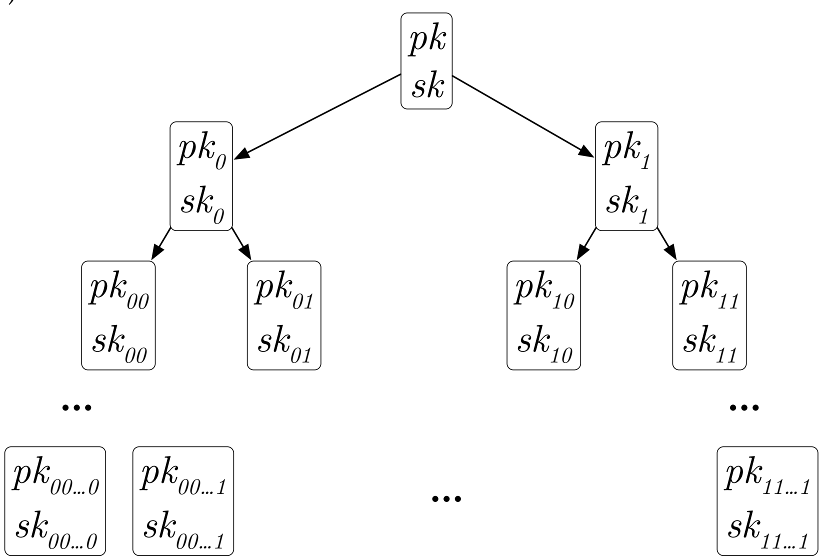
We start with a pair $(\mathrm{pk}_0, \mathrm{sk}_0) \leftarrow \mathsf{Gen}(1^n)$. To sign the first message m_1, we perform the following steps:

- Generate a new key pair for the next message: $\mathrm{pk}_1, \mathrm{sk}_1 \leftarrow \mathsf{Gen}(1^n)$
- Create signature $\sigma_1 = \mathsf{Sign}_{\mathrm{sk}_0}(m_1 \mid\mid \mathrm{pk}_1)$ on the concatenation of message m_1 and new public key pk_1.
- Output $\sigma'_1 = (1, \sigma_1, m_1, \mathrm{pk}_1)$

Thus, each signature attests to the next public key. Similarly, to sign second message m_2, we generate $\mathrm{pk}_2, \mathrm{sk}_2 \leftarrow \mathsf{Gen}(1^n)$, set $\sigma_2 = \mathsf{Sign}_{\mathrm{sk}_1}(m_2 \mid\mid \mathrm{pk}_2)$, and output $\sigma'_2 = (2, \sigma_2, \sigma'_1, m_2, \mathrm{pk}_2)$. Notice that we need to include σ'_1 (the previous signature) to show that the previous public key is correct. These signatures satisfy many-message security, but the signer must keep state, and signature size grows linearly in the number of signatures ever performed by the signer. Proving that this digital signature scheme is many-message secure is left as an exercise. We now focus on how to improve this basic idea by keeping the size of the signature constant.

5.7.1 Improving the Construction

A simple way to improve this many-message secure digital signature scheme is to attest to two new key pairs instead of one at each step. This new construction builds a balanced binary tree of depth n of key pairs, where each node and leaf in the tree is associated with one public-private key pair pk, sk, and each non-leaf node public key is used to attest to its two child nodes. Each of the 2^n leaf nodes can be used to attest to a message. Such a digital signature algorithm can perform up to 2^n signatures with signature size n (the size follows because a signature using a particular key pair $\mathrm{pk}_i, \mathrm{sk}_i$ must provide signatures attesting to each key pair on the path from $\mathrm{pk}_i \mathrm{sk}_i$ to the root). The tree looks as follows.



To sign the first message m, the signer generates and stores $\mathrm{pk}_0, \mathrm{sk}_0$, $\mathrm{pk}_{00}, \mathrm{sk}_{00}$, . . ., $\mathrm{pk}_{0^n}, \mathrm{sk}_{0^n}$ along with all of their siblings in the tree. Then pk_0 and pk_1 are signed with sk, producing signature σ_0, pk_{00} and pk_{01} are signed with sk_0, producing signature σ_1, and so on. Finally, the signer returns the signature

$$\sigma = (\mathrm{pk}, \sigma_0, \mathrm{pk}_0, \sigma_1, \mathrm{pk}_{00}, \ldots, \sigma_{n-1}, \mathrm{pk}_{0^n}, \mathsf{Sign}_{\mathrm{sk}_{0^n}}(m)$$

as a signature for m. The verification function Ver then uses pk to check that σ_0 attests for pk_0, uses pk_0 to check that σ_1 attests for pk_{00}, and so on up to pk_{0^n}, which is used to check that $\mathsf{Sign}_{\mathrm{sk}_{0^n}}(m)$ is a correct signature for m.

For an arbitrary message, the next unused leaf node in the tree is chosen, and any needed signatures attesting to the path from that leaf to the root are generated (some of these signatures will have been generated previously).

Then the leaf node key is used to sign the message in the same manner as above

Proving that this scheme is many-message secure is left as an exercise for the student. The key idea is that fact that $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ is one-time secure, and each signature is only used once. Thus, forging a signature in this scheme requires creating a second signature.

For all its theoretical value, however, this many-message secure digital signature scheme still requires the signer to keep a significant amount of state. The state kept by the signer is

- The number of messages signed

  To remove this requirement, we will assume that messages consist of at most n bits. Then, instead of using the leaf nodes as key pairs in increasing order, use the n-bit representation of m to decide which leaf to use. That is, use $\mathrm{pk}_m, \mathrm{sk}_m$ to sign m.

- All previously generated keys

- All previously generated signatures (for the authentication paths to the root)

We can remove the requirement that the signer remembers the previous keys and previous signatures if we have a pseudo-random function to regenerate all of this information on demand. In particular, we generate a public key pk and secret key sk'. The secret key, in addition to containing the secret key sk corresponding to pk, also contains two seeds s_1 and s_2 for two pseudo-random functions f and g. We then generate pk_i and sk_i for node i by using $f_{s_1}(i)$ as the randomness in the generation algorithm $\mathsf{Gen}(1^n)$. Similarly, we generate any needed randomness for the signing algorithm on message m with $g_{s_2}(m)$. Then we can regenerate any path through the tree on demand without maintaining any of the tree as state at the signer.

5.8 Intuition for Constructing Efficient Digital Signature

Consider the following method for constructing a digital signature scheme from a trapdoor permutation:

- $\mathsf{Gen}(1^n)$: $pk = i$ and $sk = t$, the trapdoor.

- $\mathsf{Sign}_{Sk}(m) = f^{-1}(m)$ using t.

- $\mathsf{Ver}_{pk}(m, \sigma)$ = "accept" if $f_i(\sigma) = m$.

The above scheme is not secure if the adversary is allowed to choose the message to be forged. Picking $m = f_i(0)$ guarantees that 0 is the signature of m. If a specific trapdoor function like RSA is used, adversaries can forge a large class of messages. In the RSA scheme,

- $\mathsf{Gen}(1^n)$: $pk = e, N$ and $sk = d, N$, such that $ed = 1 \bmod \Phi(N)$, and $N = pq$, p, q primes.
- $\mathsf{Sign}_{Sk}(m) = m^d \bmod N$.
- $\mathsf{Ver}_{pk}(m, \sigma)$ = "accept" if $\sigma^e = m \bmod N$.

Given signatures on $\sigma_1 = m_1^d \bmod N$ and $\sigma_2 = m_2^d \bmod N$ an adversay can easily forge a signature on $m_1 m_2$ by multiplying the two signatures modulo N.

To avoid such attacks, in practice, the message is first hashed using some "random looking" function h to which the trapdoor signature scheme can applied. It is secure if h is a random function RF. (In particular, such a scheme can be proven secure in the Random Oracle Model.) We cannot, however, use a pseudorandom function, because to evaluate the PRF, the adversary would have to know the hashing function and hence the seed of the PRF. In this case, the PRF ceases to be computationally indistingiushable from a random function RF. Despite these theoretical problems, this hash-and-sign paradigm is used in practice using SHA1 or SHA256 as the hash algorithm.

5.9 Zero-knowledge Authentication

Chapter 7

Composability

7.1 Composition of Encryption Schemes

7.1.1 CCA-Secure Encryption

So far, we have assumed that the adversary only captures the ciphertext that Alice sends to Bob. In other words, the adversaries attack is a *ciphertext only* attack. One can imagine, however, a variety of stronger attack models. We list some of these models below:

Attack models:

- Ciphertext only attack – this is what we considered so far.
- Known plaintext attack – The adversary may get to see pairs of form $(m_0, Enc_k(m_0)) \ldots$
- Chosen plain text (CPA) – The adversary gets access to an encryption oracle before and after selecting messages.
- Chosen ciphertext attack

  CCA1: ("Lunch-time attack") The adversary has access to an encryption oracle and to a decryption oracle before selecting the messages. (due to Naor and. Yung)

  CCA2: This is just like a CCA1 attack except that the adversary also has access to decryption oracle after selecting the messages. It is not allowed to decrypt the challenge ciphertext however. (introduced by Rackoff and Simon)

Fortunately, all of these attacks can be abstracted and captured by a simple definition which we present below. The different attacks can be captured by allowing the adversary to have *oracle*-access to a special function which allows it to mount CPA/CCA1/CCA2-type attacks.

Definition 136.1 (Secure encryption CPA / CCA1 / CCA2). Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme. Let the random variable $IND_b^{O_1,O_2}(\Pi, \mathcal{A}, n)$ where $\mathcal{A}$ is a non-uniform p.p.t., $n \in N,\ b \in \{0, 1\}$ denote the output of the following experiment:

$$\begin{aligned}
&\mathsf{IND}_b^{O_1,O_2}(\Pi, ma, n)\\
&\quad k \leftarrow \mathsf{Gen}(1^n)\\
&\quad m_0,\ m_1,\ \mathsf{state} \leftarrow A^{O_1(k)}(1^n)\\
&\quad c \leftarrow \mathsf{Enc}_k(m_b)\\
&\quad \text{Output } A^{O_2(k)}(c, \mathsf{state})
\end{aligned}$$

Then we say π is CPA/CCA1/CCA2 secure if $\forall$ non-uniform p.p.t. p.p.t. A:

$$\left\{\mathsf{IND}_0^{O_1,O_2}(\pi, A, n)\right\}_n \approx \left\{\mathsf{IND}_1^{O_1,O_2}(\pi, A, n)\right\}_n$$

where O_1 and O_2 are defined as:

CPA	$[\mathsf{Enc}_k;\ \mathsf{Enc}_k]$
CCA1	$[\mathsf{Enc}_k, \mathsf{Dec}_k;\ \mathsf{Enc}_k]$
CCA2	$[\mathsf{Enc}_k, \mathsf{Dec}_k;\ \mathsf{Enc}_k, \mathsf{Dec}_k]$

Additionally, in the case of CCA2 attacks, the decryption oracle returns $\perp$ when queried on the challenge ciphertext c.

7.1.2 A CCA1-Secure Encryption Scheme

We will now show that the encryption scheme presented in construction 78.1 satisfies a stronger property than claimed earlier. In particular, we show that it is CCA1 secure (which implies that it is also CPA-secure).

Theorem 136.1. *π in construction 78.1 is CPA and CCA1 secure.*

Proof. Consider the encryption scheme $\pi^{RF} = (Gen^{RF}, \mathsf{Enc}^{RF}, \mathsf{Dec}^{RF})$, which is derived from π by replacing PRF f_k in π by truly random function. π^{RF} is CPA and CCA1 secure. Because the adversary only has access to encryption oracle after chosen m_0 and m_1. The only chance adversary can differentiate

$\mathsf{Enc}_k(m_0) = r_0||m_0 \oplus f(r_0)$ and $\mathsf{Enc}_k(m_1) = r_1||m_1 \oplus f(r_1)$ is that the encryption oracle happens to have sampled the same r_0 or r_1 in some previous query, or additionally, in CCA1 attack, the attacker happens to have asked decryption oracle to decrypt ciphertext like $r_0||m$ or $r_1||m$. All cases have only negligible probabilities.

Given $\pi^R F$ is CPA and CCA2 secure, then so is π. Otherwise, if there exists one distinguisher D that can differentiate the experiment results ($\mathsf{IND}_0^{\mathsf{Enc}_k;\mathsf{Enc}_k}$ and $\mathsf{IND}_1^{\mathsf{Enc}_k;\mathsf{Enc}_k}$ in case of CPA attack, while $\mathsf{IND}_0^{\mathsf{Enc}_k,\mathsf{Dec}_k;\mathsf{Enc}_k}$ and $\mathsf{IND}_1^{\mathsf{Enc}_k,\mathsf{Dec}_k;\mathsf{Enc}_k}$ in case of CCA1 attack) then we can construct another distinguisher which internally uses D to differentiate PRF from truly random function. □

7.1.3 A CCA2-Secure Encryption Scheme

However, the encryption scheme π is not CCA2 secure. Consider the attack: in experiment $\mathsf{IND}_b^{\mathsf{Enc}_k,\mathsf{Dec}_k;\mathsf{Enc}_k,Dec_k}$, given ciphertext $r||c = Enc_k(m_b)$, the attacker can ask the decryption oracle to decrypt $r||c+1$. As this is not the challenge itself, this is allowed. Actually $r||c+1$ is the ciphertext for message m_b+1, as

$$\mathsf{Enc}_k(m_b+1) = r||(m_b+1) \oplus f_k(r) = r||m_b \oplus f_k(r) + 1 = r||c+1$$

Thus the decryption oracle would reply $m_b + 1$. The adversary can differentiate which message's encryption it is given.

We construct a new encryption scheme that is CCA2 secure. Let $\{f_s\}$ and $\{g_s\}$ be families of PRF on space $\{0,1\}^{|s|} \to \{0,1\}^{|s|}$.
$\pi' = (Gen',\ Enc',\ Dec')$:

Algorithm 9: Many-message CCA2-secure Encryption Scheme

Assume $m \in \{0,1\}^n$ and let $\{f_k\}$ be a PRF family
$\mathsf{Gen}(1^n) : k_1, k_2 \leftarrow U_n$
$\mathsf{Enc}_{k_1,k_2}(m)$: Pick $r \leftarrow U_n$. Set $c_1 \leftarrow m \oplus f_{k_1}(r)$. Output $(r, c_1, f_{k_2}(c))$
$\mathsf{Dec}_{k_1,k_2}((r, c_1, c_2))$: If $f_{k_2}(c_1) \neq c_2$, output $\perp$. Else output $c_1 \oplus f_{k_1}(r)$

Now we show that:

Theorem 137.1. *π' is CCA2 attack secure.*

Proof. The main idea is to prove by contradiction. In specific, if there is an CCA2 attack on π', then there is an CPA attack on π, which would contradict with the fact that π is CPA secure.

A CCA2 attack on π' is a p.p.t. machine A', s.t. it can differentiate $\left\{\mathsf{IND}_0^{Enc_k,Dec_k;Enc_k,Dec_k}\right\}$ and $\left\{\mathsf{IND}_1^{Enc_k,Dec_k;Enc_k,Dec_k}\right\}$. Visually, it works as that in figure **??**. The attacker A' needs accesses to the Enc'_k and Dec'_k oracles. To built an CPA attack on π, we want to construct another machine A as depicted in figure **??**. To leverage the CCA2 attacker A', we simulate A as in figure **??** which internally uses A'.

Formally, the simulator works as follows:

- Whenever A' asks for an encryption of message m, A asks its own encryption oracle Enc_{s_1} to get $c_1 = Enc_{s_1}(m)$. But A' expects encryption $c_1||c_2$, requiring s_2 to evaluate $g_{s_2}(c_1)$, which A has no access to. Thus Let $c_2 \leftarrow \{0,1\}^n$, and reply $c_1||c_2$.

- Whenever A' asks for a decryption $c_1||c_2$. If we previously gave A' $c_1||c_2$ to answer an encryption query of some message m, then reply m, otherwise reply $\perp$.

- Whenever A' outputs $m_0,\ m_1$, output $m_0,\ m_1$.

- Upon receiving c, feed $c||r$, where $r \leftarrow \{0,1\}^n$ to A'.

- Finally, output A''s output.

Consider encryption scheme $\pi'^{RF} = (Gen'^{RF}, Enc'^{RF}, Dec'^{RF})$ which is derived from π' by replacing every appearance of g_{s_2} with a truly random function.

Note that the simulated Enc' is just Enc'^{RF}, and Dec' is very similar to Dec'^{RF}. Then A' inside the simulator is nearly conducting CCA2 attack on π'^{RF} with the only exception when A' asks an $c_1||c_2$ to Dec' which is not returned by a previous encryption query and is a correct encryption, in which case Dec' falsely returns $\perp$. However, this only happens when $c_2 = f(c_1)$, where f is the truly random function. Without previous encryption query, the attacker can only guess the correct value of $f(c_1)$ w.p. $\frac{1}{2^n}$, which is negligible.

Thus we reach that: if A' breaks CCA2 security of π'^{RF}, then it can break CPA security of π. The premise is true as by assumption A' breaks CCA2 security of π', and that PRF is indistinguishable from a truly random function. $\square$

7.1.4 CCA-secure Public-Key Encryption

We can also extend the notion of CCA security to public-key encryption schemes.Note that, as the adversary already knows the the public key, there is no need to provide it with an encryption oracle.

Definition 139.1 (Secure Public Key Encryption). Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, Dec)$ be a public key encryption. Let the random variable $\mathsf{Ind}_b(\Pi, ma, 1^n)$ where $\mathcal{A}$ is a non-uniform p.p.t. adversary, $n \in \mathbb{N}$, and $b \in \{0, 1\}$ denote the output of the following experiment:

$$
\begin{aligned}
&\mathsf{Ind}_b(\Pi, \mathcal{A}, n)\\
&\quad (pk, sk) \leftarrow \mathsf{Gen}(1^n)\\
&\quad m_0,\ m_1,\ \mathsf{state} \leftarrow A^{O_1(sk)}(1^n, pk)\\
&\quad c \leftarrow \mathsf{Enc}_{pk}(m_b)\\
&\quad \text{Output } A^{O_2(k)}(c, \text{state})
\end{aligned}
$$

We say that Π is CPA/CCA1/CCA2 secure if for all non-uniform p.p.t. $\mathcal{A}$, the following two distributions are computationally indistinguishable:

$$\{\mathsf{Ind}_0(\Pi, \mathcal{A}, n)\}_{n\in\mathbb{N}} \approx \{\mathsf{Ind}_1(\Pi, \mathcal{A}, n)\}_{n\in\mathbb{N}}$$

The oracles O_1, O_2 are defined as follows:

CPA	$[\cdot, \cdot]$
CCA1	$[\mathsf{Dec}, \cdot]$
CCA2	$[\mathsf{Dec}, \mathsf{Dec}^*]$

where Dec^* answers all queries except for the challenge ciphertext c.

It is not hard to see that the encryption scheme in Construction 82.1 is CPA secure. CCA2 secure public-key encryption schemes are, however, significantly hard to construct; such contructions are outside the scope of this chapter.

7.1.5 Non-Malleable Encryption

Until this point we have discussed encryptions that prevent a passive attacker from discovering any information about messages that are sent. In some situations, however, we may want to prevent an attacker from creating a new message from a given encryption.

Consider an auction for example. Suppose the Bidder Bob is trying to send a message containing his bid to the Auctioneer Alice. Private key encryption could prevent an attacker Eve from knowing what Bob bids, but if she could construct a message that contained one more than Bob's bid, then she could win the auction.

We say that an encryption scheme that prevents these kinds of attacks is *non-malleable*. Informally, if a scheme is non-malleable, then it is impossible to output an encrypted message containing any function of a given encrypted message. Formally, we have the following definition:

Definition 140.1 (Non-Malleability). Let (Gen, Enc, Dec) be an encryption scheme. Define the following experiment:

$$\begin{aligned}
&\mathsf{NM}_b(\Pi, ma, n)\\
&\quad k \leftarrow \mathsf{Gen}(1^n)\\
&\quad m_0,\ m_1,\ \mathsf{state} \leftarrow A^{O_1(k)}(1^n)\\
&\quad c \leftarrow \mathsf{Enc}_k(m_b)\\
&\quad c_1', c_2', c_3', \ldots, c_\ell' \leftarrow A^{O_2(k)}(c, \mathsf{state})\\
&\quad m_i' \leftarrow \begin{cases} \perp & \text{if } c_i = c \\ \mathsf{Dec}_k(c_i') & otherwise \end{cases}\\
&\text{Output } (m_1', m_2', \ldots, m_\ell')
\end{aligned}$$

Then $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is *non-malleable* if for every non-uniform p.p.t. $\mathcal{A}$, and for every non-uniform p.p.t. $\mathcal{D}$, there exists a negligible ϵ such that for all $m_0, m_1 \in \{0,1\}^n$,

$$\Pr[\mathcal{D}(NM_0(\Pi, \mathcal{A}, n)) = 1] - \Pr[\mathcal{D}(NM_1(\Pi, \mathcal{A}, n)) = 1] \leq \epsilon(n)$$

One non-trivial aspect of this definition is the conversion to $\perp$ of queries that have already been made (step 4). Clearly without this, the definition would be trivially unsatisfiable, because the attacker could simply "forge" the encryptions that they have already seen by replaying them.

7.1.6 Relation-Based Non-Malleability

We chose this definition because it mirrors our definition of secrecy in a satisfying way. However, an earlier and arguably more natural definition can be given by formalizing the intuitive notion that the attacker cannot output an encryption of a message that is related to a given message. For example, we might consider the relation $R_{\text{next}}(x) = \{x+1\}$, or the relation $R_{\text{within-one}}(x) = \{x-1, x, x+1\}$. We want to ensure that the encryption of x does not help the attacker encrypt an element of $R(x)$. Formally:

Definition 140.2 (Relation-Based Non-Malleability). We say that an encryption scheme (Gen, Enc, Dec) is relation based non-malleable if for every p.p.t. adversary $\mathcal{A}$ there exists a p.p.t. simulator $\mathcal{S}$ such that for all p.p.t.-recognizable relations R, there exists a negligible ϵ such that for all $m \in \mathcal{M}$ with $|m| = n$, and for all z, it holds that

$$\left| \begin{array}{l} \Pr[NM(A(z), m) \in R(m)] \\ \quad - \quad \Pr[k \leftarrow Gen(1^n); c \leftarrow \mathcal{S}(1^n, z); m' = Dec_k(c) : m' \in R(m)] \end{array} \right| < \epsilon$$

where i ranges from 1 to a polynomial of n and NM is defined as above.

This definition is equivalent to the non-relational definition given above.

Theorem 141.1. *(Enc, Dec, Gen) is a non-malleable encryption scheme if and only if it is a relation-based non-malleable encryption scheme.*

Proof. $(\Rightarrow)$ Assume that the scheme is non-malleable by the first definition. For any given adversary $\mathcal{A}$, we need to produce a simulator $\mathcal{S}$ that hits any given relation R as often as $\mathcal{A}$ does. Let $\mathcal{S}$ be the machine that performs the first 3 steps of $NM(\mathcal{A}(z), m')$ and outputs the sequence of cyphertexts, and let $\mathcal{D}$ be the distinguisher for the relation R. Then

$$\begin{aligned} &|\Pr[NM(\mathcal{A}(z), m) \in R(m)] \quad - \quad \Pr[k \leftarrow Gen(1^n); c \leftarrow \mathcal{S}(1^n, z); m' = Dec_k(c) : m' \in R(m)]| \\ &\quad = |\Pr[\mathcal{D}(NM(\mathcal{A}(z), m))] \quad - \quad \Pr[\mathcal{D}(NM(\mathcal{A}(z), m'))]| \leq \epsilon \end{aligned}$$

as required.

$(\Leftarrow)$ Now, assume that the scheme is relation-based non-malleable. Given an adversary $\mathcal{A}$, we know there exists a simulator $\mathcal{S}$ that outputs related encryptions as well as $\mathcal{A}$ does. The relation-based definition tells us that $NM(\mathcal{A}(z), m_0) \approx Dec(\mathcal{S}())$ and $Dec(\mathcal{S}()) \approx NM(\mathcal{A}(z), m_1)$. Thus, by the hybrid lemma, $NM(\mathcal{A}(z), m_0) \approx NM(\mathcal{A}(z), m_1)$ which is the first definition of non-malleability. □

7.1.7 Non-Malleability and Secrecy

Note that non-malleability is a distinct concept from secrecy. For example, one-time pad is perfectly secret, yet is not non-malleable (since one can easily produce the encryption of $a \oplus b$ give then encryption of a, for example). However, if we consider security under CCA2 attacks, then the two definitions coincide.

Theorem 141.2. *An encryption scheme $\Sigma = (Enc, Dec, Gen)$ is CCA2 secret if and only if it is CCA2 non-malleable*

Proof sketch. If Σ is not CCA2 non-malleable, then a CCA2 attacker can break secrecy by changing the provided encryption into a related encryption, using the decryption oracle on the related message, and then distinguishing the unencrypted related messages. Similarly, if Σ is not CCA2 secret, then a CCA2 attacker can break non-malleability by simply decrypting the cyphertext, applying a function, and then re-encrypting the modified message. $\square$

7.2 Composition of Zero-knowledge Proofs*

7.2.1 Sequential Composition

Whereas the definition of zero knowledge only talks about a *single* execution between a prover and a verifier, the definitions is in fact closed under sequential composition; that is, sequential repetitions of a ZK protocol results in a new protocol that still remains ZK.

Theorem 142.1 (Sequential Composition Theorem). *Let (P, V) be a perfect/computational zero-knowledge proof for the language L. Let $Q(n)$ be a polynomial, and let (P_Q, V_Q) be an interactive proof (argument) that on common input $x \in \{0,1\}^n$ proceeds in $Q(n)$ phases, each on them consisting of an execution of the interactive proof (P, V) on common input x (each time with independent random coins). Then (P_Q, V_Q) is an perfect/computational* ZK *interactive proof.*

Proof. (sketch) Consider a malicious verifier V^{Q*}. Let $V^*(x, z||r||(\bar{m}_1, .., \bar{m}_i))$ denote the machine that runs $V^{Q*}(x, z)$ on input the random tape r and feeds it the messages $(\bar{m}_1, .., \bar{m}_i)$ as part of the i first iterations of (P, V) and runs just as V^{Q*} during the $i+1$ iteration, and then halts. Let S denote the zero-knowledge simulator for V^*. Let $p(\cdot)$ be a polynomial bounding the running-time of V^{Q*}. Condsider now the simulator S^{Q*} that proceeds as follows on input x, z

- Pick a length $p(|x|)$ random string r.
- Next proceed as follows for $Q(|x|)$ iterations:
  - In iteration i, run $S(x, z||r||(\bar{m}_1, .., \bar{m}_i))$ and let $\bar{m}_{i+1}$ denote the messages in the view output.

The linearity of expectations, the expected running-time of S^Q is polynomial (since the expected running-time of S is). A standard hybrid argument can be used to show that the output of S^Q is correctly distributed. $\square$

7.2.2 Parallel/Concurrent Composition

Sequential composition is a very basic notion of compostion. An often more realistic scenario consider the execution of multiple protocols at the same time, with an arbitrary scheduling. As we show in this section, zero-knowledge is not closed under such "concurrent composition". In fact, it is not even closed under "parallel-composition" where all protocols executions start at the same time and are run in a lockstep fashion.

Consider the protocol (P, V) for proving $x \in L$, where P on input x, y and V on input x proceed as follows, and L is a language with a unique witness (for instance, L could be the language consisting of all elements in the range of a $1-1$ one-way function f, and the associated witness relation is $R_L(x) = \{y | f(y) = x\}$.

143.1: ZK PROTOCOL THAT IS NOT CONCURRENTLY SECURE

$P \rightarrow V$ P provides a zero-knowledge proof of knowledge of $x \in L$.

$P \leftarrow V$ V either "quits" or starts a zero-knowledge proof of knowledge $x \in L$.

$P \rightarrow V$ If V provides a convincing proof, P reveals the witness y.

It can be shown that the (P, V) is zero-knowledge; intuitively this follows from the fact that P only reveals y in case the verifier already knows the witness. Formally, this can be shown by "extracting" y from any verifier V^* that manages to convince P. More precisely, the simulator S first runs the simulator for the ZK proof in step 1; next, if V^* produces an accepting proof in step 2, S runs the extractor on V^* to extract a witness y' and finally feeds the witness to y'. Since by assumption L has a unique witness it follows that $y = y'$ and the simulation will be correctly distributed.

However, an adversary A that participates in two concurrent executions of (P, V), acting as a verifier in both executions, can easily get the witness y even if it did not know it before. A simply schedules the messages such that the zero-knowledge proof that the prover provides in the first execution is forwarded as the step 2 zero-knowledge proof (by the verifier) in the second execution; as such A convinces P in the second execution that it knows a witness y (although it is fact only is relaying messages from the the other prover, and in reality does not know y), and as a consequence P will reveal the witness to A.

The above protocol can be modified (by padding it with dummy messages) to also give an example of a zero-knowledge protocol that is not secure under even two parallel executions.

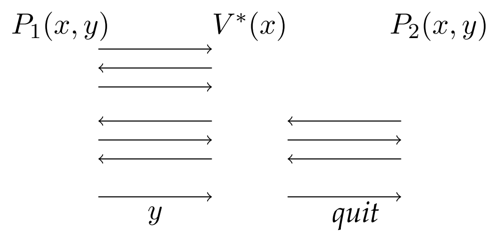


Figure 7.1: A Message Schedule which shows that protocol 143.1 does not concurrently compose. The Verifier feeds the prover messages from the second interaction with P_2 to the first interaction with prover P_1. It therefore convinces the first prover that it "knows" y, and therefore, P_1 sends y to V^*.

7.2.3 Witness Indistinguishability

- Definition
- WI closed under concurrent comp
- ZK implies WI

7.2.4 A Concurrent Identification Protocol

- y_1, y_2 is pk
- x_1, x_2 is sk
- WI POK that you know inverse of either y_1 or y_2.

7.3 Composition Beyond Zero-Knowledge Proofs

7.3.1 Non-malleable commitments

- Mention that standard commitment is malleable
- (could give construction based on adaptive OWP?)

Chapter 8

*More on Randomness and Pseudorandomness

8.1 A Negative Result for Learning

Consider a space $S \subseteq \{0,1\}^n$ and a family of concepts $\{C_i\}_{i \in I}$ such that $C_i \subseteq S$.

The Learning Question: For a random $i \in I$, given samples (x_j, b_j) such that $x_j \in S$ and $b_j = 1$ iff $x_j \in C_i$, determine for a bit string x if $x \in C_i$.

The existence of PRFs shows that there are concepts that can't be learnt.

Theorem 145.1. *There exists a p.p.t. decidable concept that cannot be learnt.*

Proof sketch.

$$\begin{aligned} S &= \{0,1\}^n \\ C_i &= \{x \mid f_i(x)_{|1} = 1\} \qquad f_i(x)_{|1} \text{ is the first bit of } f_i(x) \\ I &= \{0,1\}^n \end{aligned}$$

No (n.u.) p.p.t. can predict whether a new sample x is in C_i better than $\frac{1}{2} + \epsilon$. □

8.2 Derandomization

Traditional decision problems do not need randomness; a randomized machine can be replaced by a deterministic machine that tries all finite random tapes. In fact, we can do better if we make some cryptographic assumptions. For example:

Theorem 145.2. *If pseudo-random generators (PRG) exist, then for every constant* $\varepsilon > 0$, ***BPP*** $\subseteq$ ***DTIME***(2^{n^ε}).

Proof. where **DTIME**$(t(n))$ denotes the set of all languages that can be decided by deterministic machines with running-time bounded by $O(t(n))$.

Given a language $L \in$ **BPP**, let M be a p.p.t.Turing machine that decides L with probability at least 2/3. Since the running time of M is bounded by n^c for some constant c, M uses at most n^c bits of the random tape. Note that we can trivially de-randomize M by deterministically trying out all 2^{n^c} possible random tapes, but such a deterministic machine will take more time than 2^{n^ε}.

Instead, given ε, let $g : \{0,1\}^{n^{\varepsilon/2}} \to \{0,1\}^{n^c}$ be a PRG (with polynomial expansion factor $n^{c-\varepsilon/2}$). Consider a p.p.t.machine M' that does the following given input x:

1. Read $\varepsilon/2$ bits from the random tape, and apply g to generate n^c pseudo-random bits.

2. Simulate and output the answer of M using these pseudo-random bits.

M' must also decide L with probability negligibly close to 2/3; otherwise, M would be a p.p.t.distinguisher that can distinguish between uniform randomness and the output of g.

Since M' only uses $n^{\varepsilon/2}$ random bits, a deterministic machine that simulates M' on all possible random tapes will take time

$$2^{n^{\varepsilon/2}} \cdot \mathrm{poly}(n) \in O(2^{n^\varepsilon})$$

□

Remark: We can strengthen the definition of a PRG to require that the output of a PRG be indistinguishable from a uniformly random string, even when the distinguisher can run in sub-exponential time (that is, the distinguisher can run in time $t(n)$ where $t(n) \in O(2^{n^\varepsilon})$ for all $\varepsilon > 0$). With this stronger assumption, we can show that **BPP** $\subseteq$ **DTIME**$(2^{\mathrm{poly}(\log n)})$, the class of languages that can be decided in *quasi-polynomial* time.

However, for cryptographic primitives, we have seen that randomness is actually required. For example, any deterministic public key encryption scheme must be insecure. But how do we get randomness in the real world? What if we only have access to "impure" randomness?

8.3 Imperfect Randomness and Extractors

In this section we discuss models of imperfect randomness, and how to extract truly random strings from imperfect random sources with *deterministic extractors*.

8.3.1 Extractors

Intuitively, an extractor should be an efficient and deterministic machine that produces truly random bits, given a sample from an imperfect source of randomness. In fact, sometimes we may be satisfied with just "almost random bits", which can be formalized with the notion of ε-closeness.

Definition 147.1 (ε-closeness). Two distributions X and Y are *ε-close*, written $X \approx_\varepsilon Y$, if for every (deterministic) distinguisher D (with no time bound),

$$|\Pr[x \leftarrow X : D(x) = 1] - \Pr[y \leftarrow Y : D(y) = 1]| \leq \varepsilon$$

Definition 147.2 (ε-extractors). Let C be a set of distributions over $\{0,1\}^n$. An *m-bit ε-extractor* for C is a deterministic function $\mathrm{Ext} : \{0,1\}^n \to \{0,1\}^m$ that satisfies the following:

$$\forall X \in C,\ \{x \leftarrow X : \mathrm{Ext}(x)\} \approx_\varepsilon U_m$$

where U_m is the uniform distribution over $\{0,1\}^m$.

8.3.2 Imperfect Randomness

An obvious example of imperfect randomness is to repeatedly toss a biased coin; every bit in the string would be biased in the same manner (i.e. the bits are independently and identically distributed). Van Neumann showed that the following algorithm is a 0-extractor (i.e. algorithm produces truly random bits): Toss the biased coin twice. Output 0 if the result was 01, output 1 if the result was 10, and repeat the experiment otherwise.

A more exotic example of imperfect randomness is to toss a sequence of different biased coins; every bit in the string would still be independent, but not biased the same way. We do not know any 0-extractor in this case. However, we can get a ε-extractor by tossing a sufficient large number of coins at once and outputting the XOR of the results.

More generally, one can consider distributions of bit strings where different bits are not even independent (e.g. bursty errors in nature). Given an imperfect source, we would like to have a measure of its "amount of randomness". We first turn to the notion of *entropy* in physics:

Definition 147.3 (entropy). Given a distribution X, the *entropy* of X, denoted by $H(x)$ is defined as follows:

$$H(X) = \mathbb{E}\left[x \leftarrow X : \log\left(\frac{1}{\Pr[X = x]}\right)\right] = \sum_x \Pr[X = x] \log\left(\frac{1}{\Pr[X = x]}\right)$$

When the base of the logarithm is 2, $H(x)$ is the Shannon entropy of X.

Intuitively, Shannon entropy measures how many truly random bits are "hidden" in X. For example, if X is the uniform distribution over $\{0,1\}^n$, X has Shannon entropy

$$H(X) = \sum_{x \in \{0,1\}^n} \Pr[X = x] \log_2\left(\frac{1}{\Pr[X = x]}\right) = 2^n(2^{-n} \cdot n) = n$$

As we will soon see, however, a source with high Shannon entropy can be horrible for extractors. For example, consider X defined as follows:

$$X = \begin{cases} 0^n & \text{w.p. } 0.99 \\ \text{uniformly random element in } \{0,1\}^n & \text{w.p. } 0.01 \end{cases}$$

Then, $H(X) \approx 0.01n$. However, an extractor that samples an instance from X will see 0^n most of the time, and cannot hope to generate even just one random bit[1]. Therefore, we need a stronger notion of randomness.

Definition 148.1 (Min Entropy). Given a distribution X, the *min entropy* of X, denoted by $H_\infty(x)$, is defined as follows:

$$H_\infty(X) = \min_x \log_2\left(\frac{1}{\Pr[X = x]}\right)$$

Equivalently,

$$H_\infty(X) \geq k \Leftrightarrow \forall x,\ \Pr[X = x] \leq 2^{-k}$$

Definition 148.2 (k-source). A distribution X is a *k-source* if $H_\infty(X) \geq k$. If additionally X is the uniform distribution on 2^k distinct elements, we say X is a *k-flat source*.

Even with this stronger sense of entropy, however, extraction is not always possible.

---

[1] A possible fix is to sample X many times. However, we restrict ourselves to one sample only motivated by the fact that some random sources in nature can not be independently sampled twice. E.g. the sky in the morning is not independent from the sky in the afternoon.

Theorem 148.1. *Let C be the set of all efficiently computable $(n-2)$-sources on $\{0,1\}^n$. Then, there are no 1-bit $1/4$-extractors for C.*

Proof. Suppose the contrary that Ext is a $1/4$-extractor for C. Consider the distribution X generated as follows:

1. Sample $x \leftarrow U_n$. If $\mathrm{Ext}(x) = 1$, output x. Otherwise repeat.
2. After 10 iterations with no output, give up and output a random $x \leftarrow U_n$.

Since $U_n \in C$ and Ext is a $1/4$-extractor, we have

$$\Pr[x \leftarrow U_n : \mathrm{Ext}(x) = 1] \geq 1/2 - 1/4 = 1/4$$

which implies that $|\{x \in \{0,1\}^n : \mathrm{Ext}(x) = 1\}| \geq (1/4)2^n = 2^{n-2}$. We can then characterize X as follows:

$$X = \begin{cases} U_n & \text{w.p.} \leq \left(\frac{3}{4}\right)^{10} \\ \text{uniformly random element in } \{x \in \{0,1\}^n, \mathrm{Ext}(x) = 1\} & \text{otherwise} \end{cases}$$

Since $|\{x \in \{0,1\}^n, \mathrm{Ext}(x) = 1\}| \geq 2^{n-2}$, both cases above are $(n-2)$-sources. This makes X a $(n-2)$-source. Moreover, X is computable in polynomial time since Ext is. This establishes $X \in C$.

On the other hand,

$$\Pr[x \in X : \mathrm{Ext}(x) = 1] \geq 1 - \left(\frac{3}{4}\right)^{10} > 0.9$$

and so $\{x \in X : \mathrm{Ext}(x)\}$ is definite not $1/4$-close to U_1, giving us the contradiction. □

8.3.3 Left-over hash lemma

Appendix A

Basic Probability

Basic Facts

- Events A and B are said to be *independent* if

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

- The *conditional probability* of event A given event B, written as $\Pr[A \mid B]$ is defined as
$$\Pr[A \mid B] = \frac{\Pr[A \cap B]}{\Pr[B]}$$

- *Bayes theorem* relates the $\Pr[A \mid B]$ with $\Pr[B \mid A]$ as follows:
$$\Pr[A \mid B] = \frac{\Pr[B \mid A] \Pr[A]}{\Pr[B]}$$

- Events $A_1, A_2, \ldots, A_n$ are said to be *pairwise independent* if for every i and every $j \neq i$, A_i and A_j are independent.

- *Union Bound:* Let $A_1, A_2, \ldots, A_n$ be events. Then,
$$\Pr[A_1 \cup A_2 \cup \ldots \cup A_n] \leq \Pr[A_1] + \Pr[A_2] + \ldots + \Pr[A_n]$$

- Let X be a random variable with range Ω. The *expectation* of X is the value:
$$E[X] = \sum_{x \in \Omega} x \Pr[X = x]$$
The *variance* is given by,
$$Var[X] = E[X^2] - (E[X])^2$$

- Let $X_1, X_2, \ldots, X_n$ be random variables. Then,

$$E[X_1 + X_2 + \cdots + X_n] = E[X_1] + E[X_2] + \cdots + E[X_n]$$

- If X and Y are *independent random variables*, then

$$\begin{aligned} E[XY] &= E[X] \cdot E[Y] \\ Var[X+Y] &= Var[X] + Var[Y] \end{aligned}$$

Markov's Inequality

If X is a positive random variable with expectation $E(X)$ and $a > 0$, then

$$\Pr[X \geq a] \leq \frac{E(X)}{a}$$

Chebyshev's Inequality

Let X be a random variable with expectation $E(X)$ and variance σ^2, then for any $k > 0$,

$$\Pr[|X - E(X)| \geq k] \leq \frac{\sigma^2}{k^2}$$

Chernoff's inequality

Let $X_1, X_2, \ldots, X_n$ denote independent random variables, such that for all i, $E(X_i) = \mu$ and $|X_i| \leq 1$.

$$\Pr\left[\left|\frac{\sum X_i}{n} - \mu\right| \geq \epsilon\right] \leq 2^{-\epsilon^2 n}$$

A useful corollary of this inequality is the following. Assume you can get independent and identically distributed, but *biased*, samples of a bit b; furthermore, these samples are correct only w.p. $\frac{1}{2} + \frac{1}{poly(n)}$. Then, if you taking $poly(n)$ samples and output the most frequent value b' of the samples, it holds with high probability that $b = b'$. This procedure is often called a *majority vote*.

Pairwise-independent sampling inequality

Let $X_1, X_2, \ldots, X_n$ denote pair-wise independent random variables, such that for all i, $E(X_i) = \mu$ and $|X_i| \leq 1$.

$$\Pr\left[\left|\frac{\sum X_i}{n} - \mu\right| \geq \epsilon\right] \leq \frac{1 - \mu^2}{n\epsilon^2}$$

Note that this is a Chernoff like bound when the random variables are only pairwise independent. The inequality follows as a corollary of Chebyshev's inequality.

Appendix B

Basic Complexity Classes

We recall the definitions of the basic complexity classes **DP**, **NP** and **BPP**.

The Complexity Class DP. We start by recalling the definition of the class **DP**, i.e., the class of languages that can be decided in (deterministic) polynomial-time.

Definition 155.1 (Complexity Class **DP**). A language L is recognizable in (deterministic) polynomial-time if there exists a deterministic polynomial-time algorithm M such that $M(x) = 1$ if and only if $x \in L$. **DP** is the class of languages recognizable in polynomial time.

The Complexity Class NP. We recall the class **NP**, i.e., the class of languages for which there exists a proof of membership that can be verified in polynomial-time.

Definition 155.2 (Complexity Class **NP**). A language L is in **NP** if there exists a Boolean relation $R_L \subseteq \{0,1\}^* \times \{0,1\}^*$ and a polynomial $p(\cdot)$ such that R_L is recognizable in polynomial-time, and $x \in L$ if and only if there exists a string $y \in \{0,1\}^*$ such that $|y| \leq p(|x|)$ and $(x, y) \in R_L$.

The relation R_L is called a *witness relation* for L. We say that y is a witness for the membership $x \in L$ if $(x, y) \in R_L$. We will also let $R_L(x)$ denote the set of witnesses for the membership $x \in L$, i.e.,

$$R_L(x) = \{y : (x, y) \in L\}$$

We let co-**NP** denote the complement of the class **NP**, i.e., a language L is in co-**NP** if the complement to L is in **NP**.

The Complexity Class BPP. We recall the class **BPP**, i.e., the class of languages that can be decided in *probabilistic* polynomial-time (with two-sided error).

Definition 156.1 (Complexity Class **BPP**). A language L is recognizable in probabilistic polynomial-time if there exists a probabilistic polynomial-time algorithm M such that

- $\forall x \in L, \Pr[M(x) = 1] \geq 2/3$
- $\forall x \notin L, \Pr[M(x) = 0] \geq 2/3$

BPP is the class of languages recognizable in probabilistic polynomial time.