A Course in Cryptography

RAFAEL PASS
ABHI SHELAT

Lecture Notes 2007

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

First edition: June 2007

Contents

List of Algorithms and Protocols

List of Definitions

Preface

We would like to thank the students of CS687 for scribing the original lecture notes which served as a starting point for this book.

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.

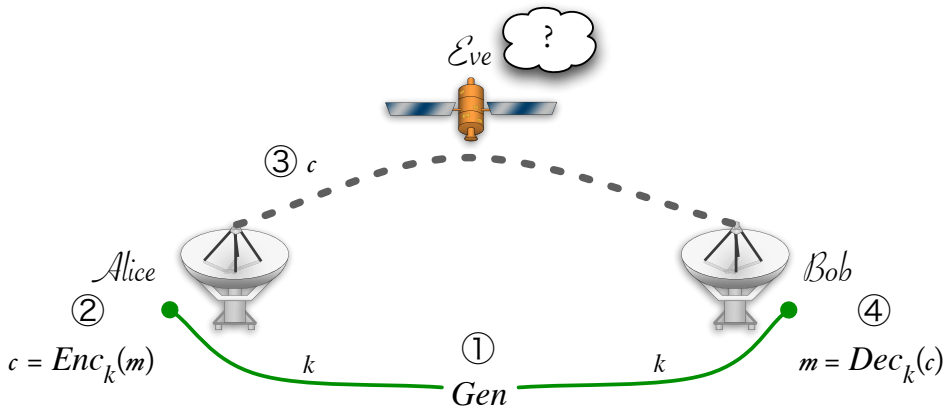


Figure 1.1: Illustration of the steps involved in private-key encryption. First, a key k must be generated by the Gen algorithm and privately given to Alice and Bob. In the picture, this is illustrated with a green "land-line." Later, Alice encodes the message m into a ciphertext c and sends it over 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.

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 $\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}$ 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 $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ 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 $m \in M$ and $k \in K$,

$$Pr[Dec_k(Enc_k(m)) = m] = 1$$

To simplify notation we also say that $(\mathcal{M}, \mathcal{K}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a private-key encryption scheme if $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ 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.

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.

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

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.

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.

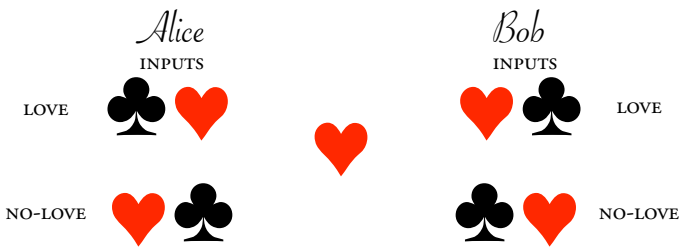


Figure 1.2: Illustration of the Match game with Cards

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

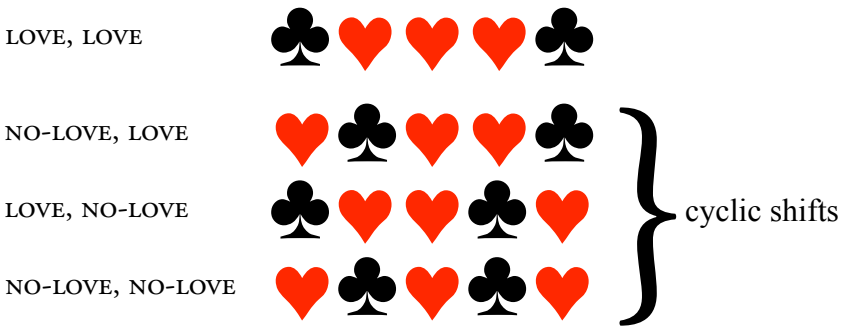


Figure 1.3: The possible outcomes of the Match Protocol. In the case of a mismatch, all three outcomes are cyclic shifts of one-another.

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.

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.

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

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

□

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.

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

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

we conclude that

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

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

Note that 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

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

but

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

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*

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

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 $m_1 \in \mathcal{M}, k \leftarrow \mathcal{K}$,

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

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

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

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

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

□

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 $m_2 \in \mathcal{M}$,

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

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 ex-

plore 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.

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.

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.

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.

Some computationally "easy" and "hard" problem

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 $f : \{0,1\}^* \rightarrow \{0,1\}$ 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 very famous $P \neq NP$ conjecture.

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 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 20.1 (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 $f : \{0,1\}^* \to \{0,1\}^*$ 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 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.

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 22.1 (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 22.1. (Worst-case One-way Function). A function $f : \{0,1\}^* \to \{0,1\}$ 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 $\forall x \ \Pr[\mathcal{A}(f(x)) \in f^{-1}(f(x))] = 1$

We will see that assuming $SAT \notin BPP$, one-way functions according to the above definition must exist. In fact, these two assumptions are equivalent. 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*.

Definition 23.1. 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 23.2 (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 Hardness Amplification

A first candidate for a one-way function is the function $f_{\mathsf{mult}} : \mathbb{N}^2 \to \mathbb{N}$ defined by $f_{\mathsf{mult}}(x, y) = xy$, with $|x| = |y|$. 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. 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.

Definition 24.1 (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.** Any efficient algorithm will fail to invert f on random input with non-negligible probability. More formally, for any non-uniform p.p.t. machine $\mathcal{A} : \{0,1\}^* \to \{0,1\}^*$, there exists a polynomial function $q : \mathbb{N} \to \mathbb{N}$ such that for any input length $n \in \mathbb{N}$,

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

It is conjectured that f_{mult} is a weak one-way function.

Hardness Amplification

Fortunately, a weak version of a one-way function is sufficient for everything because it can be used to produce a strong one-way function. The main insight is that by running a weak one-way function f with enough inputs produces a list which contains at least one member on which it is hard to invert f.

Theorem 25.1. *Given a weak one-way function $f : \{0,1\}^* \to \{0,1\}^*$, there is a fixed $m \in \mathbb{N}$, polynomial in the input length $n \in \mathbb{N}$, such that the following function $f' : (\{0,1\}^n)^m \to (\{0,1\}^n)^m$ is strongly one-way:*

$$f'(x_1, x_2, \ldots, x_m) = (f(x_1), f(x_2), \ldots, f(x_m)).$$

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.

*Proof of Theorem 25.1

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 $n \in \mathbb{N}$,

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

Define $m = 2nq(n)$.
Assume that f' as defined in the theorem 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.,

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

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

$$\mathsf{Pr}\left[x_i \leftarrow \{0,1\}^n; y_i = f(x_i) : f'(\mathcal{A}'(y_1, y_2, \ldots, y_m)) = (y_1, y_2, \ldots, y_m)\right] > \frac{1}{p(n)}. \tag{2.1}$$

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 follows.

(1) Input $y \in \{0,1\}^n$.
(2) Pick a random $i \leftarrow [1, m]$.
(3) For all $j \neq i$, pick a random $x_j \leftarrow \{0,1\}^n$, and let $y_j = f(x_j)$.
(4) Let $y_i = y$.
(5) Let $(z_1, z_2, \ldots, z_m) = \mathcal{A}'(y_1, y_2, \ldots, y_m)$.
(6) 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

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

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 (2.1) 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:

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

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

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

So taking a union bound, we have

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

Also,

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

Hence, $\mathsf{Pr}\left[x_i \leftarrow \{0,1\}^n; y_i = f'(x_i) : \mathcal{A}'(\vec{y}) \text{ succeeds}\right] < \frac{1}{2p(n)} + e^{-n} < \frac{1}{p(n)}$, thus contradicting (2.1). □

This theorem indicates that the existence of weak one-way functions is equivalent to that of strong one-way functions.

2.4 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 28.1. (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 PPT 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 PPT that on input i returns a uniformly random element of $\mathcal{D}_i$
3. It is easy to evaluate, i.e. there exists a PPT that on input $i, x \in \mathcal{D}_i$ computes $f_i(x)$.
4. It is hard to invert, i.e. for any PPT $\mathcal{A}$ there exists a negligible function ϵ such that

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

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 28.1. *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. □

2.5 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.

Modular Arithmetic

We state the following basic facts about modular arithmetic:

Claim 28.1. *For $N > 0$ and $a, b \in \mathbb{Z}$,*

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

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 1: ExtendedEuclid(a, b)

Input: (a, b) s.t $a > b \geq 0$
Output: (x, y) s.t. $ax + by = \gcd(a, b)$

```
if a mod b = 0 then
    Return (0, 1)
else
    (x, y) ← ExtendedEuclid (b, a mod b)
    Return (y, x − y(⌊a/b⌋))
```

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 1 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 29.1. *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 30.1. $\gcd(a, b) = \gcd(b, a \bmod b)$

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.

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 2: ModularExponentiation(a, x, N)

Input: $a, x \in [1, N]$

```
r ← 1
while x > 0 do
    if x is odd then
        r ← r · a mod N
    x ← ⌊x/2⌋
    a ← a^2 mod N
Return r
```

Theorem 31.1. *On input* (a, x, N) *where* $a, x \in [1, N]$*, Algorithm 2 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.

Groups

Definition 32.1. A group G is a set of elements G with a binary operator $\oplus : G \times G \to G$ 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 32.1. $(\mathbb{Z}_N^*, \times)$ *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 33.1. 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 33.1 (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 34.1 (Fermat's little theorem)**.** $\forall a \in \mathbb{Z}_p^*, a^{p-1} \equiv 1 \bmod p$

N)

Corollary 34.2. $a^x \bmod N = a^{x \bmod \Phi(N)} \bmod N$. *Thus, given* $\Phi(N)$*, the operation* $a^x \bmod N$ *can be computed efficiently in* $\mathbb{Z}_N$ *for any* x*.*

Thus, we can efficiently compute $a^{2^{2^x}} \bmod N$.

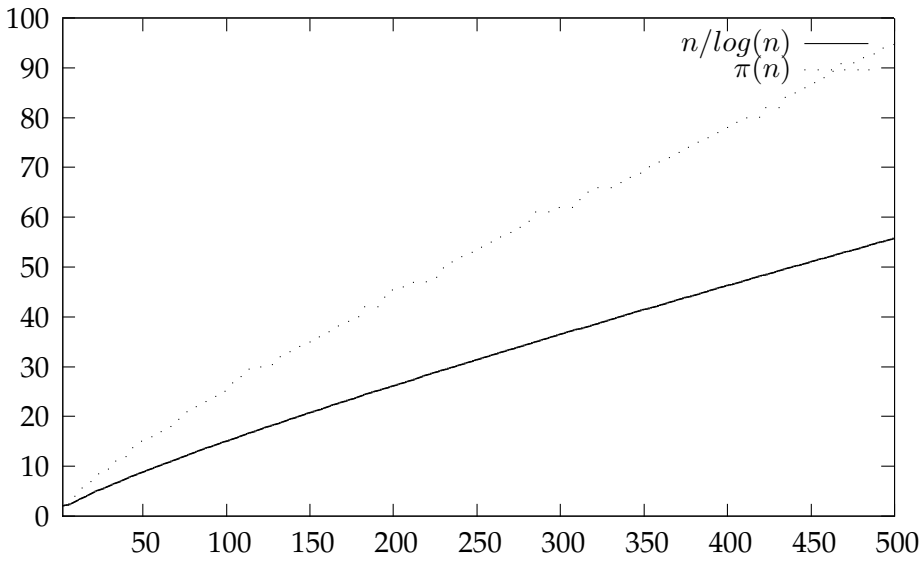
There are many primes

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 $\frac{N}{\log N}$ as N grows to infinity. Here, we will prove a much simpler theorem which only lower-bounds $\pi(x)$:

Theorem 35.1 (Chebyshev)**.** *For $x > 1$, $\pi(x) > \frac{x}{\log x}$*

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

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

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

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 35.1. 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. □

□

Miller-Rabin 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

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

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

Theorem 37.1. *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 3: Miller-Rabin Primality Test

```
Handle base case N = 2
for t times do
    Pick a random α ∈ Z_N
    if α ∉ 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 38.1. *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."*

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 4: SamplePrime(n)

```
repeat
    x ←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 35.1, 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.

2.6 Factoring-based Collection

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 this book:

Conjecture 38.1 (Factoring Assumption)**.** *For every non-uniform p.p.t. algorithm $\mathcal{A}$, there exists a negligible function ϵ such that*

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

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.

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

Theorem 39.1. *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. □

2.7 Discrete Logarithm-based Collection

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

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 39.1 (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 40.1 (Discrete Log Assumption)**.** *For every non-uniform p.p.t. algorithm $\mathcal{A}$, there exists a negligible function ϵ such that*

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

Theorem 40.1. *Let* $\boldsymbol{DL} = \{f_i : \mathcal{D}_i \to \mathcal{R}_i\}_{i \in I}$ *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 $\boldsymbol{DL}$ *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.

2.8 RSA collection

Another popular assumption is the RSA-assumption.

Conjecture 41.1 (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 41.1 (RSA Collection)**.** *Let **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 41.2. *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.

2.9 One-way Permutations

Definition 42.1. (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 42.1. *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

Algorithm 5: 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 $y^d \bmod N$

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 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—that is, whether the factoring assumption also implies the RSA-assumption.

2.10 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 43.1. (Trapdoor Permutations). A *collection of trapdoor permutations* if a family $\mathcal{F} = \{f_i : \mathcal{D}_i \to \mathcal{R}_i\}_{i \in \mathcal{I}}$ 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 44.1. *Let **RSA** be defined as per Theorem 41.1 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.

2.11 Levin's 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 there this assumption is true:

Theorem 44.2. *If there exists a one-way function, then the following polynomial-time computable function f_{Levin} is also a one-way function.*

Proof. We will construct function f_{Levin} 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_{Levin} should combine the computations in all efficient functions in such a way that inverting f_{Levin} allows us to invert all other functions. The naïve approach is to interpret the input y to f_{Levin} as a machine-input pair $\langle M, x\rangle$, and then let the output of f_{Levin} be $M(x)$. The problem with this approach is that f will not be computable

Algorithm 6: Function $f_{\text{Levin}}(y)$: A Universal OWF

1 Interpret y as $\langle \ell, M, x \rangle$ where $\ell \in \{0,1\}^{\log\log|y|}$ and $|M| = \ell$
2 Run M on input x for $|y|^3$ steps
3 **if** M *terminates* **then** Output $M(x)$ **else** Output $\perp$

in polynomial time, since M may or may not even terminate on input x. In words, this function interprets the first $\log\log|y|$ bits of the input y as a length parameter between $[0, \log|y|]$. The next ℓ bits are interpreted as a machine M, and the remaining bits are considered input x. We claim that this function is weakly one-way. Clearly f_{Levin} 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.

Without loss of generality, we can assume that g is computable in $O(n^2)$ time. This is because if g runs in $O(n^c)$ for some $c > 2$, then we can 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_{Levin} 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 $|M_g|$. Denote $\log|M_g| = \ell_g$. Thus, on a random input $y = \langle \ell, M_g, x \rangle$ of size $|y| = n$, the probability that $\ell = \ell_g$ is $2^{-\log\log n} = 1/\log n$ and probability that machine $M = M_g$ is $2^{-\log n} = 1/n$. In other words

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

Denote this event as event PickG.

We now lower bound the probability that $\mathcal{A}$ inverts a hard instance (i.e., one in which f has chosen M_g) as follows:

$$\begin{aligned}
\Pr[\textsf{Invert and PickG}] &= \Pr[\textsf{Invert}] - \Pr[\textsf{Invert and !PickG}] \\
&> \Pr[\textsf{Invert}] - \Pr[\textsf{!PickG}] \\
&> (1 - 1/n^3) - (1 - 1/n \log n) \\
&> 1/n \log n - 1/n^3
\end{aligned}$$

Applying Bayes rules to the previous inequality implies that $\Pr[\textsf{Invert} \mid \textsf{PickG}] > 1 - (n \log n)/n^3$ which contradicts the assumption that g is strongly one-way; therefore f_{Levin} 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 repeatedly 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 and 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. As our treatment is asymptotic, the actual formalization of this notion considers sequences—called *ensembles*—of probability distributions (or growing output length).

Definition 48.1 (Ensembles of Probability Distributions)**.** ...add

Definition 48.1. (Computational Indistinguishability). Let $\{X_n\}_{n\in N}$ and $\{Y_n\}_{n\in N}$ 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 $\{X_n\}_{n\in N}$ and $\{Y_n\}_{n\in N}$ are *computationally indistinguishable* (abbr. $\{X_n\}_{n\in N} \approx \{Y_n\}_{n\in N}$) if for all non-uniform PPT D (called the "distinguisher"), there exists a negligible function $\epsilon(n)$ such that $\forall n \in N$

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

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

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

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

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 indistinguishiable, then they remain indistinguishable also if one applies a p.p.t. computable operation to them.

Lemma 49.1. *Let $\{X_n\}_{n\in N}$,$\{Y_n\}_{n\in N}$ be ensembles of probability distributions where X_n, Y_n are probability distributions over $\{0,1\}^{l(n)}$ for some polynomial $l(\cdot)$, and let M be a p.p.t. machine. If $\{X_n\}_{n\in N} \approx \{Y_n\}_{n\in N}$, then $\{M(X_n)\}_{n\in N} \approx \{M(Y_n)\}_{n\in N}$.*

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

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

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

Transitivity - The Hybrid Lemma

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

Lemma 49.2 (The 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,

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

Let $g_i = \mathsf{Pr}\left[t \leftarrow X^i : D(t) = 1\right]$. 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 50.1 (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.)

3.2 Pseudo-randomness

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

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

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.

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 51.1. An ensemble of probability distributions $\{X_n\}_{n\in N}$ 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

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

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

Remark 51.1. 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 $t_0 \dots t_i$.

Theorem 51.1 (Completeness of Next-Bit Test)**.** *If a probability ensemble $\{X_n\}_{n\in N}$ passes the next-bit test then $\{X_n\}_{n\in N}$ is pseudo-random.*

Proof. Assume for the sake of contradiction that there exists a nonuniform 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.

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

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 $i \in [\ell(n)]$ such that D distinguishes between H_n^i and H_n^{i+1} with probability $\frac{1}{p(n)\ell(n)}$. That is,

$$\left|\mathsf{Pr}\left[t \leftarrow H_n^i \ : \ D(t)=1\right] - \mathsf{Pr}\left[t \leftarrow H_n^{i+1} \ : \ D(t)=1\right]\right| > \frac{1}{p(n)\ell(n)}$$

We assume without loss of generality that

$$\mathsf{Pr}\left[t \leftarrow H_n^{i+1} \ : \ D(t)=1\right] - \mathsf{Pr}\left[t \leftarrow H_n^i \ : \ D(t)=1\right] > \frac{1}{p(n)\ell(n)} \qquad (3.1)$$

Note that this is without loss of generality since we could always replace D with $D'(\cdot) = 1 - D(\cdot)$; it then must be the case that there exists infinitely many n for which either D or D' works.

Recall, that the only difference between H^{i+1} and H^i is that in H^{i+1} the $(i+1)$'th bit is x_{i+1}, whereas in H^i it is u_{i+1}. Thus, intuitively, D—given only the prefix $x_1 \dots x_i$—can tell apart x_{i+1} from a uniformly chosen bit. We show how to constuct a predictor A that uses such a D to predict x_{i+1}: A on input $(1^n, t_1 t_2 \dots t_i)$ picks $\ell(n) - i$ random bits $u_{i+1} \dots u_{\ell(n)} \leftarrow U^{\ell(n)-1}$, and lets $g \leftarrow D(t_1 \dots t_i u_{i+1} \dots u_{\ell(n)})$. If $g = 1$, it outputs u_{i+1}, otherwise it outputs $\bar{u}_{i+1} = 1 - u_{i+1}$. We show that

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

Towards this goal, consider the distribution $\tilde{H}(i)_n$ defined as follows:

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

Note that,

$$\begin{aligned}
&\mathsf{Pr}\left[t \leftarrow X_n;\ A(1^n, t_1 \dots t_i) = t_{i+1}\right] \\
&\quad = \frac{1}{2}\,\mathsf{Pr}\left[t \leftarrow H_n^{i+1} \ : \ D(t)=1\right] + \frac{1}{2}\,\mathsf{Pr}\left[t \leftarrow \tilde{H}_n^{i+1} \ : \ D(t) \neq 1\right] \\
&\quad = \frac{1}{2}\,\mathsf{Pr}\left[t \leftarrow H_n^{i+1} \ : \ D(t)=1\right] - \frac{1}{2}(1 - \mathsf{Pr}\left[t \leftarrow \tilde{H}_n^{i+1} \ : \ D(t)=1\right])
\end{aligned}$$

Also note that,

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

By rearranding the above equation and substituting, we conclude

$$\begin{aligned}
&\mathsf{Pr}\left[t \leftarrow X_n \ : \ A(1^n, t_1 \dots t_i) = t_{i+1}\right] \\
&\quad = \frac{1}{2}\mathsf{Pr}\left[t \leftarrow H_n^{i+1} \ : \ D(t) = 1\right] \\
&\qquad - \left(\mathsf{Pr}\left[t \leftarrow H_n^{i} \ : \ D(t) = 1\right] - \frac{1}{2}\mathsf{Pr}\left[t \leftarrow H_n^{i+1} \ : \ D(t) = 1\right] - \frac{1}{2}\right) \\
&\quad = \frac{1}{2} + \mathsf{Pr}\left[t \leftarrow H_n^{i+1} \ : \ D(t) = 1\right] - \mathsf{Pr}\left[t \leftarrow H_n^{i} \ : \ D(t) = 1\right] \\
&\quad > \frac{1}{2} + \frac{1}{p(n)\ell(n)}
\end{aligned}$$

where the last inequality follows from Equation 3.1. This concludes the proof the Theorem 51.1. □

3.3 Pseudo-random generators

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

Definition of a Pseudo-random Generators

Definition 53.1. 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 PPT.
2. (expansion): $|G(x)| > |x|$
3. (pseudo-randomness): The ensemble $\{x \leftarrow U_n : G(x)\}_{n \in N}$ is pseudo-random.

Constructions of Pseudorandom generators.

First attempt The first attempt to provide a general construction of pseudo-random generators was by Adi Shamir. The construction is as follows:

Definition 53.2 (PRG-Shamir)**.** Start with a one-way permutation f, then define G_{Shamir} as follows: $G(s) = f^n(s) \parallel f^{n-1}(s) \parallel \dots \parallel f(s) \parallel s$

(The $\parallel$ symbol stands for string concatentation.) The basic idea here to iterate a one-way function, then output, in reverse order, all the intermediary valuers. The insight behind the security of the scheme is that given some

prefix of the random sequence, computing the next block is equivalent to inverting the one-way function f.

This scheme leads to unpredictable numbers, but not necessarily pseudo-random *bits*; since some of the output bits of a one-way function may 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. If f is a permutation, this is true, but it need not hold for a general one-way function.

There is an extension of this technique, though, that does what we want.

Hard-core bits

Definition 54.1 (Hard-core predicate)**.** A predicate $h : \{0,1\}^* \rightarrow \{0,1\}$ 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 $\forall k \in \mathbb{N}$

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

In other words, a hard-core predicate for a function can not be computed given the result of the function, but can be computed given the function's input.

The least significant bit of the RSA one-way function is known to be hardcore (under the RSA assumption). Formally, given n,e, and $f_{RSA}(x) = x^e \bmod n$, there is no efficient algorithm to predict LSB(x) from $f_{RSA}(x)$.

Some other examples:

- 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.
- For the exponentiation function mod p, under the DL assumption, half$_{p-1}$ is a hardcore predicate.

Constructing the Generator

Now, how do we construct a PRNG from a one-way permutation? [Blum and Micali]

Proposition 54.1. *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 PRNG.*

Proof. Assume for contradiction that there exists a nonuniform p.p.t. adversary A and polynomial $p(n)$ so that for infinitely many n, there exists an i so that A predicts the i^{th} bit with probability $\frac{1}{p(n)}$. The first n bits of $G(s)$ are a permutation of a uniform distribution, and are therefore also distributed uniformly at random. So 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. Therefore, G is a PRNG. □

This construction only extends an n-bit seed to $n + 1$ bits. This, however is sufficient as we can cleverly iterate this procedure in order to generate a much longer sequence.

Theorem 55.1. *If there exists a PRG $f : \{0,1\}^n \to \{0,1\}^{n+1}$ which expands its seed by 1 bit, then there exists another PRG f' which expands its seed to $\ell(n)$ bits for any polynomial ℓ.*

Lemma 55.1. *Let $G : \{0,1\}^n \to \{0,1\}^{n+1}$ be a PRG. For some polynomial ℓ, define $G' : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ as follows:*

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

Then G' is a PRG.

Proof sketch. A hybrid argument. □

Corollary 55.1. *Let f be a OWP and h a hard core bit for f. Then*

$$g(x) = h(x) \parallel h(f(x)) \parallel h(f^{(2)}(x)) \parallel \dots \parallel h(f^{(n)}(x))$$

is a PRG.

Proof. Let $g'(x) = f(x) \parallel h(x)$. The function g' is a PRG. The result follows by the Lemma above. □

Note: The PRG construction above can be computed "on-line." If we remember x_i we can compute the continuation of the function. The construction also works for collections of OWP:

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

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

Concrete examples 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 $half_{p-1}(x) \parallel half_{p-1}(g^x \bmod p) \parallel half_{p-1}(g^{g^x} \bmod p) \parallel \ldots$
- **RSA** (Blum-Micali 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 PRG, 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.

3.4 *Hard-Core Bits from Any OWF

In the previous section, we showed that if f is a one way permutation and h is a hard-core bit for f, then the function

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

if a pseudo-random generator. One issue, however, is how to construct a hard-core bit for a given one-way permutation. We have illustrated examples for some of the well-known permutations. Here we will show that every one-way function (and therefore one-way permutation) can be transformed into another one-way function which has a hard-core bit.

Theorem

Let f be a OWF. Then $f'(X, r) = f(X), r$ (where $|X| = |r|$) is a OWF and $b(X, r) = \langle X, r \rangle_2 = \Sigma X_i r_i \bmod 2$ (inner product mod 2) is a hardcore predicate for f.

Here r essentially tells us which bits of x on which to take parity. Note that f' is a OWP if f is a OWP.

Proof. (by *reductio ad absurdum*) We show that if $\mathcal{A}$, given $f'(X,r)$ can compute $b(X,r)$ with probability significantly better than $1/2$, then there exists a p.p.t. adversary $\mathcal{B}$ that inverts f.

Our proof takes three steps. In the first step, we prove the theorem for an oversimplified case. In the next step, we take a less simplified case and finally we prove the theorem the general case.

In the **oversimplified** case, we assume $\mathcal{A}$ always computes b correctly. And so, we can construct a f' with an r such that the first bit is 1 and the other bits are 0. In such a case $\mathcal{A}$ would return the first bit of X. Similarly we can set the second bits of r to be 1 to obtain the second bit of X. Thus, we have $\mathcal{B}$ given by

$\mathcal{B}(y)$: Let $X_i = \mathcal{A}(y, e_i)$ where $e_i = 000\ldots1\ldots000$ where the 1 is on the i^{th} position.

-Output $X_1, X_2, \ldots, X_n$

This works, since $\langle X, e_i\rangle_2 = X_i$

Now, in the **less simplified** case, we assume that $\mathcal{A}$, when given random $y = f(X)$ and random r, computes $b(X,r)$ w.p. $\frac{3}{4} + \epsilon$, ($\epsilon = \frac{1}{poly(n)}$, n is the length of X).

Intuition: we want the attacker to compute b with a fixed X and a varying r so that given enough observations, X can be computed eventually. The trick is to find the set of good X, for which this will work.

As an attempt to find such X, let $S = \{X | \mathsf{Pr}\,[[]\, \mathcal{A}(f(X), r) = b(X,r)] > \frac{3}{4} + \frac{\epsilon}{2}\}$. It can be shown that $|S| > \epsilon/2$.

A simple attack with various e_i might not work here. More rerandomization is required. Idea: Use linearity of $\langle a, b\rangle$.

Useful relevant fact: $\langle a, b \oplus c\rangle = \langle a, b\rangle \oplus \langle a, c\rangle \; mod\; 2$

Proof.

$$\begin{aligned} \langle a, b \oplus c\rangle &= \Sigma a_i (b_i + c_i) \\ &= \Sigma a_i b_i + \Sigma a_i c_i \\ &= \langle a, b\rangle + \langle a, c\rangle \; mod\; 2 \end{aligned}$$

Attacker asks: $\langle X, r\rangle, \langle X, r + e_1\rangle$
and then XOR both to get $\langle X, e_1\rangle$ without ever asking for e_1.
And so, $\mathcal{B}$ inverts f as follows: $\mathcal{B}(y)$:

For i = 1 to n

1. Pick random r in $\{0,1\}^n$

2. Let $r' = e_i \oplus r$
3. Compute guess for X_i as $\mathcal{A}(y, r) \oplus \mathcal{A}(y, r')$
4. Repeat $poly(1/\epsilon)$ times and let X_i be majority of guesses.

Finally output $X_1, \ldots, X_n$.

If we assume e_1 and $r+e_1$ as independent, the proof works fine. However, they are not independent. The proof is still OK though, as can be seen using the union bound:

The proof works because:

- w.p. $\frac{1}{4} - \frac{\epsilon}{2}$ $\mathcal{A}(y, r) \neq b(X, r)$
- w.p. $\frac{1}{4} - \frac{\epsilon}{2}$ $\mathcal{A}(y, r') \neq b(X, r)$
- by union bound w.p. $\frac{1}{2}$ both answers of $\mathcal{A}$ are OK.
- Since $\langle y, r\rangle + \langle y, r'\rangle = \langle y, r \oplus r'\rangle = \langle y, e_i\rangle$, each guess is correct w.p. $\frac{1}{2} + \epsilon$
- Since samples are independent, using Chernoff Bound it can be shown that every bit is OK w.h.p.

Now, to the **general** case. Here, we assume that $\mathcal{A}$, given random $y = f(X)$, random r computes $b(X, r)$ w.p. $\frac{1}{2} + \epsilon$ $(\epsilon = \frac{1}{poly(n)})$

Let $S = \{X | \mathsf{Pr}[[] \mathcal{A}(f(X), r) = b(X, r)] > \frac{1}{2} + \frac{\epsilon}{2}\}$. It again follows that $|S| > \frac{\epsilon}{2}$.

Assume set access to oracle C that given $f(X)$ gives us samples

$$\begin{array}{l} \langle X, r_1\rangle, r_1 \\ \quad \vdots \qquad \text{(where } r_1, \ldots, r_n \text{ are independent and random)} \\ \langle X, r_n\rangle, r_n \end{array}$$

We now recall Homework 1, where given an algorithm that computes a correct bit value w.p. greater than $\frac{1}{2} + \epsilon$, we can run it multiple times and take the majority result, thereby computing the bit w.p. as close to 1 as desired.

From here on, the idea is to eliminate C from the constructed machine step by step, so that we don't need an oracle in the final machine $\mathcal{B}$.

Consider the following $\mathcal{B}(y)$:

For i = 1 to n

1. $C(y) \rightarrow (b_1, r_1), \ldots, (b_m, r_m)$
2. Let $r'_j = e_i \oplus r_j$
3. Compute $g_j = b_j \oplus \mathcal{A}(y, r')$
4. Let $X_i = majority(g_1, \ldots, g_m)$

Output $X_1, \ldots, X_m$

Each guess g_i is correct w.p. $\frac{1}{2} + \frac{\epsilon}{2} = \frac{1}{2} + \epsilon'$. As in HW1, by Chernoff bound, an x_i is wrong w.p. $\leq 2^{-\epsilon'^2 m}$ (was $2^{-4\epsilon^2 m}$ in the HW). If $m >> \frac{1}{\epsilon'^2}$, we are OK.

Now, we assume that C gives us samples $\langle X, r_1 \rangle, r_1; \ldots; \langle X, r_n \rangle, r_n$ which are random but only **pairwise independent**. Again, using results from HW1, by Chebyshev's theorem, each X_i is wrong w.p. $\leq \frac{1-4\epsilon'^2}{4m\epsilon'^2} \leq \frac{1}{m\epsilon'^2}$ (ignoring constants).

By union bound, any of the X_i is wrong w.p. $\leq \frac{n}{m\epsilon'^2} \leq \frac{1}{2}$, when $m \geq \frac{2n}{\epsilon'^2}$. Therefore, as long as we have polynomially many samples (precisely $\frac{2n}{\epsilon'^2}$ pairwise independent samples), we'd be done.

The question now is: How do we get pairwise independent samples? So, our initial attempt to remove C would be to pick $r_1, \ldots, r_m$ on random and guess $b_1, \ldots, b_m$ randomly. However, b_i would be correct only w.p. 2^{-m}.

A better attempt is to pick $log(m)$ samples $s_1, \ldots, s_{log(m)}$ and guessing $b'_1, \ldots, b'_{log(m)}$ randomly. 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. Mathematically

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

In HW1, we showed that these r_i are pairwise independent samples. 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 w.p. $1/2$. That is $\mathcal{B}(y)$ inverts w.p.

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

which contradicts the (strong) one-way-ness of f.

Yao proved that if OWF exists, then there exists OWF with hard core bits. But this construction is due to Goldreich and Levin[**?**] and by Charles Rackoff[**?**].

3.5 Pseudo-random Functions (PRFs)

Just like strings, functions can be sampled from the set of all functions. Thus, a natural question is whether one can construct a special family of functions which cannot be distinguished from the set of all functions. In this section, we do exactly that. In particular, we show how pseudo-random generators can be used to construct pseudo-random functions. Let us first define a random function.

Definition 60.1. A function $F : \{0,1\}^n \to \{0,1\}^n$ is a random function if it is constructed as follows: For each $x \in \{0,1\}^n$ pick a random $y \in \{0,1\}^n$, and let $F(x) = y$.

The description length of a random function is $n2^n$, and there are a total of 2^{n2^n} different random functions.

A random function can be simulated in by lazily generating the random values for each input the first time that input is queried.

Let RF_n denote the probability distribution resulting from picking a random function $F : \{0,1\}^n \to \{0,1\}^n$.

Definition 60.2.

$$\mathcal{F} = \left\{ f_s : \{0,1\}^{|s|} \to \{0,1\}^n \mid s \in \{0,1\}^* \right\}$$

is a family of PRFs if:

- [EASY TO COMPUTE] given $s \in \{0,1\}^n$ and $x \in \{0,1\}^n$, can efficiently compute $f_s(x)$.
- [PSEUDO-RANDOM] for all non-uniform PPT "oracle machines" D, there exists a negligible function $\epsilon(k)$ such that

$$\left| \Pr[s \leftarrow \{0,1\}^k : D^{f_s}(1^n) = 1] - \Pr[F \leftarrow RF_k : D^F(1^n) = 1] \right| \leq \epsilon(k)$$

Note that the number of PRFs $\{f_s \mid s \in \{0,1\}^k\}$ is much smaller than the number of random functions of the same length.

Construction of a PRF

Let

$$G(x) = G_0(x) \parallel G_1(x)$$

be a length doubling PRG such that $|G_i(x)| = |x|$. Let

$$f_s(b_1 b_2 \dots b_n) = G_{b_n}(G_{b_{n-1}}(\dots G_{b_2}(G_{b_1}(s)) \dots))$$

(That is, in an imperative form, $y := s$; *for* $i \in 1..n, y := G_{b_i}(y)$; *output* y.)

Theorem 61.1. *The set* $\{f_s \mid s \in \{0,1\}^n\}$ *is a family of PRFs.*

Proof sketch. Use a hybrid argument: assume for contradiction that there exists a PPT D and polynomial $p(n)$ such that D distinguishes f_s from a random function with probability better than $p(n)$.

Construct HF_i^n by picking the first i "layers" at random and then apply the construction. That is HF_i^n is a family of functions of the form $f(b_1 \dots b_n) = G_{b_n}(G_{b_{n-1}}(\dots G_{b_i}(r) \dots))$, where r is drawn from U_n, the uniform random distribution over $\{0,1\}^n$.

Note that:

$$HF_1^n = \{f_s \mid s \leftarrow \{0,1\}^n\}$$
$$HF_n^n = RF_n$$

Using the same proof as the hybrid lemma, there must exist an i such that D distinguishes between HF_i^n and HF_{i+1}^n with probability $\frac{1}{p(n)}$ for infinitely many n.

Note that HF_i^n can be efficiently sampled: first time an i-prefix is asked, pick a random string, and next time output the same.

Also, the only difference between HF_i^n and HF_{i+1}^n is that in HF_i^n the $i+1$ layer is computed as $G(U_n)$, whereas in HF_{i+1}^n it is U_{2n}.

Thus, D is able to distinguish polynomial many samples of $G(U_n)$ and U_{2n}.

Using an additional hybrid argument, we can contradict the assumption that G is a PRG, by defining hybrid H_i to take the first i bits from $G(U_n)$, and the rest of the bits from U_{2n}. □

Note: A question was asked, can the roles of s and b be swapped in the construction of the PRF, that is would

$$f_s(b) = G_{s_n}(G_{s_{n-1}}(\dots G_{s_2}(G_{s_1}(b)) \dots))$$

also result in a family of PRFs? No, it would not, since the PRGs G_0 and G_1 require the input to be a random value. The bit string s is a random value,

but the bit string $b = b_1 \dots b_n$ is picked by the adversary, and thus may not be random.

At this point in the course, we have shown how to define secrecy and how to construct tools such as one-way functions, pseudo-random generators, and pseudo-random functions. We will now use these concepts to build secure encryption schemes whose keys are smaller than the messages one can encrypt.

3.6 Secure Encryption Scheme

Definition 62.1 (Secure Encryption - Indistinguishability)**.** Let (Gen, Enc, Dec) be an encryption scheme over message space $\mathcal{M}$ and key space $\mathcal{K}$. The encryption scheme (Gen, Enc, Dec) is said to be *single-message secure* if $\forall$ non uniform p.p.t $\mathcal{A}$, there exists a negligible function $\epsilon(n)$ such that $\forall m_0, m_1 \in \mathcal{M}, |m_0| = |m_1|$ it holds that

$$|\Pr[k \leftarrow \mathsf{Gen}(1^n) : \mathcal{A}(\mathsf{Enc}_k(m_0)) = 1] - \Pr[k \leftarrow \mathsf{Gen}(1^n) : \mathcal{A}(\mathsf{Enc}_k(m_1)) = 1]| \leq \epsilon(n)$$

The above definition is based on the indistinguishability of the distribution of ciphertexts created by encrypting two different messages. The above definition does not, however, explicitly capture any *a priori* information that an adversary might have. Later in the course, we will see a definition which explicitly captures any *a priori* information that the adversary might have and in fact show that the indistinguishability definition is equivalent to it.

3.7 An Encryption Scheme with Short Keys

Recall that we constructed an encryption scheme using one time pads. However, we proved that to be able to prove perfect secrecy, the one time pad needed to be as large as the message. Can we construction encryption schemes using smaller keys which are secure under the new definitions we saw today? The idea is to use pseudorandomness instead of pure randomness. Since we know how to take a small seed and construct a long pseudorandom sequence, we can perform encryption with smaller keys.

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

Theorem 62.1. $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is single-message secure.*

Proof. Consider any two messages m_0 and m_1, and a distinguisher D. Consider the following distributions:

Algorithm 7: Encryption Scheme for n-bit messages

$\mathsf{Gen}(1^n)$: $k \leftarrow U_{n/2}$
$\mathsf{Enc}_k(m)$: Output $m \oplus G(k)$
$\mathsf{Dec}_k(c)$: Output $c \oplus G(k)$

- H_1 (real with m_0): $\{s \leftarrow Gen(1^n) : m_0 \oplus G(s)\}$.
- H_2 (OTP with m_0): $\{r \leftarrow U_n : m_0 \oplus r\}$.
- H_3 (OTP with m_1): $\{r \leftarrow U_n : m_1 \oplus r\}$.
- H_4 (real with m_1): $\{s \leftarrow Gen(1^n) : m_1 \oplus G(s)\}$.

By the perfect secrecy of one-time pad, there is no distinguisher D that can tell apart H_2 and H_3. Since $G(s)$ is pseudorandom, for every non-uniform p.p.t. distinguisher D, there is a negligible function ϵ such that for every n,

$$\left|\Pr[s \leftarrow U_{\frac{n}{2}} : D(G(s)) = 1] - \Pr[r \leftarrow U_n : D(r) = 1]\right| \leq \epsilon(n)$$

Therefore, since $f(x) = m \oplus x$ is a p.p.t. computable function, for any m, it holds that

$$\left|\Pr[s \leftarrow U_{\frac{n}{2}} : D(m \oplus G(s)) = 1] - \Pr[r \leftarrow U_n : D(m \oplus r) = 1]\right| \leq \epsilon(n)$$

Hence, there is no distinguisher which can tell apart H_1 and H_2 by more than a negligible function. Also, there is no distinguisher which can tell apart H_3 and H_4 by more than a negligible function. Therefore, by the hybrid lemma, there is no distinguisher which can tell apart H_1 from H_4 by more than a negligible function.

What we just showed is that for all non-uniform p.p.t distinguisher D, there is a negligible function ϵ, such that for all messages $m_1 \in \mathcal{M}, |m_1| = n$ and $m_2 \in \mathcal{M}, |m_2| = n$,

$$|\Pr[k \leftarrow \mathsf{Gen}(1^n) : D(\mathsf{Enc}_k(m_1)) = 1] - \Pr[k \leftarrow \mathsf{Gen}(1^n) : D(\mathsf{Enc}_k(m_2)) = 1]| \leq \epsilon(n)$$

Hence, $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is single-message secure. □

3.8 Many-message security

Unfortunately, the scheme described in the previous section is not secure if an eavesdropper receives encryptions of many messages. (Show this. In particular, consider the xor of two ciphertexts.) Thus, the previous definitions of

security are insufficient; although they guarantee security for one message, they guarantee nothing when two messages are sent. In this section, we will adapt the definitions to quantify over many messages.

Definition 64.1 (Secure Encryption, many message indistinguishability)**.** Let (Gen, Enc, Dec) be an encryption scheme over message space $\mathcal{M}$ and key space $\mathcal{K}$. Then (Gen, Enc, Dec) is said to be many-message secure if $\forall$ non uniform p.p.t. $\mathcal{A}$, $\forall$ polynomial $q(n)$, $\exists$ a negligible function $\epsilon(n)$ such that $\forall m_0, m_1, \ldots, m_{q(n)}, m'_0, m'_1, \ldots, m'_{q(n)}$ for which $|m_0| = \cdots = |m_{q(n)}| = |m'_0| = \cdots = |m'_{q(n)}|$ it holds that

$$\left| \begin{array}{l} \Pr[k \leftarrow \mathsf{Gen}(1^n) : \mathcal{A}(\mathsf{Enc}_k(m_0), \mathsf{Enc}_k(m_1) \ldots \mathsf{Enc}_k(m_{q(n)})) = 1] - \\ \quad \Pr[k \leftarrow \mathsf{Gen}(1^n) : \mathcal{A}(\mathsf{Enc}_k(m'_0), \mathsf{Enc}_k(m'_1), \ldots \mathsf{Enc}_k(m'_{q(n)})) = 1] \end{array} \right| \leq \epsilon(n)$$

The insight is to use a pseudo-random 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 8: 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 64.1. *(Gen, Enc, Dec) is a many-message secure encryption scheme.*

Proof. Consider some distinguisher D and two sequences of messages: $m_0, m_1 \ldots, m_{q(n)}$ and $m'_0, m'_1 \ldots, m'_{q(n)}$ respectively. We introduce the following sequence of distributions and argue that any two adjacent distributions are computationally indistinguishable.

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

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

  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 one-time pad encryption 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 one-time pad encryption 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 execution with $m'_0, m'_1 \ldots, m'_{q(n)}$

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

D can distinguish H_1 and $H2$ with at most negligible probability, otherwise it contradicts the pseudo-randomness properties of $\{f_s\}$. 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 with negligible probability, therefore D can distinguish H_2 and H_3 only with negligible probability. The same argument applies for H_4 and H_5.

H_3 and H_4 are identical because of the properties of the one time pad.

By the polyjump lemma, D can distinguish between H_1 and H_6 with at most negligible probability. □

In what follows we shall explore stronger attack scenarios. This far we had Eve listening to the communication channel between Alice and Bob – suppose Eve has *greater powers*.

3.9 Stronger Attack Models

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 66.1 (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. PPT A:

$$\left\{\mathsf{IND}_0^{O_1,O_2}(\pi, A, n)\right\}_{n\in N} \approx \left\{\mathsf{IND}_1^{O_1,O_2}(\pi, A, n)\right\}_{n\in N}$$

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.

3.10 CPA/CCA1 Secure Encryption Scheme

We will now show that the encryption scheme presented in construction 8 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 67.1. *π in construction 8 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. □

3.11 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 68.0. *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 PPT 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. □

3.12 Non-Malleability

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 69.1 (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 PPT $\mathcal{A}$, and for every non-uniform PPT $\mathcal{D}$, there exists a negligible ϵ such that for all $m_0, m_1 \in \{0,1\}^n$,

$$\mathsf{Pr}\left[\mathcal{D}(NM_0(\Pi, \mathcal{A}, n)) = 1\right] - \mathsf{Pr}\left[\mathcal{D}(NM_1(\Pi, \mathcal{A}, n)) = 1\right] \leq \epsilon(n)$$

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.

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 doesn't help the attacker encrypt an element of $R(x)$. Formally:

Definition 70.1 (Relation Based Non-Malleability)**.** We say that an encryption scheme (Gen, Enc, Dec) is relation based non-malleable if for every PPT adversary $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$ such that for all PPT-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 70.1. *(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

$$\begin{array}{rcl} |\Pr[NM(\mathcal{A}(z), m) \in R(m)] & - & \Pr[k \leftarrow Gen(1^n); c \leftarrow \mathcal{S}(1^n, z); m' = Dec_k(c) : m' \in R(m)]| \\ = |\Pr[\mathcal{D}(NM(\mathcal{A}(z), m))] & - & \Pr[\mathcal{D}(NM(\mathcal{A}(z), m'))]| \leq \epsilon \end{array}$$

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. □

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 CCA2 attacks, then the two definitions coincide.

Theorem 71.1. *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. □

3.13 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 72.1 (Public Key Encryption Scheme)**.** A triple $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a public key encryption scheme over a message space $\mathcal{M}$ 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 $m \in \mathcal{M}$ produces a cyphertext 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 $m \in \mathcal{M} \cup \perp$
4. For all $m \in \mathcal{M}$ and for all $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$,

$$\mathsf{Dec}_{S_k}(\mathsf{Enc}_{P_k}(m)) = m$$

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. Notice, however, that by giving the adversary the public key, there is no need to provide an encryption oracle. Thus, the oracle only plays a role in the CCA1/2 definitions.

Definition 72.2 (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:

$$\begin{aligned}
&\mathsf{Ind}_b(\Pi, \mathcal{A}, n)\\
&\quad (pk, sk) \leftarrow \mathsf{Gen}(1^n)\\
&\quad m_0,\ m_1,\ \mathsf{state} \leftarrow A^{O_1(sk)}(1^n, pk)\\
&\quad c \leftarrow \mathsf{Enc}_{pk}(m_b)\\
&\quad \text{Output } A^{O_2(k)}(c, \mathsf{state})
\end{aligned}$$

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.

With these 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 $\mathcal{M}$ 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.

In addition, it is a straightforward excercise to show that single-message security implies many-message security.

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 $Enc(m, i) = f_i(m)$, and $Dec(c, i, t) = f_i^{-1}(c)$. 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 41.1, 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 $\mathsf{Enc}(x, i) = \{r \leftarrow \{0,1\}^n : \langle f_i(r), b(r) \oplus m\rangle\}$ where b is a hardcore bit for f. This scheme is secure, because distinguishing encryptions of 0 and 1 is essentially the same as recognizing the hardcore bit of a one-way permutation, which we have argued is infeasible.

Theorem 73.1. *If* one-way trapdoor *permutations exist, then scheme 11 is a secure* public-key encryption *system for the message space* $\mathcal{M} = \{0, 1\}$.

Proof: To show that $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a secure *Public-Key Encryption* system, we must show that

$$\{(pk, sk) \leftarrow \mathsf{Gen}(1^n) : \mathsf{Enc}_{pk}(0)\} \approx \{(pk, sk) \leftarrow \mathsf{Gen}(1^n) : \mathsf{Enc}_{pk}(1)\}$$

Algorithm 10: 1-Bit Secure Public Key Encryption

Let $(f_i, f^{-1})_{i \in I}$ be a family of one-way trapdoor permutations and let b_i be the hard-core bit corresponding to f_i. Let Gen_T be the p.p.t that samples a trapdoor permutation index from I.
$\mathsf{Gen}(1^n) : (f_i, f_i^{-1}) \leftarrow \mathsf{Gen}_T(1^n)$. 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 $b_i(r) \oplus c_2$.

We define the following distributions.

$$
\begin{aligned}
\mathsf{Ind}_0 &= \{(pk, sk) \leftarrow Gen(1^n) : \mathsf{Enc}_{pk}(0)\} \\
&= \{(pk, sk) \leftarrow Gen(1^n) : r \leftarrow \{0,1\}^n : (f_{pk}(r), b(r) \oplus 0)\} \\
\mathsf{Ind}_1 &= \{(pk, sk) \leftarrow Gen(1^n); r \leftarrow \{0,1\}^n; : (f_{pk}(r), b(r) \oplus 1)\} \\
H_1 &= \{(pk, sk) \leftarrow Gen(1^n); r \leftarrow \{0,1\}^n : (f_{pk}(r), b(r))\} \\
H_2 &= \{(pk, sk) \leftarrow Gen(1^n); r \leftarrow \{0,1\}^n; r' \leftarrow \{0,1\} : (f_{pk}(r), r')\} \\
H_3 &= \{(pk, sk) \leftarrow Gen(1^n); r \leftarrow \{0,1\}^n; r' \leftarrow \{0,1\} : (f_{pk}(r), r' \oplus 1)\}
\end{aligned}
$$

The proof is by the hybrid argument. We first argue that $\mathsf{Ind}_0 = H_1$: This follows trivially because $b(r) \oplus 0 = b(r)$. Next, we claim that $H_1 \approx H_2$. This follows from the definition of hard-core bits. Now $H_2 = H_3$ because $\{r' \leftarrow \{0,1\} : r'\} = \{r' \leftarrow \{0,1\} : r' \oplus 1\}$ is the uniform distribution over $\{0,1\}$. Finally, by the definition of hard-core bits, we have $H_3 \approx \mathsf{Ind}_1$. Therefore, by the *Hybrid Lemma*, we have that $\mathsf{Ind}_0 \approx \mathsf{Ind}_1$. □

3.14 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 74.1 (Decisional Diffie-Hellman assumption (DDH))**.** *For all p.p.t $\mathcal{A}$, the following two distributions are computationally indistinguishable*

$$
\begin{aligned}
&\{p \leftarrow \Pi_n, y \leftarrow Gen_p, a, b \leftarrow \mathbb{Z}_q : p, y, y^a, y^b, y^{ab}\}_n \\
&\quad \approx \{p \leftarrow \Pi_n, y \leftarrow Gen_p, a, b, z \leftarrow \mathbb{Z}_q : p, y, y^a, y^b, y^z\}_n
\end{aligned}
$$

Notice that this assumption implies the discrete-log assumption 40.1 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. Again, with this assumption it is easy to see that the above key-exchange scheme is indistinguishably secure.

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

Algorithm 11: El-Gamal Secure Public Key Encryption

$\mathsf{Gen}(1^n)$: Pick a safe prime $p = 2q + 1$ of length n. Choose h a generator from QR_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 $c_2/c_1^a \bmod p$.

This is secure assuming the DDH assumption because h^{ab} is indistinguishable from a random element and hence $h^{ab} \cdot m$ is indistinguishable from a random element too.

Theorem 75.1. *Under the DDH Assumption, the El-Gamal encryption scheme is CPA-secure.*

Proof. □

Notice that instead of working in Z_p^*, the scheme only works with elements of the subgroup QR_p (even though the operations will be performed $\bmod p$). We further restrict our primes p to be of the form $2q + 1$ where q is prime too. Such primes are p are called *safe primes* and the corresponding q is called a *Sophie Germain prime*. This allows for an easy method to test whether an element h is a generator.

3.15 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.

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 a number defined as follows.

$$E[X] = \sum_{x \in \Omega} x Pr[X = x]$$

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 $|X_i| \leq 1$.

$$Pr\left[\left|\frac{\sum X_i}{n} - \mu\right| \geq \epsilon\right] \leq 2^{-\epsilon^2 n}$$