CS4230 / CS5430

Foundations of Modern Cryptography

Lecture Notes

These notes can always be improved. Any comments are very much appreciated. We would like to thank all the students of NUS CS4230/CS5430 who have contributed to scribing these notes.

Latest update, May 2026

Contents

National University of Singapore
CS4230/CS5430
Foundations of Modern Cryptography
AY 2025/26, Sem 2

Lecture 1 (Jan 15, 2026)
Lecture: Rahul Jain
Notes: Courtesy many

Security Definitions, Perfect Indistinguishability

For thousands of years, the idea of sending secret messages that no one but the recipient can read has been of interest in civilizations across the world. In 1587, Mary the queen of Scots (heir to the throne of England, who wanted to arrange the assassination of her cousin, queen Elisabeth I of England) sent a coded letter to Sir Anthony Babington using *substitution cipher*. Other well known examples of substitution ciphers include the Caesar cipher (≈2000 years ago), the Vigenère cipher (named after Blaise de Vigenère who described it in a book in 1586, though it was invented earlier by Bellaso), and the Enigma machine (≈90 years ago). Until relatively recently, however, most methods of such *secure communication* have been ad hoc and have been eventually broken, either with the development of new mathematical ideas or better technology.

Starting with the work of Claude Shannon in the 1940's [Sha49], the past few decades have seen a more systematic approach being adopted to construct cryptographic systems. Although conceptual and technological advances are always threats to security, this approach has placed cryptography on more secure foundations and tied it closely to various other parts of mathematics and computer science. This approach, which we shall call the *Cryptographic Method*, consists of four steps:

1. **Model** honest parties and adversaries
2. Rigorously **define** the desired security (and other) properties
3. **Construct** a system consistent with the modeling
4. **Prove** that the system has the desired properties

Although this is not how cryptosystems are always developed in practice, this is the approach to cryptography that we shall take in this course. We shall apply this method to a wide range of cryptographic tasks, far beyond the problem of secure communication that had been more-or-less the only cryptographic task that had been attempted for centuries. We shall leave out of our modeling several important details that need to be considered for cryptosystems to be implemented and deployed in practice, and we shall consider only the core theoretical constructions that may then be built upon.

An important principle to be followed in the first and second steps above is *Kerckhoff's Principle* (stated by Auguste Kerckhoffs in 1883):

> *A cryptosystem should be secure even if everything about the system, except the key, is public knowledge.*[1]

[1] The actual quote is "Il faut qu′il n′exige pas le secret, et qu′il puisse sans inconvénient tomber entre les mains de l′ennemi" loosely translated as "The system must not require secrecy and can be stolen by the enemy without causing trouble". According to Steve Bellovin the NSA version is "assume that the first copy of any device we make is shipped to the Kremlin".

That is, there is no security derived from obscurity. Even if the adversary knows the code of the algorithms that the honest parties run and the hardware they run it on, as long as the keys (and the internal states of the execution) are secure, the whole system should be secure.

Of the above steps, often the most difficult is the last. The task of *proving unconditionally* that a given cryptosystem cannot be broken is often beyond the reach of the mathematical techniques we know. Very often, this turns out to imply that $\mathsf{P} \neq \mathsf{NP}$, which is the biggest question in theoretical computer science that we are not even close to solving. So often, we prove the security of cryptosystems conditioned on some other problem being hard.

We might prove, for instance, that breaking a cryptosystem A would imply an efficient algorithm for a computational problem B. And if problem B (e.g., factoring) is believed to be a hard problem following decades or centuries of study, then this provides a strong basis for believing that cryptosystem A is secure. Further, if problem B happens to be of independent interest (as factoring is, for instance), then we end up with a win-win – either we have a secure cryptosystem, or we have an algorithm to solve an interesting problem. So, in the words of Silvio Micali, "Science wins either way".

1 Secure Communication

We shall start, as Shannon did, by applying the cryptographic method to the classical problem of secure communication. In this setting, there are two parties, let's say Alice and Bob. Alice has a channel of communication with Bob, but there happens to be an eavesdropper Eve who can see everthing that is sent on this channel. Alice has a message m (referred to as *plaintext*) that she wants to send to Bob in such a way that Eve does not learn m. For now, we will allow Alice, Bob, and Eve to be computationally unbounded – they can compute anything they want on what they know.

In this setting, it is clear that unless some additional correlation is present between Alice and Bob, this task will be impossible – otherwise, Eve looks the same to Alice as Bob, so anything Bob can learn from the channel, Eve can too. So we will allow Alice and Bob to confer beforehand to establish some common information that we will refer to as the key, and model the procedure to generate this key as an algorithm KeyGen. We will model the procedure Alice uses to *encrypt* her message using the key by the algorithm Enc, and the procedure Bob uses to *decrypt* her *ciphertext* and recover the message by the algorithm Dec. Below, the symbols $\mathcal{K}$, $\mathcal{M}$, and $\mathcal{C}$ refer to the spaces of keys, messages, and ciphertexts, respectively.

- $\mathsf{KeyGen}()$: Outputs a key $k \in \mathcal{K}$
- $\mathsf{Enc}(k, m)$: Given a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$, outputs a ciphertext c
- $\mathsf{Dec}(k, c)$: Given a key $k \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$, outputs a message $\hat{m} \in \mathcal{M}$

The adversary Eve here knows everything about the system except the key that is generated in a given specific execution. In each instance of Alice and Bob executing the above algorithms, Eve sees the ciphertext c and nothing else. This completes the first step of the method – we have modelled the system, distilling it down to the basic theoretical concerns. This combination of algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is referred to as an *encryption scheme*.

Next, we need to define rigorously the properties that we want the system to have. There are two: we want Bob to correctly decrypt the ciphertext to the message that Alice wants to send, and we want Eve to learn nothing about this message. The former is straightforward to state.

Definition 1.1 (Correctness)**.** An encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is said to be *correct* if, for any $m \in \mathcal{M}$,

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

The security, as is often the case, is less straightforward to define. We shall use a definition that turns out to be equivalent to Shannon's, but not the one in his original treatment. For any two messages $m_0, m_1 \in \mathcal{M}$, we define the following two hypothetical worlds:

World 0:

- $k \leftarrow \mathsf{KeyGen}()$
- $c_0 \leftarrow \mathsf{Enc}(k, m_0)$

World 1:

- $k \leftarrow \mathsf{KeyGen}()$
- $c_1 \leftarrow \mathsf{Enc}(k, m_1)$

For security, we ask that these two worlds be perfectly indistinguishable to anyone who only gets to see the ciphertext c. The idea here is that we want the ciphertext to look the same to Eve irrespective of the message being encrypted. If the ciphertext were deterministic, this would be easy – we could just require that $c_0 = c_1$. This is almost never the case since the key is usually chosen by KeyGen in a randomized manner. So we need to ask that the distributions of c_0 and c_1 are the same. Below, denote by C_0 and C_1 the random variables corresponding to the c_0 and c_1 in the two worlds.

Definition 1.2 (Perfect Indistinguishability)**.** An encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ satisfies perfect indistinguishability if, for any $m_0, m_1 \in \mathcal{M}$ and any $c \in \mathcal{C}$, we have:

$$\Pr[C_0 = c] = \Pr[C_1 = c]$$

This implies, in particular, that if Eve did not already know what message was being encrypted, then still cannot tell given the ciphertext. As we require this to hold for any two messages m_0 and m_1, it could even be that Eve has narrowed down what Alice is going to send to two messages, but still seeing the ciphertext does not tell her which of the two was encrypted.

2 The One-Time Pad

Next we construct an encryption scheme that satisfies the two properties we want of it. The scheme is very simple, as are the proofs that it has the required properties, though we shall later see that it has its shortcomings. Below, the symbol $\oplus$ represents bitwise XOR.

One-Time Pad

The message space, key space, and ciphertext space are all $\{0,1\}^n$ for some parameter $n \in \mathbb{N}$.

$\mathsf{KeyGen}()$**:**

- Sample and output a uniformly random $k \leftarrow \{0,1\}^n$

$\mathsf{Enc}(k, m)$**:**

- Output $k \oplus m$

$\mathsf{Dec}(k, c)$**:**

- Output $k \oplus c$

Theorem 2.1 (One-Time Pad)**.** *The one-time pad encryption scheme satisfies correctness and perfect indistinguishability.*

Proof. We will show the one-time has both properties.

Correctness. The proof of correctness is straightforward due to the properties of the XOR functions. For any $k \in \{0,1\}^n$,

$$\mathsf{Dec}(k, \mathsf{Enc}(k, m)) = k \oplus \mathsf{Enc}(k, m) = k \oplus (k \oplus m) = m$$

Perfect Indistinguishability. To show perfect indistinguishability, we need to show that for any m_0, m_1 and c in $\{0,1\}^n$, the following is true:

$$\Pr_{k \leftarrow \mathsf{KeyGen}()}[\mathsf{Enc}(k, m_0) = c] = \Pr_{k \leftarrow \mathsf{KeyGen}()}[\mathsf{Enc}(k, m_1) = c]$$

Fixing any m_0, m_1, and c, and we can expand the left-hand side as follows:

$$\Pr_{k \leftarrow \mathsf{KeyGen}()}[\mathsf{Enc}(k, m_0) = c] = \Pr_{k \leftarrow \{0,1\}^n}[k \oplus m_0 = c] = \Pr_{k \leftarrow \{0,1\}^n}[k = c \oplus m_0] = \frac{1}{2^n}$$

where the second equality follows from XOR'ing m_0 on both sides of the equation, and the last equality follows from the fact that k is being drawn uniformly at random. Further, the same arguments can be made with m_1 as well, leading to the probability being the same with m_1, thus proving perfect indistinguishability. □

2.1 Repeated Use of a One-Time Pad

As the name suggests, each key of the one-time pad is only usable once. If used to encrypt even two messages, it starts leaking information. Consider, for instance, the following two worlds where this is done.

World 0:

- $k \leftarrow \mathsf{KeyGen}()$
- $c_0 \leftarrow \mathsf{Enc}(k, m_0)$
- $c_0' \leftarrow \mathsf{Enc}(k, m_0')$

World 1:

- $k \leftarrow \mathsf{KeyGen}()$
- $c_1 \leftarrow \mathsf{Enc}(k, m_1)$
- $c_1' \leftarrow \mathsf{Enc}(k, m_1')$

In these worlds, the eavesdropper Eve sees (c_0, c_0') and (c_1, c_1'), respectively. In keeping with perfect indistinguishability, we would like these two worlds to be indistinguishable to Eve for any messages m_0, m_0', m_1, m_1'. However, consider the case where $m_0 = m_0'$, but $m_1 \neq m_1'$. Here, c_0 will always be equal to c_0', whereas c_1 will never be equal to c_1', and thus Eve can always distinguish between these worlds.

So the one-time pad loses security if a key is used more than once. This is inconvenient, as it would necessitate generating as many keys beforehand as there will be messages to encrypt. It turns out, however, that this is not a fault specific to the one-time pad, but rather of the security notion of perfect indistinguishability itself.

3 The Drawback of Perfect Indistinguishability

In the same paper where Shannon introduced the notion of perfect indistinguishability[2], he also shows that it has a significant drawback – in order for an encryption scheme to satisfy it, its keys have to be as large as the messages. This is not good because the messages we want to send might be gigabytes long, and we would ideally not have to generates single-use keys that are also as long and have to store them too.

Theorem 3.1 (Shannon's Theorem (restated) [Sha49])**.** *In any encryption scheme that satisfies both correctness and perfect indistinguishability,* $|\mathcal{K}| \geq |\mathcal{M}|$.

Proof. Suppose there is an encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with $|\mathcal{K}| < |\mathcal{M}|$ that satisfies both correctness and perfect indistingiushability. Fix some $m_0 \in \mathcal{M}$ and $k_0 \in \mathcal{K}$, and let $c = \mathsf{Enc}(k_0, m_0)$. Correctness implies that for any $k \in \mathcal{K}$, there is at most one message $m \in \mathcal{M}$ such that $\mathsf{Enc}(k, m) = c$. So the number of messages $m \in \mathcal{M}$ that are mapped to c by any k is at most $|\mathcal{K}|$. This implies that there is at least one message $m_1 \in \mathcal{M}$ for which there is no $k \in \mathcal{K}$ such that $\mathsf{Enc}(k, m_1) = c$. This implies that $\Pr_{k \leftarrow \mathcal{K}}[\mathsf{Enc}(k, m_1) = c] = 0$, but we know that $\Pr_{k \leftarrow \mathcal{K}}[\mathsf{Enc}(m_0) = c] \neq 0$. Thus, this contradicts perfect indistinguishability. So such an encryption scheme cannot exist. □

Starting in the next lecture, we will see how relaxing this definition by restricting the computational power of the adversary opens up the way to all of modern cryptography. A different relaxation is presented in the first problem set, which you are asked to show is not quite sufficient.

[2]Rather, he defined a different notion called "perfect secrecy" that is equivalent to it.

National University of Singapore
CS4230/CS5430
Foundations of Modern Cryptography
AY 2025/26, Sem 2

Lecture 2 (Jan 22, 2026)
Lecture: Rahul Jain
Notes: Courtesy many

Pseudorandom Generators, Computational Indistinguishability

In the last lecture, we saw that by Shannon's theorem [Sha48], any encryption scheme (e.g., one-time pad) satisfying perfect indistinguishability would have a key at least as long as the message, which is expensive to exchange/store/encapsulate. Moreover, the lengths of plaintexts, in reality, are flexible; hence, the key space would be hard to define and sample. This leads to the following key question.

Can we do encryption using much shorter keys than the length of plaintext?

A natural idea to solve these problems is to use instead a short random seed (or random tape) and an efficient deterministic algorithm to "stretch" this seed into a longer "somewhat" random longer string as the key. This deterministic algorithm is called a *Pseudorandom Generator* (PRG), which we define and study below.

More generally, we intend to relax our security definition by assuming a reasonable upper bound on the adversary's computational power. For instance, we aim to weaken the definition of indistinguishability to a more limited one, namely *computational indistinguishability*, assuming that the adversary is polynomial-time bounded. For completeness, some commonly used concepts and notation will be listed in the first section.

1 Notation and Conventions

For a random variable X and an algorithm A that takes a sample of X as input, we denote $\Pr[A(X, \cdots) = v]$ as a shorthand of $\Pr_{x \leftarrow X}[A(x, \cdots) = v]$ for a certain value v. For a non-empty set S, we use $x \leftarrow S$ to represent uniformly sampling an element from S. If X is a random variable, $x \leftarrow X$ means sampling of the random variable X. We let U_n denote the uniform distribution on $\{0,1\}^n$. The symbol $\leftarrow$ means value assignment if the right-hand side is not a random variable.

We will use $\mathcal{M}$, $\mathcal{K}$, $\mathcal{C}$ to represent the message space, the keyspace, and the ciphertext space in our encryption schemes, respectively.

1.1 Security Parameter

To more precisely quantify the level of security of a cryptosystem, we use a positive integer-valued *security parameter* (usually denoted by λ) that is fixed when the cryptosystem is set up and it parametrizes both cryptographic schemes as well as all involved parties. The honest parties in the system are required to run in time that is polynomial in λ, and the adversary (usually denoted Adv) is also assumed to have running time polynomial in λ.

Deterministic polynomial time algorithms are referred to as *Polynomial-Time* (PT) algorithms and randomized polynomial time algorithms are referred to as *Probabilistic-Polynomial-Time* (PPT) algorithms.

We will consider *non-uniform* PPT adversaries with polynomial size *advice* string. A non-uniform algorithm gets an advice string as input that depends on the length of the original input (and not on the specific original input itself). In our case, the advice string will depend on λ. From hereon we will assume non-uniform PPT adversaries and will not explicitly mention non-uniform.

1.2 Negligible Functions

Corresponding to our assumption that all parties in the system run in polynomial time, we usually equate the notion of "tiny probability of success" or "infeasibility" with success probabilities smaller than any inverse polynomial in the security parameter λ.

Definition 1.1 (Negligible Function)**.** A function $\nu : \mathbb{N} \to \mathbb{R}$ ($\mathbb{R}$ is the set of reals) is *negligible* if for every positive polynomial $p(\cdot)$ there is a λ_0 such that for all integer $\lambda \geq \lambda_0$, we have $\nu(\lambda) < \frac{1}{p(\lambda)}$.

For brevity, we will say "for all large enough λ" as shorthand for the existence of such number λ_0 and use $\mathrm{negl}(\cdot)$ to denote some negligible function.

Definition 1.2 (Noticeable Function)**.** A function $\nu : \mathbb{N} \to \mathbb{R}$ is *noticeable* if there exists a positive polynomial $p(\cdot)$ such that for infinitely many λ, we have $\nu(\lambda) \geq \frac{1}{p(\lambda)}$.

1.3 Distinguishing Advantage

Definition 1.3 (Advantage)**.** For random variables D_0, D_1 and algorithm A, we define A's advantage in distinguishing between D_0 and D_1 as:

$$\mathsf{adv}_A^{D_0,D_1} := \Pr_{x \leftarrow D_0}[A(x) = 1] - \Pr_{x \leftarrow D_1}[A(x) = 1]$$

Definition 1.4 (Computational Indistinguishability [GB01])**.** Two ensembles of random variables $\{X_\lambda, Y_\lambda\}$ are called computational indistinguishable if for all PPT A,

$$\mathsf{adv}_A^{X,Y} := \mathsf{adv}_A^{X,Y}(\lambda) := \Pr\left[A(X_\lambda, 1^\lambda) = 1\right] - \Pr\left[A(Y_\lambda, 1^\lambda) = 1\right] = \mathrm{negl}(\lambda)$$

2 Computational Indistinguishability for Encryption

As discussed earlier, we will hereon consider only PPT adversaries and relax perfect indistinguishability to indistinguishability by such adversaries. To perform this relaxation, we first redefine the perfect indistinguishability of the encryption scheme in terms of the advantage.

Definition 2.1 (Perfect Indistinguishability)**.** Given an encryption scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, for any $m \in \mathcal{M}$ we define random variable $C(m)$ as the output distribution of the following procedure:

Algorithm 1: $C(m)$

```
k ← KeyGen()
c ← Enc(k, m)
return c
```

We say the encryption scheme Π satisfies perfect indistinguishability if, for any $m_0, m_1 \in \mathcal{M}$, for any algorithm A:

$$\mathsf{adv}_A^{C(m_0),C(m_1)} = 0$$

We will relax this definition in two ways: first we will assume the algorithm A is PPT and second we allow A to have some negligible advantage in λ. In order to incorporate λ, we give it as input to all the algorithms in the system. We give this input in unary form, denoted 1^λ, to indicate that the algorithms are allowed to run in time polynomial in λ.

Given an encryption scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, for any $m \in \mathcal{M}$ and $\lambda \in \mathbb{N}$, we define the random variable $C(m, \lambda)$ as the output distribution of the following procedure:

Algorithm 2: $C(m, \lambda)$

$k \leftarrow \mathsf{KeyGen}(1^\lambda)$
$c \leftarrow \mathsf{Enc}(1^\lambda, k, m)$
return c

Definition 2.2 (Computational Indistinguishability of Encryption Scheme)**.** An encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ satisfies computational indistinguishability if, for any $m_0, m_1 \in \mathcal{M}$, for any PPT A, we have,

$$\mathsf{adv}_A^{C(m_0,\lambda),C(m_1,\lambda)} = \mathrm{negl}(\lambda)$$

where $C(m_i, \lambda)$ (for $i \in \{0, 1\}$) is defined in Algorithm 2.

3 Pseudorandom Generators

There are many ways to think about what a *truly random* object is. For our purposes, we will consider ideal randomness to correspond to a uniformly distributed variable over the sample space. That is, a random variable X over, say, $\{0,1\}^n$ is truly random if for any $x \in \{0,1\}^n$, $\Pr[X = x] = 1/2^n$.

Loosely speaking, we say a random variable is pseudorandom if it is computationally indistinguishable (or in short indistinguishable) from a uniform one in the same sample space. A pseudorandom variable can often be used in place of a uniform random variable with negligible degradation in security and better efficiency in some respects if we can generate it from a shorter uniform random variable. Such an algorithm that generates a pseudorandom variable over a larger space from a truly random variable over a smaller space is called a pseudorandom generator (PRG). We can define this property of pseudorandomness using the same scaffolding of indistinguishability we established earlier.

Definition 3.1 (PRG, Standard Definition)**.** A PRG is a PT G such that for every $\lambda \in \mathbb{N}$, G maps inputs from $\{0,1\}^\lambda$ to outputs in $\{0,1\}^{m(\lambda)}$, and the following two conditions are satisfied:

- **Expansion:** For all large enough $\lambda \in \mathbb{N}$: $m(\lambda) > \lambda$
- **Pseudorandomness:** For any PPT D (called a *distinguisher*):

$$\mathsf{adv}_D^{G(U_\lambda),U_{m(\lambda)}} = \mathrm{negl}(\lambda)$$

This uses the definition of true randomness and computational indistinguishability. There is another definition (sometimes more natural to use) that relaxes the Next-Bit-Unpredictability of truly random variables.

Definition 3.2 (PRG, Next-Bit Unpredictability)**.** A PRG is a PT $G : \{0,1\}^\lambda \to \{0,1\}^{m(\lambda)}$ such that the following two conditions are satisfied:

- **Expansion:** For all large enough $\lambda \in \mathbb{N}$: $m(\lambda) > \lambda$
- **Next-Bit Unpredictability:** For any PPT P (called a *predictor*) and for all $i \in \{1, \ldots, m(\lambda)\}$:

$$\Pr_{x \leftarrow \{0,1\}^\lambda, y \leftarrow G(x)} [P(y_1, \ldots, y_{i-1}) = y_i] = \frac{1}{2} + \text{negl}(\lambda)$$

These two definitions turn out to be equivalent – see the appendix for a proof. The proof involves a technique called the *hybrid argument* that will be useful in proofs throughout the rest of the class.

Theorem 3.1. *A PT G is pseudorandom if and only if it is computationally next-bit unpredictable.*

4 Encryption using PRG

Taking inspiration from the one-time pad an encryption scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with keys shorter than the messages can be constructed that satisfies computational indistinguishability.

Encryption using a PRG

Let $G : \{0,1\}^\lambda \to \{0,1\}^{m(\lambda)}$ be a PRG and $\mathcal{M} := \{0,1\}^{m(\lambda)}$, $\mathcal{K} := \{0,1\}^\lambda$, and $\mathcal{C} := \{0,1\}^{m(\lambda)}$.

$\mathsf{KeyGen}(1^\lambda)$:

- output $k_s \leftarrow \{0,1\}^\lambda$

$\mathsf{Enc}(1^\lambda, k_s, m)$:

- $k_r \leftarrow G(k_s)$
- output $k_r \oplus m$

$\mathsf{Dec}(1^\lambda, k_s, c)$:

- $k_r \leftarrow G(k_s)$
- output $k_r \oplus c$

Theorem 4.1. *The above encryption scheme satisfies correctness and computational indistinguishability.*

Proof. **Correctness.** For any $k_s \in \{0,1\}^\lambda$,

$$\mathsf{Dec}(1^\lambda, k_s, \mathsf{Enc}(1^\lambda, k_s, m)) = G(k_s) \oplus \mathsf{Enc}(1^\lambda, k_s, m) = G(k_s) \oplus (G(k_s) \oplus m) = m$$

Computational indistinguishability. We will use a *security reduction* to prove computational indistinguishability. First, we give a high-level overview of it. Similar to reductions in complexity theory, we efficiently transform a PPT adversary A that breaks security scheme Π to a PPT adversay A' that breaks another security scheme Π'. In this way, we ensure that security scheme Π is compromised only if scheme Π' is compromised. Notably, the complexity of the transformation needs to be polynomial-time. The security of most cryptographic schemes relies on some conjectured hardness (e.g., $NP \not\subseteq BPP$) that is used through such a security reduction.

The intuition of proving computational indistinguishability is that if this scheme (call it Π) uses a uniform pad instead of the pseudorandom pad $G(k_s)$, the scheme would be identical to one-time pad (referred to as Π') and the advantage of any adversary should be zero. Suppose an adversary D has a noticeable advantage on Π, then D can be used as a subroutine to construct a D' that distinguishes the output of G from a random string with noticeable advantage. Specifically, we assume for contradiction that G is a PRG but that the encryption scheme Π is not computationally indistinguishable. Formal argument is presented next.

Suppose there exits $m_0, m_1 \in \mathcal{M}$ and a PPT D and a polynomial $p(\cdot)$ such that there are infinite many λ with (below $C(m,\lambda)$ is as defined in Algorithm 2):

$$\begin{aligned}
\frac{1}{p(\lambda)} &\leq \Pr\left[D(C(m_0,\lambda)) = 1\right] - \Pr\left[D(C(m_1,\lambda)) = 1\right] \\
&= \Pr\left[D(G(U_\lambda) \oplus m_0) = 1\right] - \Pr\left[D(G(U_\lambda) \oplus m_1)) = 1\right] \\
&= \Pr\left[D(G(U_\lambda) \oplus m_0) = 1\right] - \Pr\left[D(U_{m(\lambda)} \oplus m_0) = 1\right]) \\
&\quad + \Pr\left[D(U_{m(\lambda)} \oplus m_1) = 1\right]) - \Pr\left[D(G(U_\lambda) \oplus m_1)) = 1\right] \quad \text{(from the secuity of one-time pad)}
\end{aligned}$$

By the *pigeonhole principle* we see at least one of the two statements hold:

- $W_0 : \Pr\left[D(G(U_\lambda) \oplus m_0) = 1\right] - \Pr\left[D(U_{m(\lambda)} \oplus m_0) = 1\right] \geq \frac{1}{2 \cdot p(\lambda)}$
- $W_1 : \Pr\left[D(U_{m(\lambda)} \oplus m_1) = 1\right] - \Pr\left[D(G(U_\lambda) \oplus m_1) = 1\right] \geq \frac{1}{2 \cdot p(\lambda)}$

Suppose it is W_0 that holds (the other case with W_1 is handled similarly). We use D as a subroutine to construct PPT D' that contradics the pseudorandomness of G as follows:

Algorithm 3: Distinguisher $D'(y)$

```
c ← m_0 ⊕ y                    // D' gets m_0 as advice depending on λ
b ← D(c)
return b
```

The advantage of D' is:

$$\begin{aligned}
\mathsf{adv}_{D'}^{G(U_\lambda), U_{m(\lambda)}} &= \Pr\left[D'(G(U_\lambda)) = 1\right] - \Pr\left[D'(U_{m(\lambda)}) = 1\right] \\
&= \Pr\left[D(G(U_\lambda) \oplus m_0) = 1\right] - \Pr\left[D(U_{m(\lambda)} \oplus m_0) = 1\right] \\
&\geq \frac{1}{2 \cdot p(\lambda)}
\end{aligned}$$

which contradicts our assumption that G is a PRG, and Theorem 4.1 follows. □

5 Constructions of PRGs

The existence of PRGs is unproven without computational assumptions because it is not hard to show that a PRG cannot exist unless $NP \not\subset BPP$ [Gol00], however there are plenty of candidates (e.g., [HILL99b], [PS98], [IN89]). Here we will give an efficient and simple one from the subset sum problem [IN89] as an example.

5.1 Subset Sum PRG

The subset sum problem of dimensions n and ℓ is: given n numbers $\vec{a} = (a_1, \ldots, a_n)$, each ℓ bits long, and a number T, find a subset $S \subseteq [n]$ such that $\sum_{i \in S} a_i = T \mod 2^\ell$. The problem can be viewed as inverting the following function:

$$f(\vec{a}, S) = (\vec{a}, \sum_{i \in S} a_i \mod 2^\ell)$$

Notice that the length of the input is $(\ell + 1)n$ bits, and that of the output is $\ell(n + 1)$ bits, which is larger if $\ell > n$. Inverting the above function in the worst case (that is, finding the subset S) is known to be NP-hard. The best algorithms we know to solve it for random inputs is still exponential time. Impagliazzo and Naor [IN89] showed that if f is hard to invert, then it is also a PRG.

5.2 Increasing the Stretch

For a PRG that stretches a λ-bit random seed to an $m(\lambda)$-bit, the function $(m(\lambda) - \lambda)$ is also called its *stretch*. In the encryption scheme we constructed, this stretch of the PRG determined how much smaller the key could be than the message. In general, PRG's with larger stretch seem better and harder to construct. It turns out, however, that in terms of existence of a PRG, the stretch does not matter.

Theorem 5.1. *If there exists a PRG G with a stretch of 1 bit (that is, $m(\lambda) = \lambda + 1$), then for any polynomial $p(\cdot)$, there exists a PRG G' with stretch $p(\lambda)$.*

The rough idea in showing this is described in the following procedure. We will see the complete proof in one of the later tutorials. This also involves the hybrid argument.

Algorithm 4: $G'(s)$

```
s_0 ← s
for i ← 1 to p(λ) do
    (σ_i, s_i) ← G(s_{i-1})          // σ_i ∈ {0,1} and |s_i| = λ
return (σ_1, ..., σ_{p(λ)}, s_{p(λ)})
```

Appendix

A.1 Notions of Indistinguishability

A.1.1 Statistical Indistinguishability

Definition A.1.1. (Statistical Distance) [BS15]. For two random variables $X, Y \in \{0,1\}^n$, we have

$$\sum_{\omega \in \{0,1\}^n} \Pr[X = \omega] = \sum_{\omega \in \{0,1\}^n} \Pr[Y = \omega] = 1$$

The statistical distance (also called the *total variation distance*) between X and Y is defined as:

$$\triangle(X, Y) := \frac{1}{2} \sum_{\omega \in \{0,1\}^n} |\Pr[X = \omega] - \Pr[Y = \omega]|$$

Another equivalent definition is:

$$\triangle(X, Y) := \sup_{S \subseteq \{0,1\}^n} |X(S) - Y(S)|$$

where $X(S)$ is shorthand for $\sum_{\omega \in S} \Pr[X = \omega]$.

Fact A.1.1. *For random variables X, Y and an algorithm A we have:* $\mathsf{adv}_A^{X,Y} \leq \triangle(X, Y)$.

Definition A.1.2 (Statistical Indistinguishability)**.** Two ensembles of random variables $\{X_\lambda, Y_\lambda\}$ are called *statistically indistinguishable* if $\triangle(X_\lambda, Y_\lambda) = \mathrm{negl}(\lambda)$.

A.1.1.1 Relation between Statistical Indistinguishability and Computational Indistinguishability

Clearly, if two random variable (ensembles) are statistically indistinguishable, they are also computationally indistinguishable. Intuition is from the second definition of statistical distance, which bounds the advantage of any adversary. However, the other direction does not hold by the following proposition.

Proposition A.1.2 ([Gol00])**.** *There exist a random variable ensemble $X_\lambda \in \{0,1\}^\lambda$ that is statistically far (noticeable statistical distance) from U_λ and yet X_λ and U_λ are computationally indistinguishable.*

A.2 Truly Random

Before we go into details of a PRG, we first give some intuitions on what random (or truly random) means in information theory. This can give some insights into the idea behind the definition of a PRG.

Definition A.2.1. (Truly random/uniformly random). For a random variable $X \in \{0,1\}^n$, we say X is truly random or uniformly random if $\forall x \in \{0,1\}^n : \Pr[X = x] = \frac{1}{2^n}$.

Here we list some expected properties of truly random strings for intuition purposes:

- **Next-Bit Unpredictability (NBU)**. It's easy to check that all bits in the uniformly random strings are independent; hence, no algorithm can predict the next bit with an expected advantage larger than zero, given the previous bits of the string.
- **Maximum Entropy**. The truly random X has maximum Shannon entropy and maximum min-entropy (recall $H(X) := \mathrm{E}_{x \leftarrow X}\left[\log \frac{1}{\Pr[X=x]}\right]$ and $H_{\min}(X) := \min\{\log \frac{1}{\Pr[X=x]}\}$).
- **Incompressibility**. Shannon's source coding theorem [Sha48] suggests the entropy $H(X)$ as the lower bound on the storage length of the random string X.

A.3 Pseudorandomness and Unpredictability

As discussed, the two definitions indistinguishability and next-bit unpredictability are shown to be equivalent. Before we prove the equivalence, we will briefly discuss why the pseudorandom generator definition did not rely on the maximum entropy or incompressibility of uniform randomness. Recall that the entropy/min-entropy of random variable X is the expected/minimum of $\log \frac{1}{P_X(\cdot)}$ respectively, where $P_X(\cdot)$ refers to probabilistic mass function of X. While the entropy/min-entropy can be used as an overall insight into randomness, its quantitative relation with uniformity is poorly studied and it does not indicate uniformity directly. Specifically, it's easy to construct a distribution that has high entropy/min-entropy but is statistically/computationally distinguishable from the uniform distribution. Similarly, incompressibility is harder to quantify compared to distinguishability and next-bit unpredictability.

Definition A.3.1 (Pseudorandom)**.** A random variable ensemble $X_\lambda \in \{0,1\}^{n(\lambda)}$ (n is a polynomial) is pseudorandom if for all PPT A,

$$\mathsf{adv}_A^{X_\lambda, U_{n(\lambda)}} = \mathrm{negl}(\lambda)$$

We will use X to represent an ensemble $\{X_\lambda\}$ and n to represent $n(\lambda)$ for brevity whenever it is clear from the context.

Definition A.3.2 (Computational Next-Bit-Unpredictability)**.** A random variable $X \in \{0,1\}^n$ is computationally next-bit unpredictable if for all $i \in [n]$ and every PPT $PRED$:

$$\Pr\left[PRED(X[1:i-1], 1^\lambda) = X_i\right] = \frac{1}{2} + \mathrm{negl}(\lambda)$$

where $X[1:i]$ denotes the first i bits of X.

Theorem A.3.1. *A random variable X is pseudorandom if and only if it is computationally next-bit unpredictable.*

Proof. **Psedorandomness ⇒ Computational Next-Bit Unpredictability (Only If)**

We assume for contradiction that there exists a PPT $PRED$ that can predict (with noticeable advantage) the ith bit of a pseudorandom variable $X \in \{0,1\}^n$ given the first $i-1$ bits with for

some number $1 \leq i \leq n$. Formally, there exists $i \in [n]$ and a polynomial $p(\cdot)$ such that for infinitely many λ:

$$\Pr\left[PRED(X[1:i-1], 1^\lambda) = X_i)\right] \geq \frac{1}{2} + \frac{1}{p(\lambda)}$$

Then $PRED$ can be efficiently transformed into a PPT distinguisher, denoted DST, for compromising pseudorandomness as follows.

Algorithm 5: Distinguisher $DST(x, 1^\lambda)$

```
b_i ← x[i]                                          // i is hardcoded
b* ← PRED(x[1 : i − 1], 1^λ)
if b_i = b* then
    return 1
else
    return 0
```

Clearly, this transformation is in polynomial time and

$$\Pr\left[DST(X, 1^\lambda) = 1\right] = \Pr\left[PRED(X[1:i-1], 1^\lambda) = X_i\right] \tag{1}$$

$$\geq \frac{1}{2} + \frac{1}{p(\lambda)} \tag{2}$$

whereas for $U \leftarrow \{0,1\}^n$

$$\Pr\left[DST(U, 1^\lambda) = 1\right] = \Pr\left[PRED(U[1:i-1], 1^\lambda) = U_i\right] \tag{3}$$

$$= \frac{1}{2} \tag{4}$$

Thus, $\Pr\left[DST(X, 1^\lambda) = 1\right] - \Pr\left[DST(U, 1^\lambda)\right] \geq \frac{1}{p(\lambda)}$, which contradicts the hypothesis that X is pseudorandom. The "only if" direction follows.

Pseudorandomness $\Leftarrow$ Computational Next-Bit Unpredictability (If)

Now we move to prove the other direction, which is less intuitive. We will use the hybrid argument to complete the proof.

Assume for contradiction that there is a computationally unpredictable $X \in \{0,1\}^n$ that is computationally distinguishable from the uniformly distributed $U \leftarrow \{0,1\}^n$. Formally for all $i \in [n]$, for all PPT $PRED$:

$$\Pr\left[PRED(X[1:i-1], 1^\lambda) = X_i\right] = \frac{1}{2} + \text{negl}(\lambda)$$

Also there exits a PPT DST and a polynomial $p(\cdot)$ such that for infinitely many λ:

$$\mathsf{adv}_{DST}^{U,X} = \Pr\left[DST(U, 1^\lambda) = 1\right] - \Pr\left[DST(X, 1^\lambda) = 1\right] \geq \frac{1}{p(\lambda)}$$

We construct $n+1$ hybrid distributions $D_0, \ldots, D_n$ in the way that $D_i = X[1:i]||U[i+1:n]$ as illustrated in Fig. 1 ($X[i:j]$ means the slides of X's samples from the ith bit to the jth bit if

$1 \leq i < j \leq n$, and $||$ denotes concatenation). Clearly $\mathsf{adv}_{DST}^{X,U} = \mathsf{adv}_{DST}^{D_n,D_0}$ and we have,

$$\begin{aligned}\frac{1}{p(\lambda)} \leq \mathsf{adv}_{DST}^{D_n,D_0} &= \Pr\left[DST(D_n, 1^\lambda) = 1\right] - \Pr\left[DST(D_0, 1^\lambda) = 1\right] \\ &= \sum_{i=1}^{n} \Pr\left[DST(D_i, 1^\lambda) = 1\right] - \Pr\left[DST(D_{i-1}, 1^\lambda) = 1\right] \\ &= \sum_{i=1}^{n} \mathsf{adv}_{DST}^{D_i,D_{i-1}}\end{aligned}$$

By the pigeonhole principle, there must exist a $i \in [n]$ such that

$$\mathsf{adv}_{DST}^{D_i,D_{i-1}} \geq \frac{1}{n \cdot p(\lambda)} \tag{5}$$

We define D'_i to be with the same value as D_i except that the ith bit is flipped (as shown in Fig. 2). We have the following claim.

Claim A.3.1.

$$\Pr\left[DST(D_{i-1}, 1^\lambda) = 1\right] = \frac{\Pr\left[DST(D_i, 1^\lambda) = 1\right] + \Pr\left[DST(D'_i, 1^\lambda) = 1\right]}{2}$$

Proof. Note that for any $x \in \{0,1\}^n$,

$$\Pr\left[D_{i-1} = x\right] = \frac{\Pr\left[D_i = x\right] + \Pr\left[D'_i = x\right]}{2}$$

Also,

$$\begin{aligned}\Pr\left[DST(D_{i-1}, 1^\lambda) = 1\right] &= \sum_{x \in \{0,1\}^n} \Pr\left[D_{i-1} = x\right] \cdot \Pr\left[DST(x, 1^\lambda) = 1\right] \\ \Pr\left[DST(D_i, 1^\lambda) = 1\right] &= \sum_{x \in \{0,1\}^n} \Pr\left[D_i = x\right] \cdot \Pr\left[DST(x, 1^\lambda) = 1\right] \\ \Pr\left[DST(D'_i, 1^\lambda) = 1\right] &= \sum_{x \in \{0,1\}^n} \Pr\left[D'_i = x\right] \cdot \Pr\left[DST(x, 1^\lambda) = 1\right]\end{aligned}$$

The claim now follows by combining the above. □

Now we are ready to construct the $PRED$ as follows:

Algorithm 6: Predictor $PRED(Y, 1^\lambda)$

```
i ← |Y| + 1
U ← {0,1}^{n−i}
C ← {0,1}
if DST(Y‖C‖U, 1^λ) = 1 then
    return C
else
    return C̄
```

Lemma A.3.2. *If DST can distinguish X from U with advantage at least $\frac{1}{p(\lambda)}$, then PRED in Algorithm 6 is a predictor of distribution X that predict the ith bit of X given the first $i-1$ bits with probability at least than $\frac{1}{2}+\frac{1}{n\cdot p(\lambda)}$.*

Proof of Lemma A.3.2. Let $Y := X[1, i-1]$. We have,

$$
\begin{aligned}
&\Pr\left[PRED(Y, 1^\lambda) = X_i\right] \\
&= \Pr\left[DST(Y||C||U, 1^\lambda) = 1 \wedge C = X_i\right] + \Pr\left[DST(Y||C||U, 1^\lambda) = 0 \wedge C = \overline{X_i}\right] \\
&= \Pr\left[DST(Y||C||U, 1^\lambda) = 1 | C = X_i\right] \cdot \Pr\left[C = X_i\right] \\
&\quad + \Pr\left[DST(Y||C||U, 1^\lambda) = 0 | C = \overline{X_i}\right] \cdot \Pr\left[C \neq X_i\right] \\
&= \frac{1}{2}\left(\Pr\left[DST(Y||C||U, 1^\lambda) = 1 | C = X_i\right] + \Pr\left[DST(Y||C||U, 1^\lambda) = 0 | C = \overline{X_i}\right]\right) && (C \leftarrow \{0,1\}) \\
&= \frac{1}{2}\left(\Pr\left[DST(D_i, 1^\lambda) = 1\right] + 1 - \Pr\left[DST(D'_i, 1^\lambda) = 1\right]\right) \\
&= \frac{1}{2} + \Pr\left[DST(D_i, 1^\lambda) = 1\right] - \Pr\left[DST(D_{i-1}, 1^\lambda) = 1\right] && \text{(from Claim A.3.1)} \\
&\geq \frac{1}{2} + \frac{1}{n \cdot p(\lambda)} && \text{(Eq. (5))}
\end{aligned}
$$

Hence Lemma A.3.2 follows. □

Lemma A.3.2 contradicts our assumption and hence the "if" direction follows.

□

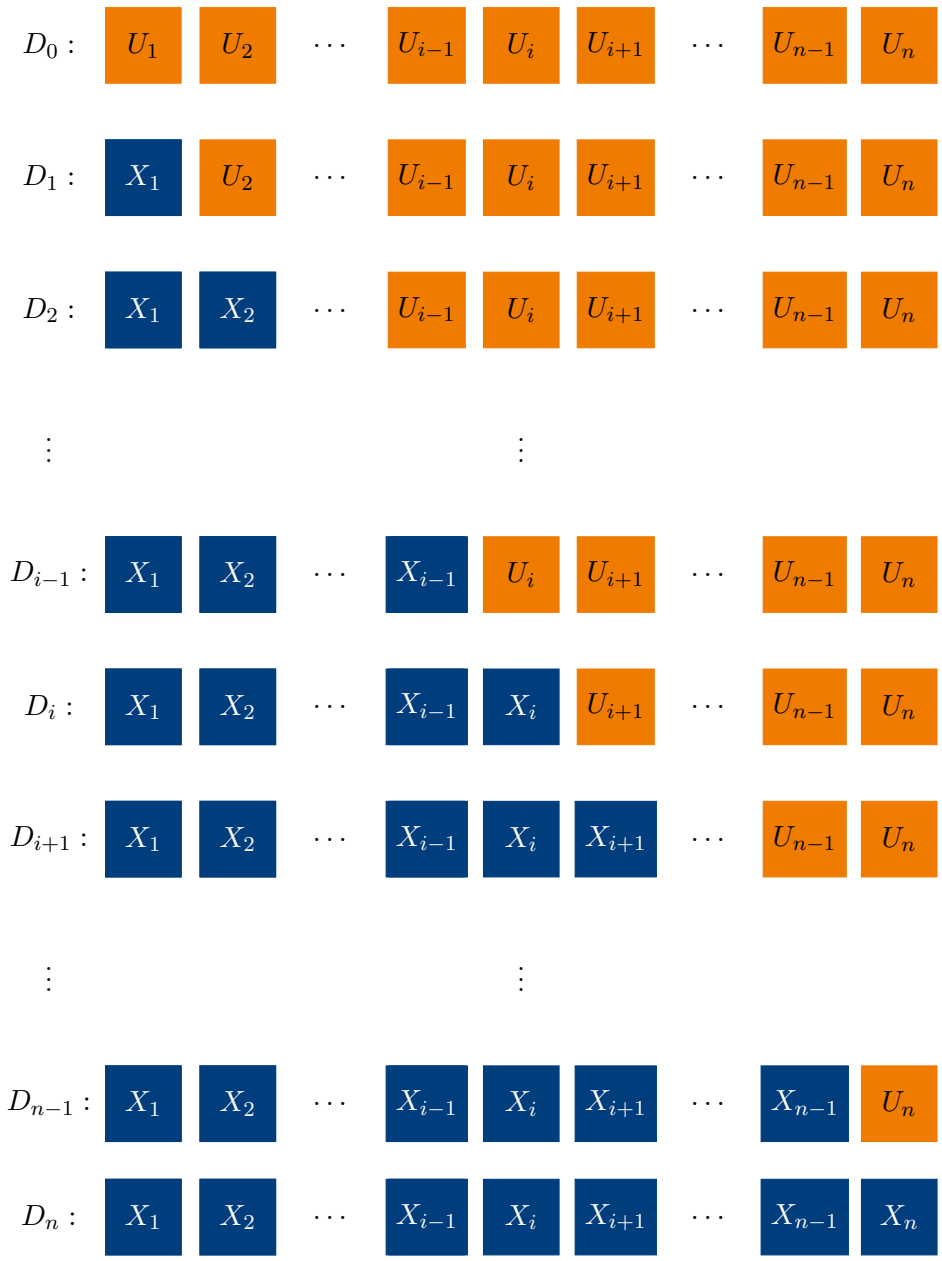


Figure 1: Hybrid Distributions

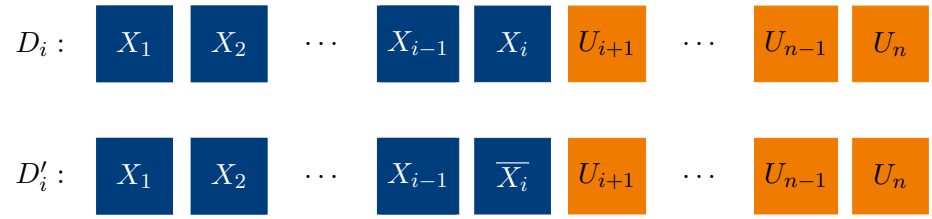


Figure 2: D'_i and D_i

Pseudorandom Functions, Chosen-Plaintext Attacks

In the last lecture, we learned that PRGs are polytime computable deterministic functions that (i) accept random binary strings of length λ as input and (ii) generate longer, *seemingly* random, binary strings of length $m(\lambda)$ as output such that any PPT Adv cannot distinguish between the output of the PRG and a uniformly random output with a noticeable advantage greater than $\frac{1}{2}$.

We also saw how PRGs can be used to construct an encryption scheme with a key-length shorter than the message length such that the encryption scheme satisfies computational-indistinguishability.

The next key question we ask is:

> *How about the security if Adv is provided with polynomially (for any polynomial) many ciphertexts for plaintexts of its choice?*

In this case a PRG does not suffice since its output length is a fixed polynomial in λ. We will need a stronger cryptographic primitive called the *Pseudorandom Function* (PRF).

Chosen-Plaintext Attacks

A chosen-plaintext attack (CPA) is an attack on an encryption scheme in which Adv has the ability to choose plaintexts and to view their corresponding ciphertexts. The goal of the attack is to gain information about the encryption scheme that weakens/compromises its security.

We consider the following security-game between a PPT Challenger (Cha) and PPT Adv for security of an encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ against CPA.

IND-CPA(λ):

1. Cha chooses a random bit $b \leftarrow \{0,1\}$ and samples $k \leftarrow \mathsf{KeyGen}(1^\lambda)$.
2. For $i \in [t]$ for some $t = \mathsf{poly}(\lambda)$:
   - Adv chooses $\hat{m}_i \in \mathcal{M}$ and sends it to Cha.
   - Cha sends corresponding encoding $\mathsf{Enc}(1^\lambda, k, \hat{m}_i)$ to Adv.
3. Adv chooses $m_0, m_1 \in \mathcal{M}$ (different from $\{\hat{m}_i \mid i \in [t]\}$) and sends them to Cha.
4. Cha sends $\mathsf{Enc}(1^\lambda, k, m_b)$ to Adv.
5. Adv outputs a guess bit b'.

6. Adv wins if $b' = b$.

Definition 0.3 (CPA-security)**.** An encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is IND-CPA-secure if for every PPT Adv:

$$\Pr\left[\text{Adv wins IND-CPA}(\lambda)\right] = \frac{1}{2} + \text{negl}(\lambda)$$

1 Pseudorandom Functions

Pseudorandom Functions (PRFs) are mathematical objects that *behave* like PRGs, with the added capability of being able to generate **just** the i^{th} block, for any arbitrary i. That is, PRFs are *locally computable* or *indexable*.

1.1 PRF Families

Let F be a function family:

$$F = \{F_\lambda\}_{\lambda \in \mathbb{N}}, \; F_\lambda = \left\{ f_k : \{0,1\}^{n(\lambda)} \to \{0,1\}^{m(\lambda)} \mid k \in \mathcal{K} \right\}$$

where n, m are polynomials, sampling $k \leftarrow \mathcal{K}$ is efficient and computation of f_k is efficient. Let F_λ^{all} as the set of all functions $f : \{0,1\}^{n(\lambda)} \to \{0,1\}^{m(\lambda)}$. Since F_λ^{all} contains all functions from $\{0,1\}^{n(\lambda)}$ to $\{0,1\}^{m(\lambda)}$, randomly sampling function from F_λ^{all} will give us a truly random function $f : \{0,1\}^{n(\lambda)} \to \{0,1\}^{m(\lambda)}$.

A PRF is considered to be secure if given output(s) from either the truly random function or a PRF, no PPT Adv can distinguish whether the output was produced by the truly random function or the PRF. This leads us to the following security-game for PRFs between PPT Cha and PPT Adv.

FN-IND(λ):

1. Cha chooses a random bit $b \leftarrow \{0,1\}$.
   (a) If $b = 0$, Cha samples a random function $f \leftarrow F_\lambda^{all}$.
   (b) If $b = 1$, Cha samples a random function $f \leftarrow F_\lambda$ from the PRF family.
2. For $i \in [t]$ for some $t = \mathsf{poly}(\lambda)$:
   - Adv sends $x_i \in \{0,1\}^{n(\lambda)}$ to Cha.
   - Cha sends $f(x_i)$ to Adv.
3. Adv outputs a guess bit b'.
4. Adv wins if $b' = b$.

Definition 1.1 (PRF)**.** The family F is a PRF if for every PPT Adv:

$$\Pr\left[\text{Adv wins FN-IND}(\lambda)\right] = \frac{1}{2} + \text{negl}(\lambda)$$

1.2 Encryption using PRFs

Designing encryption schemes using PRFs is similar to encryption schemes that are built using PRGs. To begin, we use the same three functions from before. The first step is to generate a (shared) secret key k. From here, encryption is (i) picking an arbitrary index i, (ii) computing the *local* one-time pad $b_i = f_k(i)$, and finally (iii) encrypting the plaintext message to get the ciphertext $c_i = b_i \oplus m$. Here, we send the pair (i, c_i) across the communication channel.

On the receiver's end, decryption is likewise straightforward. As the secret key k is shared, the receiver knows to use the PRF f_k. Since the index i was sent across the channel, the receiver can locally derive the one-time pad used $b_i = f_k(i)$. Then, retrieving the plaintext message can be done with $m = b_i \oplus c_i$.

Algorithm 7: Encryption scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$

Function `KeyGen`(1^λ)**:**
  sample k
  return k

Function `Enc`$(1^\lambda, k, m)$**:**
  pick $i \leftarrow \{0,1\}^\lambda$
  return $(i, f_k(i) \oplus m)$

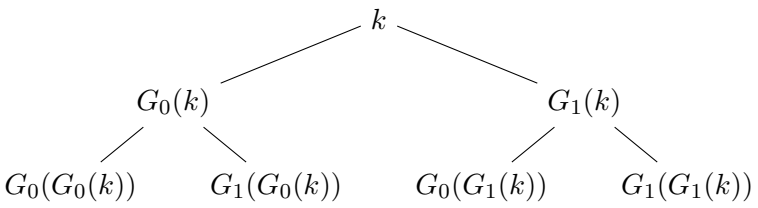
Function `Dec`$(1^\lambda, k, (i, c))$**:**
  return $(f_k(i) \oplus c)$

Claim 1.1. *The encryption scheme constructed using the PRF family is CPA secure.*

Proof Idea: The aim is to give a security reduction such that if there exists a PPT A that wins the IND-CPA game for the encryption scheme with noticeable advantage over $\frac{1}{2}$, then there exists a PPT A' that wins the FN-IND game for F with noticeable advantage over $\frac{1}{2}$ which contradicts that F is a PRF.

1.3 Construction of PRFs

Goldreich, Goldwasser, and Micali [GGM86] showed how a PRF family can be constructed from any PRG. The idea is to use a length-doubling PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ on a key k, and then apply G again on the left and the right halves of $G(k)$ and continue this process recursively. In each step of this recursive process, the length of the pseudorandom output doubles as from 1 block of length λ, we get 2 blocks each of length λ. These blocks are finally used as the outputs of the PRF family on different inputs. For example, for PRF $f_k : \{0,1\}^2 \to \{0,1\}^\lambda$, $f_k(00) = G_0(G_0(k))$, $f_k(10) = G_0(G_1(k))$ and so on.



Claim 1.2. *Let G be a length-doubling PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$. Define G_0 and G_1 as the left and right halves of the G such that $G(x) = G_0(x)||G_1(x)$ where $|G_0(x)| = |G_1(x)| = \lambda$. For any $k \in \{0,1\}^\lambda$, we define $f_k : \{0,1\}^n \to \{0,1\}^\lambda$ as*

$$f_k(x_1 \dots x_n) = G_{x_n}(G_{x_{n-1}}(...(G_{x_2}(G_{x_1}(k))...)).$$

Then

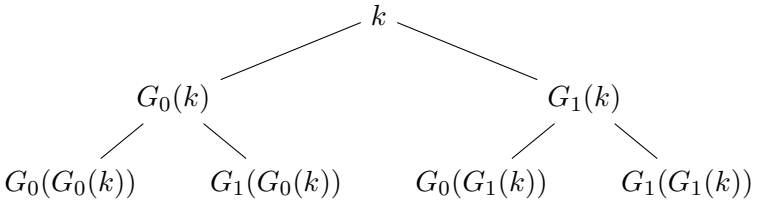
$$F = \{F_\lambda\}_{\lambda \in \mathbb{N}}, \; F_\lambda = \left\{ f_k : \{0,1\}^n \to \{0,1\}^\lambda \mid k \in \{0,1\}^\lambda \right\}$$

is a PRF family.

Proof Sketch. We consider the special case $n = 2$. The general proof can be done along similar lines.

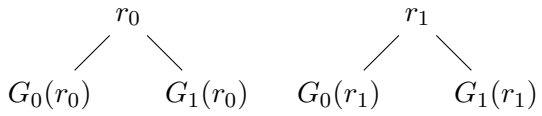
The aim is to reduce the security of PRG G to the security of PRF F using a hybrid argument so that if a PPT A can break the security of F, we can use it to create a PPT A' that is able to break the security of G.

Hybrid-1:



Since the seed k of the PRG is random, the output $G_0(k)$ and $G_1(k)$ is also random to a PPT adversary. So we create a new hybrid in which we replace $G_0(k)$ and $G_0(k)$ with random stings r_0 and r_1.

Hybrid-2:



Using the same argument, we can once again replace the outputs $G_0(r_0), G_1(r_0), G_0(r_1)$ and $G_1(r_1)$ with random strings r_{00}, r_{01}, r_{10} and r_{11}.

Hybrid-3:

$$r_{00} \qquad r_{01} \qquad r_{10} \qquad r_{11}$$

The idea behind the proof is that if a PPT adversary A can distinguish between Hybrid-1 and Hybrid-3 with noticeable advantage over $\frac{1}{2}$ then it can distinguish between either Hybrid-1 and

Hybrid-2 or Hybrid-2 and Hybrid-3 with noticeable advantage over $\frac{1}{2}$ which would allow us to break the security G. $\square$

National University of Singapore
CS4230/CS5430
Foundations of Modern Cryptography
AY 2025/26, Sem 2

Lecture 4 (February 5, 2026)
Lecture: Rahul Jain
Notes: Courtesy many

One-Way Functions, Hardcore Bits

In the previous lectures, we studied PRGs and PRFs, which are both fundamental cryptographic primitives allowing us to solve problems like private key encryption. We also saw how the existence of PRGs implies the existence of PRFs.

In this lecture, we will introduce another primitive called *One-Way Function* (OWF) which is seemingly weaker than those we've seen before. However, it turns out that these functions are enough to construct PRGs and therefore PRFs.

1 OWF

Intuitively our notion of an OWF f is that it should be *easy* to compute $f(x)$ from x, but *hard* to obtain x only from $f(x)$.

Formally, consider a function family (or just function for brevity)

$$F = \{f_\lambda : \{0,1\}^\lambda \to \{0,1\}^{m(\lambda)}\}_{\lambda \in \mathbb{N}}$$

where m is a polynomial.

Definition 1.1 (OWF)**.** We say F is an OWF if it satisfies the following:

- **Efficiency:** There exists a deterministic polynomial time (PT) algorithm which on input $(1^\lambda, x)$ computes $f_\lambda(x)$.
- **Hardness of Inversion:** For any PPT A,

$$\Pr\left[f_\lambda(A(1^\lambda, f_\lambda(U_\lambda))) = f_\lambda(U_\lambda)\right] = \text{negl}(\lambda)$$

Note that we do not require A to necessarily produce x as we are okay with any pre-image of $f(x)$. To see why this is more reasonable, suppose we instead asked A to reproduce x directly. Then, the following function would be OWF by this definition: $f_\lambda(x) = 0^{m(\lambda)}$. Since A will only get the all-zero string, there is no way it can guess the value of x from that.

Also, note that we do not require $m(\lambda) > \lambda$, but only polynomial in λ. For example, $m(\lambda)$ could be equal to $\sqrt{\lambda}$.

Definition 1.2 (One-Way Permutation (OWP))**.** F is an OWP if it satisfies the following:

- F is an OWF with $m(\lambda) = \lambda$.
- f_λ is a bijection for every λ.

2 Hardcore Bits

Given an OWF, is it possible for f to give away a certain bit of x, say x_1, and still be hard to invert? Yes, simply consider a function family such that $f_\lambda = x_1 || g_{\lambda-1}(x_2 x_3 \dots x_\lambda)$, which clearly gives away the first bit but is hard to invert as long as g is an OWF.

In this section, we look at *Hard-Core* bits, which can be intuitively thought of the *source* where the hardness of inverting f lies i.e we look at predicates which are easy to compute given x, but not when only given $f(x)$. We now formally define this notion.

Definition 2.1 (Hard-Core Predicates (HCP))**.** We say a function family $H = \{h_\lambda : \{0,1\}^\lambda \to \{0,1\}\}$ is a HCP for F if for any PPT A:

$$\Pr\left[A(1^\lambda, f_\lambda(U_\lambda)) = h_\lambda(U_\lambda)\right] = \tfrac{1}{2} + \text{negl}(\lambda)$$

Theorem 2.1. *If an OWP F has an HCP H, then there exists a PRG.*

Proof. For any λ, consider the following function $g_\lambda(s) : \{0,1\}^\lambda \to \{0,1\}^{\lambda+1}$, with $g_\lambda(s) = f_\lambda(s) || h_\lambda(s)$. We can prove that g_λ is a PRG by using the notion of Next-bit unpredictability (from Lecture 2).

Since f_λ is a bijection, the first λ bits of g can be seen as truly random. Finally, note that if there was an algorithm A that could predict the last bit of $g_\lambda(s)$ given the first λ, then it directly implies that A can also be used to predict $h_\lambda(s)$ given $f_\lambda(s)$, contradicting the fact that h_λ is an HCP for f_λ. □

2.1 An HCP for all OWF

While it is not possible to have a single deterministic HCP that works for all OWF, Goldreich and Levin [GL89] showed a *randomized* HCP for any OWF.

Theorem 2.2 (The Goldreich-Levin HCP)**.** *Let F be an OWF. Then for all PPT A:*

$$\Pr\left[A(1^\lambda, f_\lambda(U_\lambda^1), U_\lambda^2) = \langle U_\lambda^1, U_\lambda^2 \rangle \mod 2\right] = \tfrac{1}{2} + \text{negl}(\lambda)$$

Note: The above is equivalent to saying that given an OWF F, we can construct another OWF

$$G = \{g_{2\lambda} : \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda + m(\lambda)}\}_{\lambda \in \mathbb{N}}$$

with

$$g_{2\lambda}(x) = x_1 \dots x_\lambda || f(x_{\lambda+1} \dots x_{2\lambda})$$

such that the following is an HCP for G:

$$H = \{h_{2\lambda}(x)\}_{\lambda \in \mathbb{N}} \text{ with } h_{2\lambda}(x) = \langle x_1 \dots x_\lambda, x_{\lambda+1} \dots x_{2\lambda} \rangle \mod 2.$$

Proof. Assuming PPT A such that

$$\Pr\left[A(1^\lambda, f_\lambda(U_\lambda^1), U_\lambda^2) = \langle U_\lambda^1, U_\lambda^2 \rangle \mod 2\right] = \frac{1}{2} + \epsilon(\lambda)$$

where $\epsilon(\lambda)$ is noticeable. We show that this implies that F was not an OWF to begin with.

Attempt 1: When $\forall x:\ \epsilon(\lambda) = 0.5$

This implies that

$$\forall (x, r):\ A(1^\lambda, f_\lambda(x), r) = \langle x, r\rangle \mod 2$$

We can choose arbitrary r and query for $\langle x, r\rangle \mod 2$. Now let $e_1 = (1, 0, \ldots, 0)$ be the vector of length λ with only $x_1 = 1$ and all other $x_i = 0$. Similarly, e_i is the vector of length λ with only $x_i = 1$.

Notice that $\langle x, e_i\rangle \mod 2 = x_i$; thus, we can simply query A a total of λ times with $r_i = e_i$ (for each $i \in [\lambda]$) and obtain $x_1 \ldots x_\lambda$, contradicting the fact that F is an OWF.

Attempt 2: When $\forall x:\ \epsilon(\lambda) = 0.3$

Assume,

$$\forall x : \Pr\left[A(1^\lambda, f_\lambda(x), U_\lambda) = \langle x, U_\lambda\rangle \mod 2\right] \geq 0.8$$

If we try to use our previous strategy we cannot claim that

$$\forall i:\ \Pr\left[A(1^\lambda, f_\lambda(x), e_i) = \langle x, e_i\rangle \mod 2\right] \geq 0.8$$

However, we can instead select a random r_j, and ask for

$$y_1 = A(1^\lambda, f_\lambda(x), r_j) \text{ and } y_2 = A(1^\lambda, f_\lambda(x), r_j \oplus e_i).$$

A crucial observation is that if we choose r_j uniformly randomly, then the distribution of $r_j \oplus e_i$ is also uniformly random.

Assuming that A answered correctly both times, it follows that $x_i = y_1 \oplus y_2$ by the linearity of inner-products. Thus, by the union-bound,

$$\begin{aligned}\Pr[A \text{ answered correctly both times}] &= 1 - \Pr[A \text{ answered incorrectly at least once}] \\ &\geq 1 - (0.2 + 0.2) = 0.6\end{aligned}$$

Thus with probability at least $0.6 = \frac{1}{2} + \epsilon$ we can obtain x_i for any i. Further, we can amplify this probability to $1 - \frac{1}{4\lambda}$ as follows:

- We will sample k uniformly random r_j, and ask for $y_{1,j} = A(1^\lambda, f_\lambda(x), r_j)$ and $y_{2,j} = A(1^\lambda, f_\lambda(x), r_j \oplus e_i)$. Then, we will output the more frequent value for these $y_{1,j} \oplus y_{2,j}$.
- Let X_j be the indicator random variable for the event that A was successful both times when using r_j and $r_j \oplus e_i$. Let $X = \sum X_j$; we know

  $$\mathrm{E}[X_j] \geq \frac{1}{2} + \epsilon \implies \mathrm{E}[X] \geq (\frac{1}{2} + \epsilon)k$$

  and we want to upper bound the probability $\Pr[X \leq 0.5k]$.
- Note that since $\{X_j \mid j \in [k]\}$ are pairwise independent (in fact all fully independent), we have

  $$\mathrm{Var}[X] = \sum_j \mathrm{Var}[X_j] = \sum_j \Pr[X_j = 1](1 - \Pr[X_j = 1]) \leq k \cdot (0.5 - \epsilon) < 0.5k$$

- Also,
$$\Pr[X \leq 0.5k] \geq \Pr[|X - \mathrm{E}[X]| \leq \epsilon k]$$
By Chebyshev's inequality,
$$\Pr[|X - \mathrm{E}[X]| \geq \epsilon k] \leq \frac{\mathrm{Var}[X]}{\epsilon^2 k^2} \leq \frac{0.5}{\epsilon^2 k}$$

- Thus we can pick $k = 2\lambda/\epsilon^2$, which implies the failure probability is at most $\frac{0.5}{2\lambda} \leq \frac{1}{4\lambda}$ as desired.

Since we can obtain every x_i with probability at least $1 - \frac{1}{4\lambda}$, using the union-bound it follows that we can obtain x with probability at least 0.75, which is noticeable, and contradicts that F is an OWF.

The Full Proof

Notice that because of the union-bound we used in our previous argument, our proof would only work if A was correct with probability greater than 0.75. We will replace that part of our previous proof by doing something smarter so we only need to use A as an oracle once per trial X_j. Also note that our proof did not require our random strings r_j to be completely independent. Instead, we only need pairwise independence to ensure the covariance is 0. We now present the idea for the full proof using these observations.

Proof Idea. Assume we have an algorithm A with
$$\forall x : \Pr\left[A(1^\lambda, f_\lambda(U_\lambda^1), U_\lambda^2) = \langle U_\lambda^1, U_\lambda^2 \rangle \mod 2\right] = \frac{1}{2} + \epsilon(\lambda)$$
where $\epsilon(\lambda)$ is noticeable. Using Markov's inequality, we can get a good set S such that
$$\Pr_{x \leftarrow \{0,1\}^\lambda}[x \in S] \geq \epsilon(\lambda)/2$$
and
$$\forall x \in S : \Pr\left[A(1^\lambda, f_\lambda(x), U_\lambda) = \langle x, U_\lambda \rangle \mod 2\right] \geq \frac{1}{2} + \frac{\epsilon(\lambda)}{2}$$
Consider the following algorithm A' to invert f assuming $x \in S$. Below we use ϵ for $\frac{\epsilon(\lambda)}{2}$.

1. Choose smallest constant c such that $\lambda^c \geq 2\lambda/\epsilon^2$.
2. Generate $l = c \log \lambda$ random strings $r_1, r_2, \ldots, r_l$. By considering all possible subsets of these l strings, and adding them modulo 2, we obtain $2^l = \lambda^c$ pairwise-independent random strings $r'_1, \ldots, r'_{2^l}$.
3. Guess all the values $\{\langle x, r_j \rangle \mid j \in [l]\}$ using random guess. This succeeds with probability 2^{-l}. Assume this happens. Note that in this case $\{\langle x, r'_j \rangle \mid j \in [2^l]\}$ all are correct due to the linearity of the inner-product.
4. Run the same algorithm as in Attempt 2. While determining x_i (for each $i \in [\lambda]$) use the oracle A for $\{A(f(x), r'_j \oplus e_i) \mid j \in [2^l]\}$ only so that $\mathrm{E}[X_j] \geq 0.5 + \epsilon$ for each $j \in [2^l]$.

5. Let x' be the guess for x from previous step. Check if $f(x') = f(x)$ (note f is computable in polynomial time). If so, we return x'. Otherwise go to Step 2.

Replicate the analysis in the previous proof (use $k := 2^l$) to show that assuming all the guesses in Step 3 are correct: $\Pr[x' = x] \geq 0.75$. This must happen at least once after $O(2^l) = O(\frac{1}{\lambda^c})$ iterations of Step 2-5. Thus, we have a PPT algorithm A' that can invert f with noticeable probability contradicting that F is an OWF. □

National University of Singapore
CS4230/CS5430
Foundations of Modern Cryptography
AY 2025/26, Sem 2

Lecture 5 (Feb 12, 2026)
Lecture: Rahul Jain
Notes: Courtesy many

Public-Key Cryptography and Cryptographic Primitives

For the past couple of weeks, we have been diving into details of cryptographic primitives, such as pseudorandom generators and functions, and how they can be utilized in symmetric-key cryptography. Last week, in particular, the focus was on one-way functions and hardcore bits.

This week, we will transition into the domain of *public-key* (or *asymmetric*) encryption and explore both its definition and feasibility. We will conclude with some perspective on the connections between the primitives considered so far.

Previously, our situation for a secure exchange of messages relied on the fact that Alice and Bob had a shared key k (hence the name symmetric-key encryption, since both of them have the same key). The assumptions made alongside were that they must have exchanged the key privately, such that no adversary would be able to have access to that shared secret-key. The key question that arises is:

> *If there is no secret (from Eve) information shared (between Alice and Bob) prior to sending the message, is secure communication still possible?*

In 1974, Ralph Merkle [Mer74] was the first to suggest a plausible working solution to the above problem.

1 Merkle's Protocol

Merkle's protocol assumed:

- the existence of a black box that computes a random function $H : [M] \to [M^{10}]$
- and everyone has access to this black box, including the adversary Eve.

Merkle's protocol (Algorithm 8) is called *non-interactive key exchange*, as the parties act *simultaneously* and do not wait to receive the other's message before sending their own. In general, key exchange protocols could also involve multiple rounds of interaction.

1.1 Successful Agreement on Shared Key

It is easily seen that for all $i, j \in [n] : \Pr[X_i = Y_j] = \frac{1}{M}$. In particular, if we set $n = 1$ and M is much larger, we realize that the probability of success is very small. The aim then is to deduce the value of n for a given M such that we have at least one successful key match with a constant probability.

Algorithm 8: Key establishment

1 Alice samples n random numbers $X_1, \ldots, X_n \leftarrow [M]$ and sends $H(X_1), \ldots, H(X_n)$ to Bob
2 Simultaneously, Bob samples n random numbers $Y_1, \ldots, Y_n \leftarrow [M]$ and sends $H(Y_1), \ldots, H(Y_n)$ to Alice
3 Both find the first pair (i, j) such that $H(X_i) = H(Y_j)$
4 **if** *a pair* (i, j) *is found* **then**
5 establish the key $X_i = Y_j$ `// fails w.p. at most` $O(M/M^{10})$
6 **else**
7 protocol fails

Lemma 1.1. *If $n \geq 100\sqrt{M}$, there is a constant probability that Alice and Bob will find a shared key.*

Proof Sketch. For each i, j, define indicator random variables,

$$I_{ij} := \begin{cases} 1 & \text{if } X_i = Y_j \ , \\ 0 & \text{if } X_i \neq Y_j \ . \end{cases}$$

Note that $\mathrm{E}[I_{i,j}] = \frac{1}{M}$. Let I be the total number of (i, j) s.t $X_i = Y_j$. From the linearity of expectation,

$$\mathrm{E}[I] = \mathrm{E}\left[\sum_{i,j \in [n]} I_{ij}\right] = \sum_{i,j} \mathrm{E}[I_{i,j}] = n^2 \times \frac{1}{M}$$

When n^2 becomes larger than M, in expectation, we see at least 1 possible key exist. However, we are still not done. We need to ensure that the probability of there being such a collision is large, not just that the expected number of collisions is larger than 1. This is a variant of the well-known *Birthday Paradox*. We start the proof of this below, and completing it is left as an exercise.

Note $\forall (i, j) \neq (i', j') : I_{i,j}$ and $I_{i',j'}$ are independent events. We have:

$$\mathrm{Var}[I] = \mathrm{E}\left[I^2\right] - \mathrm{E}[I]^2 \, ,$$

$$\mathrm{E}\left[I^2\right] = \mathrm{E}\left[\left(\sum_{i,j} I_{i,j}\right)^2\right] = \sum_{i,j} \mathrm{E}\left[I_{i,j}^2\right] + \sum_{(i,j) \neq (i',j')} \mathrm{E}\left[I_{i,j} \cdot I_{i',j'}\right]$$

To complete the proof:

- finish computing the variance of I
- use Chebyshev's inequality to show that the probability that $I > 0$, assuming $n = 100\sqrt{M}$, is at least some constant.

□

1.2 Security

Here, there is an agreed upon (i, j) between Alice and Bob that helps to get the shared key. And all that Eve knows is $H(X_i)$ and $H(Y_j)$ alongside the fact that $H(X_i) = H(Y_j)$. Eve will only be successful in finding the key if she, at random, managed to correctly guess X_i and query that into the black box. There is no other way for Eve to know of what X_i is because there exists no other output that can leak information about it (as the function H is completely random). If Eve makes q queries,

$$\Pr\left[\text{Eve has successfully made the query } X_i\right] = \frac{q}{M}$$

This implies that the number of queries Eve needs to make to find x_i with some constant probability is $\Omega(M)$.

- So from Section 1.1, Bob and Alice have to make only $O(\sqrt{M})$ queries to the black box to establish a shared key (with constant success probability).
- However, Eve will have to make quadratic the number of Alice and Bob's queries, i.e $\Omega(M)$ queries, to find the key.

This implies that even if honest parties perform less computation than dishonest parties, they can still have an advantage in terms of security.

This protocol gave hope for the existence of Public-Key Cryptography and was subsequently followed with more work to help improve the security gap: ensuring that the scheme can be secure against PPT adversaries. This will be explored further in the upcoming lectures.

2 Public-Key Encryption

We now move from key-agreement protocols to well-established public-key encryption schemes. Some examples include the Diffie-Hellman key exchange and the RSA encryption schemes, but for this lecture, we are only looking at them in a collective, general sense. These schemes have the following form.

Public-Key Encryption (PKE)

The message space, public-key space, secret-key space, and ciphertext space are $\mathcal{M}$, $\mathcal{P}$, $\mathcal{S}$, and $\mathcal{C}$, respectively.

$\mathsf{KeyGen}(1^\lambda)$**:**

- outputs two keys: a public-key pk and a secret-key sk

$\mathsf{Enc}(1^\lambda, pk, m)$**:**

- Output ciphertext, $c \in \mathcal{C}$

$\mathsf{Dec}(1^\lambda, sk, c)$**:**

- Output $\hat{m} \in \mathcal{M}$

Let's say if Alice wants to send a message only for Bob's eyes. She can use Bob's public-key pk (available to everyone) to encrypt the message. Upon reception by Bob (who has the secret-key sk generated alongside the public-key pk), the ciphertext can be successfully decrypted with the help of sk.

Correctness. Correctness is defined in a manner analogous to that for secret-key encryption.

Definition 2.1 (Correctness of PKE)**.** A PKE scheme is correct if for all λ and m, we have

$$\Pr_{(pk,sk)\leftarrow\mathsf{KeyGen}(1^\lambda),\ c\leftarrow\mathsf{Enc}(1^\lambda,pk,m),\ \hat{m}\leftarrow\mathsf{Dec}(1^\lambda,sk,c)}[\hat{m}=m]=1$$

That is, if encryption and decryption are done right, then $\hat{m} = m$. There are some encryption schemes where the probability is negligibly close to 1 but not equal; however, for the scope of this lecture, we will only take the case listed above.

CPA Security. The definition for this is established through a game very similar to the one shown for Secret-Key Encryption in previous lectures.

IND-CPA(λ):

1. Cha starts by picking a random bit, $b \leftarrow \{0, 1\}$.
2. Cha generates: $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$.
3. pk is announced and sent to Adv.
4. (Previously, Adv would have to send several messages $\hat{m}_i$ to Cha to receive their respective ciphertexts. However, here we realize that these steps are completely unnecessary as Adv itself has access to pk used for encrypting.)
5. Adv sends messages m_0, m_1 to Cha.
6. Cha sends $\mathsf{Enc}(1^\lambda, pk, m_b)$ to Adv.
7. Adv responds with their guess b'.
8. Adv wins the game if $b' = b$.

Definition 2.2. The PKE Scheme is said to be CPA-secure if for all PPT Adv,

$$\Pr[\text{Adv wins IND-CPA}(\lambda)] = \frac{1}{2} + \text{negl}(\lambda)$$

3 Cryptographic Primitives and their Dynamics

We explore connections between various cryptographic primitives that we have discussed thus far. The relations we notice are as follows.

1. PKE $\implies$ SKE: Discontinue sharing pk with everyone and only share it with Alice and Bob.

2. PRF $\implies$ SKE: From previous lectures.

3. PRG $\implies$ PRF: From previous lectures.

4. PRF $\implies$ PRG: Given PRF family $f_k(i)$ we have PRG,

$$G(k) := f_k(1), f_k(2), \ldots \text{ (generate polynomially many)}$$

5. PRG $\implies$ OWF

Claim 3.1. *A PRG* $\{G_\lambda : \{0,1\}^\lambda \to \{0,1\}^{\lambda^2}\}_{\lambda \in (N)}$ *is an OWF.*

Proof. Let us assume for a contradiction that it is not an OWF. This implies that $\exists$ PPT $\mathcal{A}$ and a polynomial p such that for infinitely many λ,

$$\Pr_{x \leftarrow \{0,1\}^\lambda,\ y \leftarrow G_\lambda(x),\ x' \leftarrow \mathcal{A}(1^\lambda, y)} \left[G_\lambda(x') = y\right] > \frac{1}{p(\lambda)} \tag{6}$$

The following is a distinguisher D that breaks PRG G.

(a) Inputs to D: 1^λ and string $y \in \{0,1\}^{\lambda^2}$.

(b) Run the inverter on the string y: $x' \leftarrow \mathcal{A}(1^\lambda, y)$.

(c) If $G_\lambda(x') = y$, output 1. Otherwise output 0.

We now compare the probabilities of D outputting 1 on a random and a pseudorandom string.

(a) Using Eq. (6), for infinitely many λ,

$$\begin{aligned}\Pr_{x \leftarrow \{0,1\}^\lambda,\ y \leftarrow G_\lambda(x)} \left[D(1^\lambda, y) = 1\right] &= \Pr_{x \leftarrow \{0,1\}^\lambda,\ y \leftarrow G_\lambda(x),\ x' \leftarrow \mathcal{A}(1^\lambda, y)} \left[G_\lambda(x') = y\right] \\ &> \frac{1}{p(\lambda)}\end{aligned}$$

(b) On the other hand,

$$\Pr_{y \leftarrow \{0,1\}^{\lambda^2}} \left[D(1^\lambda, y) = 1\right] \leq \frac{2^\lambda}{2^{\lambda^2}} = \text{negl}(\lambda)$$

Reasoning: There are 2^{λ^2} (set A) possible random strings. On the other hand, G can output at most 2^λ strings as its input is only λ bits long (set B). Set B is contained in set A. If a y is chosen s.t $y \in A$ and $y \notin B$ then D will always output 0 since no corresponding x' exists.

(c) Distinguishing advantage $\frac{1}{p(\lambda)} - \text{negl}(\lambda)$ is noticeable.

This contradicts that G is a PRG. Hence G is an OWF. □

6. OWF $\implies$ PRG: Proven by [HILL99a].

7. SKE $\implies$ PRG: Proven by [IL89].

8. Existence of a PRG $\implies P \neq NP$

Proof. Let $\{G_\lambda\}$ be a PRG and assume for the contradiction that $P = NP$. Define a language,

$$L_G = \{y \mid \exists \lambda, x : G_\lambda(x) = y\}$$

i.e this is a set of all possible PRG output strings.

(a) It can be seen that $L_G \in NP$.
   i. witness for $y \in L_G : \lambda, x$
   ii. verifier checks $G_\lambda(x) = y$

(b) Since we are assuming that $P = NP$, we have $L_G \in P$. Hence, there exists a deterministic polynomial time algorithm $\mathcal{A}$ that decides if $y \in L_G$.

(c) Distinguisher D for the PRG takes as input $(1^\lambda, y)$ and outputs $\mathcal{A}(y)$. Note
   i. $\Pr_{x \leftarrow \{0,1\}^\lambda,\ y \leftarrow G_\lambda(x)} \left[D(1^\lambda, y) = 1\right] = \Pr_{x \leftarrow \{0,1\}^\lambda,\ y \leftarrow G_\lambda(x)} \left[\mathcal{A}(y)\right] = 1$
   ii. $\Pr_{y \leftarrow \{0,1\}^{m(\lambda)}} \left[D(1^\lambda, y) = 1\right] \leq \frac{2^\lambda}{2^{m(\lambda)}} \leq \frac{1}{2}$ (since $m(\lambda) > \lambda$).
   iii. The difference: $1 - \frac{2^\lambda}{2^{m(\lambda)}} \geq \frac{1}{2}$ is noticeable.

This contradicts that G is a PRG. □

What we realize is that, in an existential sense, these primitives are interconnected. SKE, PRF, PRG, and OWF are referred to as symmetric-key primitives and are all equivalent to each other. You can start from any one and create another (even if it isn't the most efficient option). PKE is an asymmetric-key primitive.

National University of Singapore
CS4230/CS5430
Foundations of Modern Cryptography
AY 2025/26, Sem 2

Lecture 6 (Feb 19, 2026)
Lecture: Rahul Jain
Notes: Courtesy many

Discrete Log, Diffie-Hellman and ElGamal

In the last lecture, we discussed the definition and CPA-security of Public-Key Encryption (PKE). We also learned about Merkle Key Exchange, which is a key-exchange protocol where the honest parties run in time $O(\sqrt{m})$ and the protocol is secure against adversaries that run in time $o(m)$. However, the protocol is not secure against adversaries that run in larger polynomial time. Therefore, in this lecture, we will explore better alternative PKE schemes.

We will start by revisiting the discrete log (DL) problem and one-way function (OWF) families, which are foundational concepts in cryptography. We will then delve into Diffie-Hellman (DH) key exchange and its relationship with DL. Finally, we will cover the ElGamal public-key encryption scheme, which is a PKE analogue of the DH protocol.

1 Notations

Let $\mathbb{Z}_p$ denote the ring of integers modulo p. In other words, it is defined over $\{0, 1, \cdots, p-1\}$ with addition and multiplication modulo p. Let $\mathbb{Z}_p^*$ denote the multiplicative group defined by $\mathbb{Z}_p$. In other words, it is defined over $\{a \in \mathbb{Z}_p \mid \gcd(a, p) = 1\}$ with multiplication modulo p. Throughout this lecture, we only consider prime p for the definition of $\mathbb{Z}_p^*$. Therefore, from here on, we assume p to be prime.

Let PRIMES_λ denote the set of all λ-bit primes (i.e. primes belonging to $[2^{\lambda-1}, 2^\lambda]$). Let $\text{Gen}(G)$ denote the set of all generators of a cyclic group G. We will make use of the following fact: if p is prime, $\mathbb{Z}_p^*$ is a cyclic group with $\Omega(\frac{p}{\log \log p})$ generators.

2 Discrete Logarithm (DL) Problem

2.1 Hardness of Several Operations

Consider the various operations that can be performed in $\mathbb{Z}_p^*$.

Multiplication: It can be easily performed in time $O((\log p)^2)$. It can also be performed in time $O((\log p) \log \log p)$ using algorithms such as the Fast Fourier Transform (FFT).

Exponentiation: Using binary exponentiation (also called repeated squaring), we can compute a^b in $O(M \log b)$ time where M is the time taken for multiplication. Hence, assuming $b \in \mathbb{Z}_p^*$, exponentiation will take $O((\log p)^3)$ time.

Inverse: Given $g \in \mathbb{Z}_p^*$, find g^{-1}. The complexity is $O((\log p)^3)$. This is because $g^{-1} = g^{p-2}$, so finding the inverse is equivalent to finding the exponentiation.

Logarithm: Given a generator $g \in G$ and an element $h = g^x$, x is called the discrete logarithm of h w.r.t. g, denoted as $x = \text{dlog}_g(h)$. Trivially, we can calculate this in $O(p \cdot \mathsf{poly}(\log p))$, but it turns out it's hard to do this in time $\mathsf{poly}(\log p)$. This is known as the discrete logarithm problem.

2.2 Discrete Log Algorithms

The complexity of solving discrete logarithm problem depends on the group, which is discussed below:

- **General Groups G:**
  - **Naive:** $O(|G|)$
  - **Baby-step Giant-step algorithm:** $O(\sqrt{G})$
  - **Pohlig–Hellman algorithm:** $O(\sqrt{q})$ where q = largest prime dividing $|G|$
- **Specific to $\mathbb{Z}_p^*$:**
  - **Index-Calculus:** $2^{O(\sqrt{\log p \log\log p})}$
  - **Number field sieve:** $2^{O((\log p)^{\frac{1}{3}}(\log\log p)^{\frac{2}{3}})}$

2.3 Discrete Logarithm (DL) Assumption

2.3.1 The Hardness of Discrete Logarithm

The hardness of the discrete logarithm (DL) problem is described by the following assumption for groups $\mathbb{Z}_p^*$.

Definition 2.1 (Discrete Log (DL) Assumption)**.** $\forall$ PPT A,

$$\Pr_{p \leftarrow \text{PRIMES}_\lambda,\, g \leftarrow \text{Gen}(\mathbb{Z}_p^*),\, x \leftarrow \mathbb{Z}_{p-1}} [A(p, g, g^x) = x] = \text{negl}(\lambda)$$

2.3.2 Sampling Abilities

In order to be able to use this assumption, we need to be able to sample such p and g efficiently.

Sampling a uniformly random prime number $p \in \mathbf{PRIMES}_\lambda$

- It is possible to test whether a number is prime in poly-time primality test such as the Miller-Rabin test (randomized) or the AKS (named after Manindra Agrawal, Neeraj Kayal, and Nitin Saxena) primality test (deterministic).
- By the Prime Number Theorem, asymptotically, $\Omega(\frac{1}{\lambda})$ fraction of λ-bit numbers are prime.
- So, we can pick λ-bit numbers at random and test them until a prime is found.

Sampling a generator $g \leftarrow \mathbf{Gen}(\mathbb{Z}_p^*)$

- If prime factorization of $(p-1)$ is known, it is easy to test whether given $g \in \mathbb{Z}_p^*$ is a generator. More about this in a later Problem Set.
- Though factorizing $(p-1)$ might be hard, it is possible to generate random prime p together with the factorization of $(p-1)$ using Kalai's algorithm, then randomly select elements from $\mathbb{Z}_p^*$ until a generator is found. [Kal03]
- Another approach is to use cyclic groups of prime order where every element is a generator.

2.3.3 Worst-case and Average-case Hardness

In cryptography, it is important to consider not just the worst-case hardness of a problem, but also its average-case hardness. In other words, we want to ensure that the problem is difficult to solve not just for specific, worst-case inputs, but for random inputs as well. This is because we do not want to inadvertently use easy instances of problems in our cryptographic constructions.

The discrete log problem in the group $\mathbb{Z}_p^*$ is an example of a problem that is believed to be hard not just in the worst-case, but also on average. The average-case hardness of the problem is described by the discrete log assumption, which states that for any PPT A and for any security parameter λ, there exists a negligible function ν such that for **random** primes p in the set of λ-bit prime numbers PRIMES_λ and for **random** $g \leftarrow \text{Gen}(\mathbb{Z}_p^*)$ and $x \leftarrow \mathbb{Z}_{p-1}$, the probability that A correctly computes x given p, g, and g^x is less than $\nu(\lambda)$.

It turns out that for any fixed prime p, the discrete log problem is as hard for random g and x as it is in the worst-case. That is, if there is an efficient algorithm A exists that solves a random instance of the discrete log problem (with the prime p fixed) with non-negligible advantage, then there exists another efficient algorithm B that solves all instances of the problem (for the same prime p) with advantage close to 1. We cover this reduction in a later Problem Set.

In general, problems are often easier for random inputs than for worst-case inputs. However, in the case of the discrete logarithm problem in $\mathbb{Z}_p^*$, the average-case instance is as hard as the worst-case one, which makes useful for cryptography.

2.3.4 Constructing OWF from DL

The DL assumption can be used to construct a one-way function (OWF). The simplest construction of such a function is:

$$f_\lambda(p, g, x) = (p, g, g^x)$$

where p is a λ-bit prime number, $g \leftarrow \text{Gen}(\mathbb{Z}_p^*)$ is a generator of the group $\mathbb{Z}_p^*$ and $x \in \mathbb{Z}_{p-1}$.

Inverting this function on random inputs would break the DL assumption, hence DL $\implies$ OWF. Also, we need to be a bit careful in defining our OWF. Random inputs of the above form are not random strings, which are what the definition of OWF uses. To deal with this issue, the function can be slightly modified. We can define f_λ so that it takes as input a string r that is then used as the source of randomness to sample p, g, x, and obtain the outputs (p, g, g^x).

Although DL implies more sophisticated primitives such as *collision-resistant hash functions*, it does not directly imply public-key encryption (PKE). However, PKE can be obtained from the hardness of a related problem.

3 Diffie-Hellman (DH) Problem

It is often convenient to abstract away the details of the specific groups used and focus on the essentials. The description of a group can be generated using a PPT algorithm called GroupGen. Given an input of 1^λ, GroupGen outputs a description of a group G_λ (in our discussion so far, this would be the prime p) along with a random generator g.

Definition 3.1 (Computational Diffie-Hellman Problem)**.** Given a cyclic group G of order Q, a generator g, and two elements $g^x, g^y \in \mathbb{Z}_Q$, compute g^{xy}.

In any group where discrete log can be solved efficiently, clearly CDH problem can also be solved efficiently. The hardness of the CDH problem is believed to be at least as much as the hardness of the discrete logarithm (DL) problem in the same group, though we have no formal proof of this. In other words, if the DL problem is hard in a particular group, it is expected that the CDH problem is also hard in that group. The CDH problem is an important problem in cryptography that it is widely used in various cryptographic protocols and systems.

Definition 3.2 (Computational Diffie-Hellman (CDH) Assumption)**.** For family of cyclic groups defined by a PPT algorithm GroupGen, $\forall$ PPT A

$$\Pr_{(G_\lambda, g) \leftarrow \text{GroupGen}(1^\lambda),\, x,y \leftarrow \mathbb{Z}_{|G_\lambda|}} [A(G_\lambda, g, g^x, g^y) = g^{xy}] = \text{negl}(\lambda)$$

In many groups, it is not only difficult to compute g^{xy} in this manner, but also to identify it. To reflect this stronger notion, the Decisional Diffie-Hellman Assumption is often used.

Definition 3.3 (Decisional Diffie-Hellman (DDH) Assumption)**.** $\forall$ PPT A

$$\begin{aligned}&\Pr_{(G_\lambda, g) \leftarrow \text{GroupGen}(1^\lambda),\, x,y \leftarrow \mathbb{Z}_{|G_\lambda|}} [A(G_\lambda, g, g^x, g^y, g^{xy}) = 1] \\ &\quad - \Pr_{(G_\lambda, g) \leftarrow \text{GroupGen}(1^\lambda),\, x,y,z \leftarrow \mathbb{Z}_{|G_\lambda|}} [A(G_\lambda, g, g^x, g^y, g^z) = 1] \\ &= \text{negl}(\lambda)\end{aligned}$$

Intuitively, this means that it is hard for a PPT algorithm to distinguish between g^{xy} and an independently random g^z.

3.1 Diffie-Hellman Key Exchange

Consider the following procotol:

1. $(G_\lambda, g) \leftarrow \text{GroupGen}(1^\lambda)$ is known publicly.
2. Alice samples $x \leftarrow \mathbb{Z}_{|G_\lambda|}$ and sends g^x over to Bob.
3. Simultaneously, Bob samples $y \leftarrow \mathbb{Z}_{|G_\lambda|}$ and sends g^y over to Alice.
4. Since Alice knows x, g^y, she can compute $g^{xy} = (g^y)^x$. Similarly, Bob can also compute g^{xy}.
5. An attacker Eve would be able to see (G_λ, g, g^x, g^y).

Clearly, Alice and Bob share the same element g^{xy} at the end. Further, under the CDH assumption, this protocol has the following security properties, making it a key exchange protocol secure against any polynomial-time adversary.

- Assuming CDH, Eve cannot compute g^{xy}. So, Alice can use a hardcore predicate for the OWF:
$$f_\lambda(G_\lambda, g, g^x, g^y, g^{xy}) \to (G_\lambda, g, g^x, g^y)$$
to pad a secret message to Bob.
- Assuming DDH, Eve cannot distinguish g^{xy} from uniformly random g^z. So, Alice can use g^{xy} as pad to send a secret message to Bob.

3.2 ElGamal Encryption

The ElGamal Encryption is a PKE scheme based on DDH. It is the PKE analogue of the Diffie-Hellman Key Exchange protocol.

Definition 3.4 (ElGamal Encryption)**.**

Key Generation $\text{Gen}(1^\lambda)$:
$(G_\lambda, g) \leftarrow \text{GroupGen}(1^\lambda)$
$x \leftarrow \mathbb{Z}_{|G_\lambda|}$
Output $pk = (G_\lambda, g, g^x),\ sk = x$

Encryption $\text{Enc}(1^\lambda, pk, m \in G_\lambda)$:
$y \leftarrow \mathbb{Z}_{|G_\lambda|}$
Output $(g^y, (g^x)^y m)$

Decryption $\text{Dec}(1^\lambda, sk, c)$:
Using g^y from c and x from sk, compute $(g^y)^x = g^{xy}$.
Recover m as $m = (g^{xy})^{-1}(g^{xy}m)$

Correctness is immediate.

3.2.1 CPA Security of ElGamal Encryption

The security of the ElGamal encryption scheme is based on DDH. To show the security of the scheme, we consider the IND-CPA game. In this game, an adversary is given either $(G_\lambda, g, g^x, g^y, g^{xy}m_0)$ or $(G_\lambda, g, g^x, g^y, g^{xy}m_1)$ and the goal is to distinguish between the two.

Since DDH is believed to be hard in the group described by the algorithm GroupGen, it follows that the two cases above are computationally indistinguishable from $(G_\lambda, g, g^x, g^y, g^z m_0)$ or $(G_\lambda, g, g^x, g^y, g^z m_1)$, respectively. However, since both of these have the same distribution, it follows that the first two cases are themselves computationally indistinguishable.

The above proof is only really valid if m_0 and m_1 are chosen independently of (x, y), etc. But, this is not an assumption of the IND-CPA game. The full proof of CPA-security is not much more complicated, and is left as an exercise.

3.3 Candidates for Groups

We said earlier that CDH is believed to be hard in groups where DL is hard, in particular $\mathbb{Z}_p^*$. This is not quite true for DDH, which in fact turns out to be easy in $\mathbb{Z}_p^*$. This is because g^{xy}, even if not computable, may have efficiently detectable properties that random group elements may not.

For example, if either x or y is even, then there is some $h \in \mathbb{Z}_p^*$ s.t. $h^2 = g^{xy}$. This happens with probability $\frac{3}{4}$ whereas a random $g^z \in \mathbb{Z}_p^*$ is a square with probability only $\frac{1}{2}$.

For a given $g \in \mathbb{Z}_p^*$, we can check for the existence of an $h \in \mathbb{Z}_p^*$ such that $g = h^2$ by checking if $g^{\frac{p-1}{2}} = 1$.

So, we generally choose other groups in which DDH assumption is believed to hold. The most common family of such groups is that of *quadratic residues* of $\mathbb{Z}_p^*$ for the so-called *safe-primes* p. ($g \in G$ is a quadratic residue if $\exists h : g = h^2$.) A prime p is safe if $q = \frac{p-1}{2}$ is also prime (such q's are called Sophie-Germain primes).

It is important to note that all of these problems can be solved in polynomial time by a quantum computer using Shor's algorithm [SHO97].

National University of Singapore
CS4230/CS5430
Foundations of Modern Cryptography
AY 2025/26, Sem 2

Lecture 7 (March 12, 2026)
Lecture: Rahul Jain
Notes: Courtesy many

Trapdoor Permutations, RSA, Quadratic Residuosity

Last week we covered discrete logarithms and the Diffie-Hellman key exchange protocol. Today, we will discuss trapdoor permutations with examples RSA and Rabin trapdoor permutations.

1 Trapdoor permutation

Intuitively, a trapdoor permutation (TDP) is a permutation that can be effectively computed on an input if the key is given, and efficiently inverted on an input if the trapdoor value is given, but hard to invert otherwise. Formally:

Definition 1.1 (Trapdoor permutation (TDP))**.** A TDP consists of three algorithms as follows.

- $\mathsf{Gen}(1^\lambda)$: outputs a public evaluation key k and a trapdoor t. This is a PPT algorithm.
- $\mathsf{Eval}(1^\lambda, k, x)$: for $x \in D_k$, output some $y \in D_k$. This is a PT algorithm. This defines a function
$$f_k : D_k \to D_k, \quad f_k(x) = \mathsf{Eval}(1^\lambda, k, x).$$
- $\mathsf{Invert}(1^\lambda, t, y)$: for $y \in D_k$, output $\hat{x} \in D_k$. This is a PPT algorithm.

A TDP must satisfy the following three properties.

- **Sampleability:** For all k in the support of $\mathsf{Gen}(1^\lambda)$, it's possible to sample from D_k effectively. In other words, for any valid k, $x \leftarrow D_k$ can be sampled in time polynomial in λ.
- **One-wayness:** For all PPT $\mathcal{A}$,
$$\Pr_{(k,t)\leftarrow\mathsf{Gen}(1^\lambda), x\leftarrow D_k, y\leftarrow\mathsf{Eval}(1^\lambda,k,x), \hat{x}\leftarrow\mathcal{A}(1^\lambda,k,y)}\left[\mathsf{Eval}(1^\lambda, k, \hat{x}) = y\right] = \mathrm{negl}(\lambda).$$
- **Invertibility:** For all λ,
$$\Pr_{(k,t)\leftarrow\mathsf{Gen}(1^\lambda), x\leftarrow D_k, y\leftarrow\mathsf{Eval}(1^\lambda,k,x), \hat{x}\leftarrow\mathsf{Invert}(1^\lambda,t,y)}\left[\hat{x} = x\right] = 1.$$

Note that invertibility implies that f_k is indeed a permutation.

2 Constructing a public-key encryption scheme using a TDP

The most natural method to encrypt/decrypt messages is to use $c = \mathsf{Eval}(1^\lambda, k, m)$ to encrypt and $\hat{m} = \mathsf{Invert}(1^\lambda, t, c)$ to decrypt, however this method cannot be CPA-secure because, for one, it is deterministic. A working scheme is described below. Using a TDP $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Invert})$, we can construct a public-key encryption scheme (PKE) as follows, which encrypts 1-bit messages m.

- $\mathsf{KeyGen}(1^\lambda)$: Compute $(k, t) \leftarrow \mathsf{Gen}(1^\lambda)$, output pk, sk where $pk \leftarrow k$ is the public key and $sk \leftarrow t$ is the private/secret key.
- $\mathsf{Enc}(1^\lambda, k, m)$: Sample $x \leftarrow D_k$, compute $y \leftarrow \mathsf{Eval}(1^\lambda, k, x)$. The output is the encrypted ciphertext $c = (y, H_k(x) \oplus m)$, for some H_k being a hardcore predicate for f_k.
- $\mathsf{Dec}(1^\lambda, t, (y, c))$: Compute $\hat{x} \leftarrow \mathsf{Invert}(1^\lambda, t, y)$, and output $H_k(\hat{x}) \oplus c$.

We will prove that this PKE system is secure. Write $P_\lambda \approx_C Q_\lambda$ if the (ensemble of) distributions P_λ and Q_λ are computationally indistinguishable. Note that $\approx_C$ is transitive. We omit the subscript λ when it is clear from the context.

Claim 2.1 (Security of PKE system constructed from TDP)**.** *If for each k:*

- *f_k is a one-way permutation, and*
- *H_k is a hardcore predicate for f_k,*

then for $(PK, SK) \leftarrow \mathsf{KeyGen}(1^\lambda)$,

$$(PK, \mathsf{Enc}(0)) \approx_C (PK, \mathsf{Enc}(1)).$$

Proof. Let K be the random variable representing the key and $X \leftarrow D_K$. Note that

$$(PK, \mathsf{Enc}(0)) = (K, f_K(X), H_K(X) \oplus 0).$$

As we saw in previous lectures, if H_k is hardcore for one-way permutation f_k, then

$$G(k, x) = (k, f_k(x), H_k(x))$$

is a PRG. So

$$(K, f_K(X), H_K(X)) \approx_C (K, R_1, R_2),$$

where $R_1 \leftarrow D_k$ and $R_2 \leftarrow \{0, 1\}$ respectively. Finally,

$$(K, f_K(X), H_K(X) \oplus 0) \approx_C (K, R_1, R_2 \oplus 0) = (K, R_1, R_2 \oplus 1) \approx_C (K, f_K(X), H_K(X) \oplus 1),$$

therefore the two distributions are indistinguishable, and the encryption scheme is secure. □

3 The RSA TDP

We will discuss two TDPs, RSA and Rabin. Both use the hardness of factorizing large numbers. The RSA TDP is derived from one of the earliest constructions of public-key cryptography by Rivest, Shamir and Adleman in 1977 [RSA78].

The RSA encryption scheme is built from the following TDP.

- $\mathsf{Gen}(1^\lambda)$: We do the following steps:
  - sample random λ-bit primes p and q, set $N = p \times q$,
  - pick $e \in \mathbb{Z}_N^*$ such that $\gcd(e, \phi(N)) = 1$, where ϕ is the *Euler Totient function*, $\phi(N) = (p-1)(q-1)$,
  - compute $d = e^{-1} \pmod{\phi(N)}$,
  - output $k = (N, e)$ and $t = (N, d)$.
- $\mathsf{Eval}(1^\lambda, k, x) = \mathsf{Eval}(1^\lambda, (N, e), x) = x^e \pmod N$, and
- $\mathsf{Invert}(1^\lambda, t, x) = \mathsf{Invert}(1^\lambda, (N, d), y) = y^d \pmod N$.

The reason this works is because if generator $g \in \mathbb{Z}_N^*$, then $g^{\phi(N)} = 1$. As such multiplication, and exponentiation in $\mathbb{Z}_N^*$ can be thought of as addition and multiplication modulo $\phi(N)$.[3]

The following problem is believed to be difficult – if $N, e, x^e \pmod N$ are given, then it's believed to be difficult to compute x. One way to compute x is to factorize N to get p and q, then $\phi(N)$ and d can be computed, however, this may or may not be the only way.

We make the following assumption, under which the RSA TDP is immediately seen to be secure.

Assumption 3.1 (RSA assumption)**.** *For all PPT $\mathcal{A}$,*

$$\Pr_{(N,e)\leftarrow\mathsf{Gen}(1^\lambda), x\leftarrow\mathbb{Z}_N^*}[\mathcal{A}(N, e, x^e \bmod N) = x] = \mathrm{negl}(\lambda).$$

Given the RSA assumption, factoring is hard. Nevertheless, the reverse implication is not known to hold. So, the RSA assumption is potentially stronger than the assumption that factoring is hard. We do have a TDP assuming only that factoring is hard: the Rabin TDP.

4 The Rabin TDP

Definition 4.1 (Quadratic residue)**.** Let the set of quadratic residues modulo N be

$$\mathsf{QR}_N = \{y \in \mathbb{Z}_N^* \mid \exists x \in \mathbb{Z}_N^*, x^2 \equiv y \pmod N\}.$$

The following TDP was originally designed for a digital signature scheme by Michael Rabin in 1978 [Rab79].

- $\mathsf{Gen}(1^\lambda)$: Sample random λ-bit primes $p \neq q$, such that $p = 3 \pmod 4$ and $q = 3 \pmod 4$. Output key $k = N = p \times q$, trapdoor $t = (p, q)$.

[3] If p and q are odd and distinct, then all the elements in $\mathbb{Z}_N^*$ must necessarily have an order that divides $\mathrm{lcm}(p-1, q-1) \mid \frac{\phi(N)}{2}$, so it suffices to consider modulo $\frac{\phi(N)}{2}$ as well.

- $\mathsf{Eval}(1^\lambda, N, x)$: Output $x^2 \pmod N$.
- $\mathsf{Invert}(1^\lambda, (p,q), y)$: Find and output an $x \in \mathsf{QR}_N$ s.t $x^2 = y \pmod N$.

The Rabin TDP is:

$$f_N : \mathsf{QR}_N \to \mathsf{QR}_N$$
$$f_N(x) = x^2 \pmod N.$$

This is indeed a permutation, as shown below.

We note that for prime p, $|\mathsf{QR}_p| = \frac{p-1}{2}$ due to the following. Let g be any generator of $\mathbb{Z}_p^*$, then all the elements are $\{g^0, g^1, g^2, \ldots, g^{p-2}\}$. The set of possible values of x^2 is $\{g^0, g^2, g^4, \ldots\}$, and $g^{2\times\frac{p-1}{2}}$ is 1. This shows that $|\mathsf{QR}_p| \geq \frac{p-1}{2}$. On the other hand, each quadratic residue has at least two square roots. This shows that $|\mathsf{QR}_p| \leq \frac{p-1}{2}$. Hence $|\mathsf{QR}_p| = \frac{p-1}{2}$.

We have the following fundamental result in number theory.

Theorem 4.1 (Chinese Remainder Theorem)**.** *For $p \neq q$ primes, and $N = pq$, the ring $\mathbb{Z}_N$ is isomorphic to $\mathbb{Z}_p \times \mathbb{Z}_q$ by the mapping $f : \mathbb{Z}_N \to \mathbb{Z}_p \times \mathbb{Z}_q$, $f(x) = (x \bmod p, x \bmod q)$.*

For any $y \in \mathbb{Z}_N$, write $y = (y_p, y_q)$ (as an element of $\mathbb{Z}_p \times \mathbb{Z}_q$). We note that $y \in \mathsf{QR}_N$ if and only if $y_p \in \mathsf{QR}_p$ and $y_q \in \mathsf{QR}_q$ (why? home exercise, determine what is f^{-1}). So, $|\mathsf{QR}_N| = |\mathsf{QR}_p| \cdot |\mathsf{QR}_q| = \frac{p-1}{2} \cdot \frac{q-1}{2} = \frac{\phi(N)}{4}$. In other words, the image of f_N is only one fourth of $\mathbb{Z}_N^*$.

If we choose $p \neq q$ such that $p \equiv q \equiv 3 \pmod 4$, then f_N is a permutation on QR_N. To show this, we prove the following lemma.

Lemma 4.2. *If $p \equiv 3 \pmod 4$, then for all $x \in \mathbb{Z}_p^*$, exactly one of x or $-x$ is a quadratic residue.*

Proof. Let g be any generator of $\mathbb{Z}_p^*$, and write $x = g^a$. Then, $g^{\frac{p-1}{2}} \equiv -1 \pmod p$, so $-x = g^{a+\frac{p-1}{2}}$. Note that $\frac{p-1}{2}$ is odd, so exactly one of a or $a + \frac{p-1}{2}$ is even. □

To invert f_N given the trapdoor, for any quadratic residue $y = (y_p, y_q)$, let $x = (x_p, x_q)$ be any square root of it, then all of its square roots are $(x_p, x_q), (x_p, -x_q), (-x_p, x_q), (-x_p, -x_q)$. From Lemma 4.2, exactly one of x_p or $-x_p$ is in QR_p and similarly exactly one of x_q or $-x_q$ is in QR_q. Therefore exactly one of the 4 square roots is in QR_N.

Note that if $p \equiv 3 \pmod 4$, then for all quadratic residues y:

$$\left(y^{\frac{p+1}{4}}\right)^2 \equiv y^{\frac{p+1}{2}} \equiv y^{\frac{p-1}{2}+1} \equiv \left(y^{\frac{p-1}{2}}\right) \cdot y \equiv y \pmod p.$$

Hence, given p, it's efficient to compute square root modulo p for a quadratic residue y. In particular, we can invert Rabin's TDP given the factorization as follows. The algorithm $\mathsf{Invert}((p,q), y)$ works by

$$y \to (y_p, y_q) \xrightarrow{\text{square roots}} (x_p, x_q) \to x.$$

Now, we prove the following.

Claim 4.1. *Given $x, y, z \in \mathbb{Z}_N^*$ such that $x^2 \equiv z^2 \equiv y \pmod N$ and $x \notin \{\pm z\}$, we can factorize N.*

Proof. If we're given $x = (x_p, x_q)$ and $z = (x_p, -x_q)$, then we can compute q by adding the 2 values to get $(2x_p, 0)$. Note that this is zero modulo q but is nonzero modulo p. Hence, $\gcd(x+z, N) = q$. □

To prove the one-wayness of the Rabin trapdoor function, we make the following assumption about the hardness of factorization.

Assumption 4.3 (Rabin assumption, hardness of factorization)**.** *For all PPT $\mathcal{A}$,*

$$\Pr_{(N,p,q)\leftarrow \mathsf{Gen}(1^\lambda)}[\mathcal{A}(N) = (p, q)] = \mathrm{negl}(\lambda).$$

With the Rabin assumption, we have the following claim.

Claim 4.2. *The function f_N is OWF.*

Proof. Note that, if we have an algorithm $\mathcal{A}$ that inverts f_N, then if we sample $x \leftarrow \mathbb{Z}_N^*$, compute $y = x^2$ and $z = \mathcal{A}(y)$. Then,

$$\Pr[x \notin \{\pm z\}] = \frac{1}{2}.$$

Therefore we can factorize N using the method shown in Claim 4.1. □

National University of Singapore
CS4230/CS5430
Foundations of Modern Cryptography
AY 2025/26, Sem 2

Lecture 8 (Mar 19, 2026)
Lecture: Rahul Jain
Notes: Courtesy many

Message Authentication Codes, CCA-secure Encryption

Thus far, the cryptosystems we have studied all define security in terms of *privacy*: Alice wants to send a message m to Bob, such that Eve, who is monitoring their communication channel, cannot learn about the contents of m. We have constructed several encryption schemes that allow Alice to transmit a ciphertext c instead of m, which can be decrypted by Bob to recover m correctly. We have also shown that (loosely speaking) in order for Eve to learn anything about m by observing c, she would have to solve a computational problem that is widely believed to be hard. In this sense, the contents of m are kept secret from Eve, and the encryption scheme can be said to be secure.

In this week, we allow the (previously passive) adversary Eve to actively interfere with Alice and Bob's communication in two possible ways:

1. She can send her own "spoofed" message m' to Bob on the same communication channel, attempting to trick Bob into thinking that m' was sent by Alice.
2. She can "malle" the message m that Alice sent, replacing it with another message $m' \neq m$.

A cryptosystem that is secure against the first kind of attack is said to provide *authentication*: Bob can be sure that the message he receives comes from Alice. Security against the second kind of "man-in-the-middle" attack is also known as *integrity*: Bob knows that the contents of the message were not tampered by Eve.

These security objectives are orthogonal to privacy/secrecy: even the one-time pad does not defend against such attacks. What is needed is an algorithm that Bob can use to distinguish Eve's messages from the genuine, untampered messages sent by Alice. In the symmetric-key setting, where Alice and Bob can securely share a key k beforehand, this is provided by a *message authentication code* (MAC).

1 Message authentication codes (MACs)

A *MAC scheme* is a triple of algorithms $(\mathsf{Gen}, \mathsf{MAC}, \mathsf{Verify})$ that performs the following:

- $\mathsf{Gen}(1^\lambda)$: Outputs a key k
- $\mathsf{MAC}(1^\lambda, k, m)$: Given key k and message m, outputs a tag t
- $\mathsf{Verify}(1^\lambda, k, m, t)$: Given key k and message m with tag t, decide to accept (output 1) or reject (output 0)

Alice and Bob can use this MAC scheme in a similar way to an encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$. Specifically: they confer securely beforehand to agree on a symmetric key $k \leftarrow \mathsf{Gen}(1^\lambda)$. Then, when Alice wants to send a message m, she authenticates it by running $t \leftarrow \mathsf{MAC}(1^\lambda, k, m)$ and sending the tagged message (m, t). When Bob receives a message (m', t'), he verifies it by running $\mathsf{Verify}(1^\lambda, k, m', t')$, and depending on the output he either accepts (m', t') as having come from Alice, or rejects it.[4]

As with encryption schemes, we desire that the MAC scheme be correct and secure. The former is straightforward: it means that Bob should always accept correctly generated tags from Alice.

Definition 1.1 (Correctness)**.** A MAC scheme $(\mathsf{Gen}, \mathsf{MAC}, \mathsf{Verify})$ is *correct* if, for any message m,

$$\Pr_{k \leftarrow \mathsf{KeyGen}(1^\lambda),\, t \leftarrow \mathsf{MAC}(1^\lambda, k, m)} \Big[\mathsf{Verify}(1^\lambda, k, m, t) = 1 \Big] = 1$$

Security is, as usual, more nuanced: there are several possible definitions, each based on the kinds of attacks that we would like to defend against. Ordered from strongest to weakest, these include:

- **Total break**: Eve can reconstruct the key k.
- **Universal forgery**: For *every* message m, Eve can construct a tag t such that Bob accepts (m, t).
- **Existential forgery**: For *some* message m, Eve can construct a tag t such that Bob accepts (m, t).

Our notion of security will be strong enough that it not only renders the weakest attack (existential forgery) impossible, but does so even when Eve is allowed to request valid tags for messages of her choice. In other words, Eve can query the *authentication oracle* $m \mapsto \mathsf{MAC}(1^\lambda, k, m)$ on any message m, and obtain a tag $t \leftarrow \mathsf{MAC}(1^\lambda, k, m)$ such that $\mathsf{Verify}(1^\lambda, k, m, t) = 1$, on the condition that (m, t) no longer counts as a successful existential forgery. This can be formalized in the familiar setting of a game between an adversary (Adv) and a challenger (Cha):

EUF-CMA(λ): "Existential unforgeability under chosen message attack"

1. Cha samples key $k \leftarrow \mathsf{Gen}(1^\lambda)$.
2. For some $l = \mathsf{poly}(\lambda)$, for $i \in [l]$:
   - Adv chooses some message m_i and sends it to Cha.
   - Cha computes $t_i \leftarrow \mathsf{MAC}(1^\lambda, k, m_i)$ and sends it to Adv.
3. Adv outputs a tagged message (m', t').
4. Adv wins iff $\mathsf{Verify}(1^\lambda, k, m', t') = 1$ and $\forall i \in [l]$, $(m', t') \neq (m_i, t_i)$.

[4] The idea of verifying message integrity with tags is similar to using a *checksum*. However, checksums generally only offer protection against "random" modifications of the message (e.g. as caused by channel noise), but not against "worst-case" modifications (e.g. as caused by adversarial tampering).

Definition 1.2 (EUF-CMA security)**.** A MAC scheme $(\mathsf{Gen}, \mathsf{MAC}, \mathsf{Verify})$ is *EUF-CMA-secure* if for any PPT adversary A,

$$\Pr\left[A \text{ wins EUF-CMA}(\lambda)\right] = \text{negl}(\lambda).$$

It turns out that PRF families (as we have defined them in Lecture 3) naturally give rise to a MAC scheme with the desired properties, to the point that some people conflate the two.

Theorem 1.1. *Let* $F_\lambda = \left\{f_k : \{0,1\}^\lambda \to \{0,1\}^{m(\lambda)}\right\}_{\lambda \in \mathbb{N}}$ *be a PRF family. Then the following MAC scheme:*

$\underline{\mathsf{Gen}(1^\lambda)}$*:*

- *Sample a uniformly random* $f_k \leftarrow F_\lambda$
- *Output* k

$\underline{\mathsf{MAC}(1^\lambda, k, m)}$*:*

- *Output* $f_k(m)$

$\underline{\mathsf{Verify}(1^\lambda, k, m, t)}$*:*

- *If* $f_k(m) = t$*, output 1 (accept).*
- *Otherwise, output 0 (reject).*

satisfies correctness and EUF-CMA security.

Proof Sketch of Theorem 1.1. Correctness is immediate.

For security, assume towards a contradiction that some PPT adversary A wins EUF-CMA(λ) with noticeable probability $\geq 1/p(\lambda)$. We show how to use this to construct a PPT adversary B such that $\Pr\left[B \text{ wins FN-IND}(\lambda)\right] - \frac{1}{2}$ is noticeable.

For convenience, we reproduce (from Lecture 3) the game that we want B to play.

$\underline{\text{FN-IND}(\lambda)}$: "Function Indistinguishability"

1. Cha chooses a random bit $b \leftarrow \{0,1\}$. If $b = 0$, Cha samples a random function $f \leftarrow F_\lambda^{all}$. If $b = 1$, Cha samples a random function from the PRF family $f \leftarrow F_\lambda$.
2. For some $t = \mathsf{poly}(\lambda)$, for $i \in [t]$:
   - Adv chooses some input m_i and sends it to Cha.
   - Cha sends $f(m_i)$ to Adv.
3. Adv outputs a guess bit b'.
4. Adv wins iff $b' = b$.

We define B to play FN-IND(λ) against Cha (with the help of A) by forwarding each of A's queries m_i to Cha, and sending Cha's replies $f(m_i)$ back to A. When A finally outputs a tagged message (m', t'), B queries Cha one final time to obtain $f(m')$, and then checks if $f(m') = t'$ and $(m', t') \neq (m_i, t_i)$ for every previous query. If yes, then B outputs $b' = 1$, i.e. it guesses that Cha sampled $f \leftarrow F_\lambda$ from the PRF family. Otherwise, it outputs a random guess $b' \leftarrow \{0, 1\}$.

Case $b = 1$: A is playing a genuine instance of EUF-CMA(λ), so with at least a noticeable probability $1/p(\lambda)$ its final output (m', t') is "successful", in the sense that $f(m') = t'$ and $\forall m_i, m' \neq m_i$. Then it can be checked that B outputs $b' = 1$ with probability $\geq \frac{1}{2} + \frac{1}{2p(\lambda)}$.

Case $b = 0$: A is successful with at most a negligible probability. Loosely speaking: since $f \leftarrow F_\lambda^{all}$ is equivalent to sampling $f(m) \leftarrow \{0, 1\}^{m(\lambda)}$ independently for each input m, it is equivalent for Cha to "lazily" delay sampling $f(m')$ until B's final query, by which time A has already submitted t', hence $\Pr[f(m') = t'] \leq 2^{-m(\lambda)}$ which is negligible. Therefore, B outputs $b' = 1$ with probability $\leq \frac{1}{2} + \frac{2^{-m(\lambda)}}{2}$.

Finally, by recalling that $\Pr[b = 1] = \Pr[b = 0] = \frac{1}{2}$, we obtain:

$$\Pr[B \text{ wins FN-IND}(\lambda)] - \frac{1}{2} = \frac{1}{2}\left[\Pr\left[b' = 1 | b = 1\right] - \Pr\left[b' = 1 | b = 0\right]\right] \geq \frac{1}{4p(\lambda)} - \frac{2^{-m(\lambda)}}{4}$$

which is the difference of a noticeable and a negligible function, hence is noticeable. □

We close this section with two remarks on the PRF-based MAC scheme defined above.

- The algorithm MAC is deterministic in this case (though it is generally allowed to be randomized). This is in contrast to encryption schemes, where randomization of Enc was necessary for security.
- For every key k and message m, there is exactly one tag t such that (m, t) is accepted.

2 Security against chosen ciphertext attacks (CCA)

Our preceding definition of EUF-CMA security granted the adversary *oracle access* to the algorithm $m \mapsto \mathsf{MAC}(1^\lambda, k, m)$. In the context of encryption schemes, we have seen a similar notion in the form of chosen plaintext attacks (CPA), where Eve was allowed access to $m \mapsto \mathsf{Enc}(1^\lambda, k, m)$ – and in both cases, a meaningful notion of security can still be obtained. In the spirit of Kerckhoff's principle, we may ask if security is still possible even when Eve also has access to $c \mapsto \mathsf{Dec}(1^\lambda, k, c)$: that is, if she can get Bob to decrypt ciphertexts of her choice.[5] For certain encryption schemes $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, such *chosen ciphertext attacks* (CCA) are sufficient to compromise security, as captured by the following game.

[5]This scenario is conceivable, now that Eve is no longer a passive eavesdropper. For example, she could impersonate Alice and send a "spoofed" ciphertext c' of her choice to Bob, then monitor his response to the decrypted message $\mathsf{Dec}(1^\lambda, k, c')$ in order to learn about its value.

IND-CCA2(λ): "Indistinguishability under Chosen Ciphertext Attacks"

1. Cha chooses a random bit $b \leftarrow \{0,1\}$, and samples key $k \leftarrow \mathsf{KeyGen}(1^\lambda)$.
2. For some $l = \mathsf{poly}(\lambda)$, for $i \in [l]$:
   - Adv chooses some message $\hat{m}_i$ and sends it to Cha.
   - Cha sends the corresponding encryption $\mathsf{Enc}(1^\lambda, k, \hat{m}_i)$ to Adv.
   - Adv chooses some ciphertext $\hat{c}_i$ and sends it to Cha.
   - Cha sends the corresponding decryption $\mathsf{Dec}(1^\lambda, k, \hat{c}_i)$ to Adv.
3. Adv chooses two messages m_0, m_1 and sends them to Cha.
4. Cha computes the challenge ciphertext $c = \mathsf{Enc}(1^\lambda, k, m_b)$ and sends it to Adv.
5. For some $l_2 = \mathsf{poly}(\lambda)$, for $i \in [l_2]$:
   - Adv chooses some message $\hat{m}_{l+i}$ and sends it to Cha.
   - Cha sends the corresponding encryption $\mathsf{Enc}(1^\lambda, k, \hat{m}_{l+i})$ to Adv.
   - Adv chooses some ciphertext $\hat{c}_{l+i} \neq c$ and sends it to Cha.
   - Cha sends the corresponding decryption $\mathsf{Dec}(1^\lambda, k, \hat{c}_{l+i})$ to Adv.
6. Adv outputs a guess bit b'.
7. Adv wins iff $b' = b$.

Definition 2.1 (IND-CCA security)**.** An encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is *CCA-secure* if, for any PPT adversary A,

$$\Pr[A \text{ wins IND-CCA2}(\lambda)] = \frac{1}{2} + \mathrm{negl}(\lambda).$$

Note that in Step 5 of the game, Adv is no longer allowed to ask for a decryption of the challenge ciphertext c, otherwise the game becomes trivial. Eliminating Step 5 entirely results in a game called 'IND-CCA1(λ)', with a corresponding notion of 'CCA1 security'.

Claim 2.1. *The (CPA-secure) encryption scheme of Lecture* 3*, which was constructed using a PRF family, is not CCA-secure.*

Proof. We recall the construction of the aforementioned scheme, for a given PRF family F_λ.

$\underline{\mathsf{KeyGen}(1^\lambda)}$**:**

- Sample a uniformly random $f_k \leftarrow F_\lambda$
- Output k

$\underline{\mathsf{Enc}(1^\lambda, k, m)}$**:**

- Sample a uniformly random $i \leftarrow \{0,1\}^\lambda$
- Output $(i, f_k(i) \oplus m)$

$\underline{\mathsf{Dec}(1^\lambda, k, (i, c))}$**:**

- Output $f_k(i) \oplus c$

Now consider any PPT adversary A that chooses any three messages m_0, m_1, s so long as $m_0 \neq m_1$ and $s \neq 0$. It plays IND-CCA2(λ) by immediately sending m_0, m_1 to Cha and receiving (i, c) in response, where $c = f_k(i) \oplus m_b$. Then, since $c \neq c \oplus s$, it can send the (non-challenge) ciphertext $(i, c \oplus s)$ to Cha and receive its decryption

$$f_k(i) \oplus (c \oplus s) = f_k(i) \oplus f_k(i) \oplus m_b \oplus s = m_b \oplus s$$

from which A can recover $m_b = (m_b \oplus s) \oplus s$ and hence b with probability 1, breaking CCA-security. □

This attack was possible because the PRF-based encryption scheme was *malleable* – the adversary A was able to "malle" the ciphertext $\mathsf{Enc}(1^\lambda, k, m_b)$, transforming it into the ciphertext $\mathsf{Enc}(1^\lambda, k, m_b \oplus s)$ of a different but related message, without the need to decrypt it first.

For an encryption scheme to be CCA-secure, *non-malleability* is a necessary condition (it can also be sufficient, depending on precisely how non-malleability is formalized; several possibilities are analyzed by Bellare and Sahai [BS06]). Intuitively, we should not allow the adversary to forge a ciphertext $\hat{c}$ that it had not previously received from the challenger, and this suggests the idea of using a MAC scheme to authenticate the challenger's outputs. If implemented correctly, this indeed suffices.

Theorem 2.1. *Let* $\Pi = (\mathsf{KeyGen}_{CPA}, \mathsf{Enc}_{CPA}, \mathsf{Dec}_{CPA})$ *be a CPA-secure encryption scheme, and let* $\Sigma = (\mathsf{Gen}, \mathsf{MAC}, \mathsf{Verify})$ *be a EUF-CMA-secure MAC scheme. Then the following encryption scheme:*

$\underline{\mathsf{KeyGen}(1^\lambda)\text{:}}$

- *Run* $k_1 \leftarrow \mathsf{KeyGen}_{CPA}(1^\lambda)$ *and* $k_2 \leftarrow \mathsf{Gen}(1^\lambda)$
- *Output* (k_1, k_2)

$\underline{\mathsf{Enc}(1^\lambda, (k_1, k_2), m)\text{:}}$

- *Run* $c \leftarrow \mathsf{Enc}_{CPA}(1^\lambda, k_1, m)$
- *Run* $t \leftarrow \mathsf{MAC}(1^\lambda, k_2, c)$
- *Output* (c, t)

$\underline{\mathsf{Dec}(1^\lambda, (k_1, k_2), (c, t))\text{:}}$

- *If* $\mathsf{Verify}(1^\lambda, k_2, c, t) = 1$*, output* $\mathsf{Dec}_{CPA}(1^\lambda, k_1, c)$ *(accept).*
- *Otherwise, output* $\perp$ *(reject).*

is CCA-secure (and correct if Π *and* Σ *are).*

Remark 2.1. *It is crucial that* Enc *authenticates the ciphertext* c *and not the plaintext* m*, since it is not required for* MAC *to hide its input (for example, the tag* t *could include the input verbatim as a prefix).*

Proof Sketch of Theorem 2.1. As usual, the proof proceeds via security reduction, starting with the assumption of a PPT adversary A that wins IND-CCA2(λ) with a noticeable probability over random guessing. The key idea is to consider the event $\mathcal{E}$ that at some point in the game, A submits a decryption query (c, t) that is a successful existential forgery, in the sense that

$$\mathsf{Verify}(1^\lambda, k_2, c, t) = 1 \text{ and } (c, t) \neq \mathsf{Enc}(1^\lambda, (k_1, k_2), \hat{m}_i)$$

was not the result of any preceding encryption query $\hat{m}_i$. There are two cases:

- **Case 1:** $\Pr[\mathcal{E}]$ **is noticeable** (as a function of λ). Then we can break the EUF-CMA-security of Σ, by constructing a PPT adversary B that copies the submission of A.
- **Case 2:** $\Pr[\mathcal{E}]$ **is negligible**. Then we can break the CPA-security of Π, by constructing a PPT adversary D that simulates an instance of A and takes on the role of the IND-CCA challenger, while responding with $\perp$ to every one of A's "fresh" decryption queries. In other words: D simply assumes that the query will fail, since it has at most a negligible probability of being right.

We will flesh out the details for a simplified setting, where A immediately sends m_0, m_1 to the IND-CCA challenger, and then submits only a single decryption query $(\hat{c}, \hat{t})$ before guessing b'. The proof for the full setting, where A can make queries as in the IND-CCA2(λ) game, is left as an exercise.

Case 1: The PPT adversary B, plays the EUF-CMA(λ) game (while simulating A) against a challenger Cha as follows:

1. Simulate an instance of A, and take on the role of the IND-CCA2(λ) challenger.
2. Sample a random bit $b \leftarrow \{0,1\}$ and key $k_1 \leftarrow \mathsf{KeyGen}_{CPA}(1^\lambda)$.
3. Cha samples $k_2 \leftarrow \mathsf{Gen}(1^\lambda)$.
4. Receive messages m_0, m_1 from A.
5. Compute $c \leftarrow \mathsf{Enc}_{CPA}(1^\lambda, k_1, m_b)$ and send it to Cha.
6. Receive $t \leftarrow \mathsf{MAC}(1^\lambda, k_2, c)$ from Cha, and send the challenge ciphertext (c,t) to A.
7. Receive decryption query $(\hat{c}, \hat{t})$ from A and output it as a tagged message.

From the perspective of A, it is playing a genuine instance of IND-CCA2(λ), and by assumption there is a noticeable probability of its decryption query $(\hat{c}, \hat{t})$ being a successful existential forgery – that is: $\mathsf{Verify}(1^\lambda, k_2, \hat{c}, \hat{t}) = 1$ (recall that the IND-CCA2(λ) game rules require $\hat{c} \neq c$, so this query is necessarily fresh). Therefore, when B outputs $(\hat{c}, \hat{t})$ as a tagged message, it wins EUF-CMA(λ) with noticeable probability, breaking EUF-CMA-security of Σ.

Case 2: The PPT adversary D plays IND-CPA(λ) against Cha as follows:

1. Simulate an instance of A, and take on the role of the IND-CCA2(λ) challenger.
2. Sample key $k_2 \leftarrow \mathsf{Gen}(1^\lambda)$.
3. Cha samples a random bit $b \leftarrow \{0,1\}$ and key $k_1 \leftarrow \mathsf{KeyGen}_{CPA}(1^\lambda)$.
4. Receive messages m_0, m_1 from A, and send them to Cha.
5. Receive $c \leftarrow \mathsf{Enc}_{CPA}(1^\lambda, k_1, m_b)$ from Cha.
6. Compute $t \leftarrow \mathsf{MAC}(1^\lambda, k_2, c)$ and send the challenge ciphertext (c,t) to A.
7. Receive decryption query $(\hat{c}, \hat{t})$ from A, ignore it and send $\perp$ to A.
8. Receive guess bit b' from A and output it as a guess of b.

Assuming that event $\overline{\mathcal{E}}$ holds, then $\mathsf{Verify}(1^\lambda, k_2, \hat{c}, \hat{t}) \neq 1$ and $\mathsf{Dec}(1^\lambda, (k_1, k_2), (\hat{c}, \hat{t})) = \perp$, hence by replying with $\perp$ in Step 6, D is correctly simulating a genuine play of IND-CCA2(λ) against A. Moreover, by assumption A can win (i.e. guess the challenge bit b correctly) with a noticeable probability $\geq 1/p(\lambda)$ over random guessing. Therefore, by copying A's output, D wins precisely when A does, and its overall probability of correctly guessing b is (for infinitely many λ):

$$\begin{aligned}
\Pr\left[D \text{ wins IND-CPA}(\lambda)\right] &\geq \Pr\left[D \text{ wins IND-CPA}(\lambda) \wedge \overline{\mathcal{E}}\right] \\
&= \Pr\left[A \text{ wins IND-CCA2}(\lambda) \wedge \overline{\mathcal{E}}\right] \\
&\geq \Pr\left[A \text{ wins IND-CCA2}(\lambda)\right] - \Pr\left[\mathcal{E}\right] \\
&\geq \frac{1}{2} + \frac{1}{p(\lambda)} - \Pr\left[\mathcal{E}\right].
\end{aligned}$$

which is a noticeable advantage of $1/p(\lambda) - \Pr[\mathcal{E}]$ over random guessing, leading to a contradiction.

In the penultimate line, we have used the probabilistic inequality

$$\Pr\left[A \wedge \overline{B}\right] \geq \Pr[A] - \Pr[B],$$

which follows using the union bound as follows:

$$\Pr[A] = \Pr\left[(A \wedge \overline{B}) \vee (A \wedge B)\right] \leq \Pr\left[A \wedge \overline{B}\right] + \Pr[A \wedge B] \leq \Pr\left[A \wedge \overline{B}\right] + \Pr[B].$$

□

3 Security for public-key encryption (PKE)

The definitions in the previous sections can be adapted to a public-key setting. For example, the public-key analogue of a MAC scheme is a *digital signature* scheme, which consists of the following algorithms:

- $\mathsf{Gen}(1^\lambda)$: Outputs two keys: a (public) verification key vk and a (secret) signing key sk
- $\mathsf{Sign}(1^\lambda, sk, m)$: Given signing key sk and message m, outputs a signature σ
- $\mathsf{Verify}(1^\lambda, vk, m, \sigma)$: Given verification key vk and message m with signature σ, decide to accept (output 1) or reject (output 0)

EUF-CMA security is defined using an analogous security game, except that after $(vk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ is sampled, the verification key vk is immediately given to the adversary.

Naor and Yung [NY89a] showed that digital signature schemes can be constructed from just an injective OWF, although we will see a more efficient construction in the *random oracle model (ROM)*.

Similarly, CCA-security can be defined for PKE schemes by extending the corresponding IND-CPA(λ) game that was defined in Lecture 5, allowing the adversary to make decryption queries $\hat{c}_i \mapsto \mathsf{Dec}(1^\lambda, k, \hat{c}_i)$ to the challenger. Again, the PKE schemes we have seen are vulnerable to malleability attacks, hence are not CCA-secure.

However, Cramer and Shoup [CS98] have constructed a CCA-secure PKE scheme based on the DDH assumption, while Naor and Yung [NY90] have shown how to construct a CCA-secure PKE scheme from any CPA-secure PKE scheme, analogously to our Theorem 2.1 (although the primitives required are more sophisticated than the MAC scheme that we have used).

National University of Singapore
CS4230/CS5430
Foundations of Modern Cryptography
AY 2025/26, Sem 2

Lecture 9 (Mar 26, 2026)
Lecture: Rahul Jain
Notes: Courtesy many

Digital Signatures, Random Oracle Model, Hashing

Last lecture we talked about the task of ensuring the authenticity and integrity of messages in the symmetric key encryption setting. We defined message authentication codes (MAC) as a scheme which lets the sender, having access to the shared key, produce a valid tag for a message and given the message and its tag, the receiver, also knowing the key, can verify that the tag is valid. However, without knowing the key it is hard to produce a new message with a valid tag. We also constructed MAC from pseudorandom functions, and discussed the notion of chosen ciphertext security (CCA) of symmetric key encryption.

This lecture we shift our attention to the analogous task in the public-key setting. Namely, we talk about *digital signatures*, the public-key counterpart of MAC. In short, signature schemes enable signing the message with a secret *signing key*, while allowing verification of the signature (given the corresponding message) with a public *verification key* of the sender. We also cover the *random oracle model* (ROM), a useful theoretical construct for modeling idealized cryptographic properties of functions, and construct a digital signature scheme from trapdoor permutations (TDP) in this model. Finally, we formally define cryptographic *hash functions* and discuss some constructions.

1 Digital signatures

Suppose a sender has a unique and secret signing key she can use to sign messages, and the verification key is publicly available, so that anyone can verify her signatures against the received messages. A signature scheme is formally defined as follows.

Definition 1.1 (Signature schemes)**.** A *signature* scheme consists of three PPT algorithms:

- $\mathsf{KeyGen}(1^\lambda)$: outputs (sk, vk).
- $\mathsf{Sign}(1^\lambda, sk, m)$: outputs signature σ.
- $\mathsf{Verify}(1^\lambda, vk, m, \sigma)$: outputs 1 (accept) or 0 (reject).

A signature scheme must satisfy correctness and EUF-CMA security defined as follows.

Definition 1.2 (Correctness)**.** A signature scheme $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is correct if $\forall m \in M, \lambda \in \mathbb{N}$, with $(sk, vk) \leftarrow \mathsf{KeyGen}(1^\lambda)$, $\sigma \leftarrow \mathsf{Sign}(1^\lambda, sk, m)$,

$$\Pr\left[\mathsf{Verify}(1^\lambda, vk, m, \sigma) = 1\right] = 1.$$

For defining security consider the following game between Adv (adversary) and Cha (challenger).

EUF-CMA(λ) "Existential unforgeability under chosen message attack"

1. Cha starts by sampling the signing and verification keys $(sk, vk) \leftarrow \mathsf{KeyGen}(1^\lambda)$.
2. Cha sends vk to Adv.
3. For $i \in \{1, \ldots, \ell\}$ where $\ell \in \mathsf{poly}(\lambda)$:
   (a) Adv sends some $m_i \in M$ to Cha.
   (b) Cha responds with $\sigma_i = \mathsf{Sign}(1^\lambda, sk, m_i)$.
4. Adv produces some (m, σ) and sends it to Cha.

Adv wins EUF-CMA(λ) if:

- $\mathsf{Verify}(1^\lambda, vk, m, \sigma) = 1$.
- $\forall i \in \{1, \ldots, \ell\}$, $(m, \sigma) \neq (m_i, \sigma_i)$.

Note that Adv has to produce a new message and signature pair to win, otherwise the game could be won trivially by simply copying m_i for some $i \in \{1, \ldots, \ell\}$ and its signature received from Cha.

Definition 1.3 (EUF-CMA security)**.** A signature scheme $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is EUF-CMA secure if $\forall$ PPT algorithm $\mathcal{A}$,

$$\Pr[\mathcal{A} \text{ wins EUF-CMA}(\lambda)] = \text{negl}(\lambda).$$

2 Random oracle model (ROM)

In the random oracle model all parties have access to an oracle that computes a function

$$H : \{0,1\}^n \to \{0,1\}^m$$

for some $m, n \in \mathbb{N}$, where H is selected uniformly at random from all such functions. Everyone can query the oracle for $H(x)$ given input x, and no one knows the value $H(x)$ if they have not queried the oracle at x. The oracle can be thought of as returning a random output $y \in \{0,1\}^m$ upon being queried with previously unseen $x \in \{0,1\}^n$, setting $H(x) := y$, and returning $H(x)$ in case it has been queried with x before.

2.1 Applications

Although ROM is not necessarily available in the real world, it has properties that are very useful in modeling cryptographic systems, namely:

- One-wayness: The only way (except with a negligible probability) to know the preimage x of $H(x)$ is to have queried H on input x.

- Randomness.
- Publicly available oracle access.

In that sense, H is an idealized OWF, idealized source of uniform randomness, idealized *collision-resistant* hash function, and it can be used to model many other primitives. Therefore, many theoretical constructions can be first defined in ROM, and then have ROM replaced with its approximations, preserving security properties.

3 Signatures from TDP in ROM

How can we construct a signature scheme from TDP? Suppose we have a TDP where,

- $\mathsf{Gen}(1^\lambda)$: outputs (k, t).
- $\mathsf{Eval}(1^\lambda, k, x)$: evaluates $f_k(x)$.
- $\mathsf{Invert}(1^\lambda, t, y)$: evaluates $f_k^{-1}(y)$.

At the first glance one way to achieve this would be take k to be the verification key, and t to be the signing key, and sign messages by finding a preimage $f_k^{-1}(m)$ for a message m using the trapdoor t, and verify σ by checking if $f_k(\sigma) = m$ using the key k. However, such scheme cannot be EUF-CMA secure since the adversary could simply take an arbitrary σ not seen before and compute $f_k(\sigma) = m$, so that trivially σ is a valid signature for m. This brings us to the conclusion that in a signature scheme:

1. given a signature σ, producing a message m such that σ is a valid signature for m should be hard, and
2. given a message m, producing a signature σ such that σ is a valid signature for m should be hard.

The scheme described above violates (1).

To solve the above-mentioned problem we take advantage of ROM, and define a signature scheme in terms of a TDP and a blackbox H as follows.

Signatures from TDP in ROM

- $\mathsf{KeyGen}(1^\lambda)$: $(k, t) \leftarrow \mathsf{Gen}(1^\lambda)$.
- $\mathsf{Sign}(1^\lambda, t, m)$: output $f_k^{-1}(H(m))$.
- $\mathsf{Verify}(1^\lambda, k, m, \sigma)$: if $f_k(\sigma) = H(m)$ then output 1 else output 0.

Note that for this scheme the attack described above does not work, since even if we pick an arbitrary σ, and compute $f_k(\sigma)$, it will be random and it will be hard to find its preimage under H, unless the signature and the preimage were observed before. Now we would like to formally prove the correctness and security of this scheme.

Theorem 3.1. *The signature scheme constructed using TDP is correct and EUF-CMA secure.*

Proof. Correctness is straightforward as an honestly generated signature will always be accepted.

To show security, we will show how to use an adversary $\mathcal{A}$ that wins the EUF-CMA game to construct a $\mathcal{B}$ that inverts the TDP.

Note that $\mathcal{A}$ may try to make queries to H during its execution. Hence $\mathcal{B}$, which is running the code of $\mathcal{A}$ and simulating the challenger, gets to decide what the responses to these queries are. Moreover $\mathcal{B}$ should ensure that the resulting game still looks identical to the EUF-CMA(λ) from $\mathcal{A}$'s point of view.

This is what the EUF-CMA(λ) in Definition 1.3 looks like for our scheme, note that both Cha and Adv have access to a random oracle H.

EUF-CMA(λ)

1. Cha generates $(k, t) \leftarrow \mathsf{Gen}(1^\lambda)$ and sends key k to Adv.
2. For $i \in [\ell]$ ($\ell \in \mathsf{poly}(\lambda)$):
   - Adv sends message m_i.
   - Cha responds with $\sigma_i \leftarrow \mathsf{Invert}(1^\lambda, t, H(m_i)) = f_k^{-1}(H(m_i))$.
3. Adv outputs (m, t).
4. Adv wins if
$$H(m) = \mathsf{Eval}(1^\lambda, k, \sigma) = f_k(\sigma)$$
and for all $i \in [\ell] : (m, t) \neq (m_i, t_i)$.

Note,

- The central observation behind the reduction is that $\mathcal{A}$, when successful, comes up with (m, σ) such that $H(m) = f_k(\sigma)$, or equivalently $\sigma = f_k^{-1}(H(m))$.
- Recall that to break the one-wayness of the TDP, $\mathcal{B}$ has to invert f_k at some random y it is given. If $\mathcal{B}$ can arrange things so that $H(m) = y$, then $\mathcal{A}$ would do the inversion for it.
- We will make one assumption of $\mathcal{A}$, that is at some point in its execution, it queries H at the m that it eventually outputs. This is easily arranged by modifying $\mathcal{A}$ appropriately. Further, oracle queries and sign queries can be done in an interleaving manner, but any sign query must be preceded by an oracle query for the same input.
- For simplicity, suppose $\mathcal{A}$ makes q queries in all to H, and never repeats a query. In fact, we can make this assumption because if $\mathcal{A}$ queries with the same input twice (or more) then $\mathcal{B}$ can get around by maintaining a lookup table for H to memorize all responses it has given to $\mathcal{A}$ so far.

Given key k and y, we construct $\mathcal{B} = \mathcal{B}(k, y)$ that acts as follows to invert y.

1. $\mathcal{B}$ samples $j^* \leftarrow [q]$
2. $\mathcal{B}$ sends key k to $\mathcal{A}$
3. **(Oracle queries)** For $j \in [q]$

(a) $\mathcal{A}$ queries the oracle H with the input z_j

(b) If $j = j^*$, $\mathcal{B}$ responds with y and sets $H(z_j) := y$

(c) Otherwise, $\mathcal{B}$ samples a random x_j and responds with $f_k(x_j)$ and sets $H(z_j) := f_k(x_j)$

4. **(Sign queries)** For $i \in [\ell]$

   (a) $\mathcal{A}$ sends message m_i and due to our assumption, there exists some $j \in [q]$ such that $m_i = z_j$

   (b) If $m_i = z_{j^*}$, $\mathcal{B}$ samples and responds with a random σ_i

   (c) Otherwise, $\mathcal{B}$ responds with $\sigma_i \leftarrow x_j$ (because $f_k^{-1}(H(z_j)) = f_k^{-1}(f_k(x_j)) = x_j$)

5. $\mathcal{A}$ outputs (m, σ)

6. $\mathcal{B}$ outputs σ

Let the game played above be game G. Let the random variables corresponding to the variables in G (for random (K, Y)) be represented by the same but capital letters. Let the game $\hat{G}$ be when $\mathcal{A}$ is playing a genuine EUF-CMA game and making ROM and sign queries. Let the random variables involved in $\hat{G}$ be represented with hat on top. Consider,

$$\begin{aligned}
\Pr[\mathcal{B} \text{ inverts } Y \text{ successfully}] &= \Pr[\mathcal{A} \text{ wins in } G \text{ and } M = Z_{J^*}] \\
&= \sum_{j=1}^{q} \Pr[\mathcal{A} \text{ wins in } G \text{ and } M = z_j \mid J^* = j] \cdot \Pr[J^* = j] \\
&= \sum_{j=1}^{q} \Pr\left[\mathcal{A} \text{ wins in } \hat{G} \text{ and } \hat{M} = z_j\right] \cdot \Pr[J^* = j] \\
&= \frac{1}{q} \sum_{j=1}^{q} \Pr\left[\mathcal{A} \text{ wins in } \hat{G} \text{ and } \hat{M} = z_j\right] \\
&= \Pr\left[\mathcal{A} \text{ wins in } \hat{G}\right] \cdot \frac{1}{q},
\end{aligned}$$

which is noticeable if $\mathcal{A}$ wins with noticeable probability in $\hat{G}$ (and since $q \in \mathsf{poly}(\lambda)$). The third equality above follows since for any j, when $M = z_j$ and $J^* = j$ in G,

1. Y is identically distributed to $f_K(X'_j)$ for any $j' \neq j$.

2. z_j is not a sign query in G.

3. Hence Step 4(b) is not invoked which is a key difference between G and $\hat{G}$.

4. For any fixed k, the replies (by $\mathcal{B}$) to the oracle queries (by $\mathcal{A}$) in G are uniformly distributed (since f_k is a permutation) and independent (since for each query a random and independent x_j is chosen in Step 3(c)).

5. Hence G and $\hat{G}$ look identical to $\mathcal{A}$. □

Remark: While signatures appear like a public-key primitive and this construction uses TDPs, which also gives public-key encryption, they can actually be constructed from one-way functions (OWF) [NY89b], though the construction is not efficient. More commonly used schemes are based on assumptions related to discrete log and Decision Diffie–Hellman (DDH) (see Digital Signature Algorithm (DSA) [Sch96], Schnorr signatures [Seu12], ElGamal signatures [Gam84]), and also use a cryptographic hash function. There are also many variants with additional or different properties like group signatures, ring signatures, blind signatures, threshold signatures, etc.

4 Hashing

Consider a scenario (described in [NY89b]) where users share computer programs that reside in the common (read/write) space. We assume that a small read-only area is available in the shared memory. To prevent computer viruses from modifying the programs, the users would like to authenticate the programs before their use. A possible way to do it is to use the read-only area as a common security server that stores the hash value of the files of the common space, where the hash function itself is publicly known and can be stored in the read-only space as well.

A property that is commonly desired of such cryptographic hash functions is that it is (computationally) hard to find two inputs that map to the same output even though many such pairs may exist. This property is called *collision resistance*.

4.1 Collision-resistant hash functions

Definition 4.1 (Collision-resistant hash family)**.** A *collision-resistant hash family* (CRHF) is a family of functions $H_\lambda = \{h_k : \{0,1\}^{n(\lambda)} \to \{0,1\}^{m(\lambda)}\}$ defined by the following algorithms:

- $\mathsf{Gen}(1^\lambda)$: outputs hash key k. This is a PPT algorithm.
- $\mathsf{Eval}(1^\lambda, k, x)$: outputs evaluation $y = h_k(x)$. This is a PT algorithm.

We want a CRHF to have two properties:

- **Shrinkage:** $\forall\lambda : m(\lambda) < n(\lambda)$.

  This property is what makes hashes useful as digests. The more shrinking the function is, the better, though too much shrinking would contradict the next property.

- **Collision-resistance:** For any PPT algorithm $\mathcal{A}$,

$$\Pr_{k\leftarrow\mathsf{Gen}(1^\lambda),(x,y)\leftarrow\mathcal{A}(1^\lambda,k)}[x \neq y \wedge h_k(x) = h_k(y)] = \mathrm{negl}(\lambda).$$

4.2 CRHF from discrete log

We assume that discrete log is hard in a prime-order group G_λ (such as the group of quadratic residues (QR) modulo a safe prime). We will need average-case hardness similar to the assumption from Lecture 6 and Lecture 7.

Definition 4.2 (Discrete log hash family)**.** A *discrete log hash family* (DLHF) is a family of functions $H_\lambda = \{h_k : \{0,1\}^{n(\lambda)} \to \{0,1\}^{m(\lambda)}\}$ defined by the following PPT algorithms:

- $\mathsf{Gen}(1^\lambda)$: samples random generator $g \leftarrow \mathsf{Gen}(G_\lambda)$ and $x \leftarrow \mathbb{Z}_{|G_\lambda|}$ to output the hash key (g, g^x).
- $\mathsf{Eval}(1^\lambda, (g, g^x), (y_0, y_1))$: outputs $g^{y_0 + x \cdot y_1} \in \mathbb{Z}_{|G_\lambda|}$ for $y_0, y_1 \in \mathbb{Z}_{|G_\lambda|}$.

Theorem 4.1. *A DLHF is a CRHF.*

Proof. We show that DLHF satisfies the two desired properties.

- **Shrinkage:** The output is half the input length.
- **Collision-resistance:** Given a collision for any key (g, g^x), we will show how to recover x. The hardness of discrete log then implies collision resistance.

Suppose we know a collision $(y_0, y_1), (z_0, z_1)$ for key (g, g^x), that is

$$(y_0, y_1) \neq (z_0, z_1) \text{ and } g^{y_0 + x \cdot y_1} = g^{z_0 + x \cdot z_1}$$

which implies (since g is a generator)

$$y_0 + x \cdot y_1 = z_0 + x \cdot z_1 \mod |G_\lambda|.$$

1. If $y_1 = z_1$, then $y_0 = z_0$, which is a contradiction.
2. If $y_1 \neq z_1$, then $x = \frac{z_0 - y_0}{y_1 - z_1} \mod |G_\lambda|$.

Note $|G_\lambda|$ is a prime and hence the inverse $\frac{1}{y_1 - z_1}$ always exists, which gives us the discrete log. □

CRHFs can be constructed from a few other assumptions too, but we don't know how to get them from OWFs (and there are some known separations).

In practice, fixed hash functions are used instead of a family of keyed functions, and often a random *initialization vector* (IV) is used that can play part of the role of a key. It turns out that any shrinkage at all can be built upon to get a lot of shrinkage. This is quite important in practice when one might want to hash large files.

National University of Singapore
CS4230/CS5430
Foundations of Modern Cryptography
AY 2025/26, Sem 2

Lecture 10 (April 9, 2026)
Lecture: Rahul Jain
Notes: Courtesy many

Multiparty Computation, Oblivious Transfer

In the previous lecture, we explored signatures and their construction in the random oracle model (ROM). In this lecture, we study *multiparty computation* (MPC) that is perfectly secure against *semi-honest* adversaries. We discuss and use *Shamir's secret sharing scheme* for this purpose. We prove the existence of such an MPC protocol for any function and show the limits of statistical security. We then relax our notion to obtain a security model for these cases similar to how we relaxed perfect indistinguishability to obtain computational indistinguishability. Lastly, we examine how to use the *oblivious transfer* protocol to construct a secure MPC protocol and give an outlook toward malicious adversaries.

1 Introduction

Suppose all CS5430 students want to jointly find out the average grade for the midterm. Can they do this in a way where they do not have to trust a third-party, and also do not have to reveal their grades? In more general terms, n parties $P_1, ..., P_n$ are interested in jointly computing a function $f(x_1, ..., x_n)$ where x_i denotes some private information held by P_i. The central question now is how to model *no party learns anything* in a formal way. We will explore this in today's lecture covering multiparty computation protocols.

1.1 A simple example

Let us first consider an example where three parties P_a, P_b, P_c are interested in learning the output bit of the following function $f(a, b, c) = a \oplus b \oplus c$ without revealing their private bits a, b, c to each other.

Approach 1

1. P_a samples a random bit $r \leftarrow \{0, 1\}$ and sends $a \oplus r$ to P_b.
2. P_b sends $a \oplus r \oplus b$ to P_c.
3. P_c sends $a \oplus r \oplus b \oplus c$ to P_a.

Party P_a can then obtain $a \oplus b \oplus c = f(a, b, c)$ by XORing the result with r one more time. From the properties of XOR, it is clear that this scheme is correct. An individual party also does not learn anything about the private information of their peers, e.g. $a \oplus r$ looks completely random to P_b. However, two colluding parties, e.g. P_a and P_b could learn P_c's private bit c once they know

$f(a, b, c)$, b, and a. Note that in this case this is fine for them to know, as the output of the function together with their inputs determines c, and they are allowed to know the output and their own inputs.

Approach 2
We also present a second approach to solving this problem. While this involves a few extra steps, it will give us useful directions for studying more complicated questions later on.

1. Each P_x with $x \in \{a, b, c\}$ splits their private bit x into x_a, x_b, x_c s.t. $x = x_a \oplus x_b \oplus x_c$.
2. Each P_x sends x_y to P_y with $x \neq y \in \{a, b, c\}$ s.t. every party possesses their private bit and one share for every other party.
3. Everyone combines all the shares they got, e.g. P_a computes $r_a = a_a \oplus b_a \oplus c_a$ and sends the result to P_b and P_c.
4. To obtain the result everyone computes $f(a, b, c) = r_a \oplus r_b \oplus r_c$.

The correctness of this protocol can be verified by exploiting commutativity of the exclusive OR (XOR). Shares of other parties that one party holds look random to them and they cannot learn anything more from that than what $f(a, b, c)$ revealed. However, just as in the first approach, P_b and P_c can determine P_a's private bit. The process of splitting private information into shares will later be of interesting use.

2 Multiparty computation (MPC)

We present a formalization for the correctness and security of an MPC scheme along the lines of **perfect statistical security**. This kind of security definition is different from our previous notions, so we first try to capture formally what it means for parties to not learn anything about other parties' private information besides what they already know from the function output. For this, we define the view of one or multiple parties as all of their inputs, and private randomness as well as all the messages they exchange.

Definition 2.1 (View of one party)**.** The $View$ of a party P_i with input x_i, private randomness r_i in protocol Π is defined as:

$$View_{P_i}(\Pi(x_1, ..., x_n)) = (x_i, r_i, M_i)$$

where M_i is the set of messages sent and received by P_i.

Definition 2.2 (View of multiple parties)**.** The $View$ of multiple parties $S \subseteq \{P_1, ..., P_n\}$ is defined as:

$$View_S(\Pi(x_1, ..., x_n)) = \{View_{P_i}\}_{P_i \in S}.$$

Definition 2.3 (Perfect t-secure MPC scheme Π)**.** A protocol that computes the function f for n parties $P = \{P_1, ..., P_n\}$ with every party P_i holding the value of x_i is a perfectly t-secure MPC protocol Π against t colluding parties if it satisfies:

- **Correctness** For all $(x_1, ..., x_n)$, at the end of the protocol Π it holds that every party $P_i \in \{P_1, ..., P_n\}$ can learn $f(x_1, ..., x_n)$.
- **Security** There exists a simulator Sim s.t. for all $S \subseteq \{P_1, ..., P_n\}$ with $|S| \leq t$:

$$Sim(\{x_i\}_{p_i \in S}, f(x_1, ..., x_n)) \equiv View_S(\Pi(x_1, ..., x_n)).$$

Intuitively, this can be explained as anything that the adversarial parties learn during the protocol being identically distributed to something the parties can learn on their own using just their inputs and the output of the function. We assume that all parties, including the adversaries, are semi-honest, that is they follow the protocol.

Definition 2.4 (Semi-honest adversary)**.** An adversarial party that correctly follows the protocol, but wants to learn as much as possible about the other parties' inputs is referred to as *an honest but curious* or a semi-honest adversary.

We will first discuss Shamir's secret sharing scheme (SSS) which we use for constructing an MPC scheme.

2.1 Shamir's secret sharing scheme (SSS) [Sha79]

We start by discussing *Lagrange interpolation.*

Theorem 2.1 (Lagrange interpolation)**.** *Let $\mathbb{F}$ be a field and let*

$$(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k) \in \mathbb{F} \times \mathbb{F}$$

be k points with distinct $x_1, \ldots, x_k$. There exists a unique *polynomial $f(x)$ of degree at most $k-1$ such that*

$$f(x_i) = y_i \quad \text{for all } i = 1, \ldots, k.$$

Proof. For each $i \in \{1, \ldots, k\}$, define the *Lagrange basis polynomial*

$$\ell_i(x) = \prod_{\substack{1 \leq j \leq k \\ j \neq i}} \frac{x - x_j}{x_i - x_j}.$$

Each $\ell_i(x)$ is a polynomial of degree $k-1$ and satisfies

$$\ell_i(x_j) = \begin{cases} 1, & \text{if } j = i, \\ 0, & \text{if } j \neq i. \end{cases}$$

Now define

$$f(x) = \sum_{i=1}^{k} y_i \ell_i(x).$$

Since f is a linear combination of polynomials of degree at most $k-1$, we have $\deg f \leq k-1$. Moreover, for each $j \in \{1, \ldots, k\}$,

$$f(x_j) = \sum_{i=1}^{k} y_i \ell_i(x_j) = y_j,$$

so f interpolates the given points.

To prove uniqueness, suppose $g(x)$ is another polynomial of degree at most $k-1$ with $g(x_i) = y_i$ for all i. Then the polynomial $h(x) = f(x) - g(x)$ has degree at most $k-1$ and satisfies

$$h(x_i) = 0 \quad \text{for all } i = 1, \ldots, k.$$

Thus $h(x)$ has k distinct roots. Since a nonzero polynomial of degree at most $k-1$ over a field can have at most $k-1$ roots, it follows that $h(x) \equiv 0$, and hence $f = g$. Therefore, the interpolating polynomial is unique. □

Definition 2.5 ((k,n)-threshold secret sharing)**.** Let $\mathbb{F}$ be a finite field and let $n, k \in \mathbb{N}$ with $1 \le k \le n$. A (k,n)-threshold secret sharing scheme distributes a secret $s \in \mathbb{F}$ among n participants such that:

- any set of at least k shares uniquely determines s, and
- any set of fewer than k shares reveals no information about s.

Setup. Choose a prime power $|\mathbb{F}| > n$. Let the secret be $s \in \mathbb{F}$. Select coefficients $a_1, \ldots, a_{k-1} \leftarrow \mathbb{F}$ uniformly at random and define the polynomial

$$f(x) = s + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1}.$$

Fix distinct, nonzero $x_1, \ldots, x_n \in \mathbb{F}$.

Share generation. For each participant $i \in \{1, \ldots, n\}$, output the share

$$\mathsf{Share}_i \stackrel{\text{def}}{=} [s]_i = (x_i, f(x_i)).$$

Reconstruction. Given any k shares $\{(x_{i_1}, y_{i_1}), \ldots, (x_{i_k}, y_{i_k})\}$, reconstruct f by Lagrange interpolation and output the secret

$$s = f(0) = \sum_{j=1}^{k} y_{i_j} \prod_{\substack{1 \le \ell \le k \\ \ell \ne j}} \frac{-x_{i_\ell}}{x_{i_j} - x_{i_\ell}}.$$

Proposition 2.2 (Correctness)**.** *Any set of at least k shares uniquely reconstructs the secret s.*

Proof. The shares are evaluations of a polynomial $f(x)$ of degree at most $k-1$. By Lagrange interpolation (Theorem 2.1), any k distinct points determine f uniquely. Hence $f(0) = s$ is uniquely recovered. □

Proposition 2.3 (Perfect secrecy)**.** *Any coalition of fewer than k participants obtains no information about the secret s.*

Proof. Fix any set of distinct $k-1$ shares. For every possible secret $s' \in \mathbb{F}$, there exists exactly one polynomial of degree $k-1$ whose constant term is s' and which is consistent with the given $k-1$ shares. Since the coefficients $a_1, \ldots, a_{k-1}$ were chosen uniformly at random, all secrets $s' \in \mathbb{F}$ are equally likely conditioned on these shares. Therefore the shares give no information about s. □

Remark. SSS is linear: the share of a sum of secrets equals the sum of the shares.

2.2 Perfect MPC for $t < \frac{n}{2}$

We now show that we can achieve perfect security under this model for any arbitrary function. In particular, it turns out that this threshold bound is tight, or in other words the best we can do without relaxing our security notation for any arbitrary function.

Theorem 2.4 (Completeness theorem for semi-honest MPC)**.** *For every function $f : \mathbb{F}^n \to \mathbb{F} \in F_{all}$ and $t < \frac{n}{2}$, there exists a perfect t-secure (semi-honest) MPC scheme Π.*

We first present the BGW protocol due to Ben-Or, Goldwasser, and Wigderson [BOGW88], and argue its correctness. We will then use this protocol to prove Theorem 2.4.

2.2.1 The BGW protocol

The intuition behind the BGW protocol is the following. We are expanding on an idea already used in the introduction's example. We will now use SSS over a sufficiently large field $\mathbb{F}$ to share private information. We model the function f as an *arithmetic circuit* over the field $\mathbb{F}$. The circuit is computed by the parties gate by gate using the homomorphisms of SSS.

1. **Secret sharing**

   (a) Each party P_i splits their input x_i into n parts $[x_i]_1, ..., [x_i]_n$, using $(t+1, n)$-SSS.

   (b) Share $[x_i]_j$ is sent to party P_j resulting in everyone possessing one share of each other party's input.

2. **Addition gate**

   Denote the inputs of the addition gate as y and z, and the output as w. The parties already hold shares for y and z. Namely, the party P_i holds $[y]_i$ and $[z]_i$. Each party then locally computes the addition of two shares $[y]_i + [z]_i = [y+z]_i = [w]_i$.

   We now show that this works and is a correct sharing with the fact that the parties used SSS. There exists a polynomial p with $\deg(p) \leq t$ s.t. $p(0) = y$ and $p(i) = [y]_i$. Similarly, there exists a polynomial q with $deg(q) \leq t$ s.t. $q(0) = z$ and $q(i) = [z]_i$. Note $\deg(p+q) \leq t$. From the homomorphism properties of polynomials it follows that

   $$(p+q)(0) = p(0) + q(0) = y + z = w,$$

   and also

   $$(p+q)(i) = p(i) + q(i) = [y]_i + [z]_i = [w]_i.$$

3. **Multiplication gate**

   The multiplication case turns out to be more intricate compared to its addition counterpart. This is due to the multiplication of two polynomials increasing the degree of the product of these polynomials. Let p, q be the polynomials used to share y, z respectively. The goal now is to find a polynomial r s.t. $r(0) = w \stackrel{\text{def}}{=} yz$ and $\forall i \in [n] : r(i) = [w]_i$ and importantly $\deg(r) \leq t$.

We can reduce the degree of $r = p \times q$ by using Langrange interpolation (Theorem 2.1). From the theorem we get that for all polynomials h with $\deg(h) < n$, we have

$$h(0) = \sum_{j=1}^{n} L_j h(j), \text{ where } \forall j \in [n] : L_j = \prod_{\ell \neq j} \frac{-\ell}{j - \ell}.$$

Every party P_i uses this to:

(a) Compute $v_i = [y]_i \times [z]_i$.

(b) Share v_i into $[v_i]_1, ..., [v_i]_n$, say using (random) degree t polynomial r_i, such that

$$\forall i, j \in [n] : r_i(0) = v_i \quad ; \quad r_i(j) = [v_i]_j.$$

Send $[v_i]_j$ to P_j and receive $[v_j]_i$ from each P_j.

(c) Set $[w]_i = \sum_{j=1}^{n} L_j [v_j]_i$.

Consider the polynomial $r \stackrel{\text{def}}{=} \sum_{j=1}^{n} L_j r_j$. Since each r_j is degree t, $\deg(r) \leq t$. Now,

$$\begin{aligned} r(0) &= \sum_{j=1}^{n} L_j r_j(0) = \sum_{j=1}^{n} L_j v_j = \sum_{j=1}^{n} L_j \times [y]_j \times [z]_j \\ &= \sum_{j=1}^{n} L_j \times p(j) \times q(j) = \sum_{j=1}^{n} L_j \times pq(j) = pq(0) = p(0)q(0) = yz = w. \end{aligned}$$

Above pq represents the product of polynomials p and q. Note that $\deg(pq) \leq 2t < n$, hence the last equality follows from Lagrange interpolation (Theorem 2.1). Similarly,

$$[w]_i = \sum_{j=1}^{n} L_j [v_j]_i = \sum_{j=1}^{n} L_j r_j(i) = \left(\sum_{j=1}^{n} L_j r_j \right)(i) = r(i).$$

4. **Computing output**

Finally, after each party P_i has computed their share $[w]_i$ for the output gate of the circuit, they share it with every other party and everyone can then reconstruct the output.

The correctness of the protocol follows from the descriptions above. It now remains to prove the security of the BGW protocol. We will show this by simulating the views of any set of parties smaller than t. We will now construct a simulator bottom-up that is run by the semi-honest adversarial parties $S = \{P_1, ..., P_t\}$. Observe that since $|S| = t$, shares of elements that are not already known to the adversarial parties look uniformly random to them.

Proof Sketch of Theorem 2.4. Suppose the adversarial parties are $S = \{P_1, \ldots, P_t\}$. After the first step of the BGW protocol, secret sharing, the adversarial parties know

$$\{(x_1 = ([x_1]_1, ..., [x_1]_n)), ..., (x_t = ([x_t]_1, ..., [x_t]_n))\}$$

(their own inputs) as well as the following shares of the inputs of non-adversarial parties:

$$\{[x_{t+1}]_1, ..., [x_{t+1}]_t, ..., [x_n]_1, ..., [x_n]_t\}.$$

All the shares they hold for other parties look completely random to them by the threshold $t+1$ of SSS. That way a simulator can simulate the view as follows given all adversarial parties $P_i \in S$ and their random input x_i:

1. Compute valid shares for x_i as $[x_i]_1$ to $[x_i]_n$ using SSS.
2. Uniformly sample the shares the non-adversarial parties sent to P_i since they appear random in the protocol.

Now, what the simulator has simulated given all the inputs of the adversarial parties is identically distributed to what the adversarial parties learned through the protocol.

Suppose without loss of generality that the first evaluated gate is addition. Then, the simulator simply adds all individual shares. This works since from the first step of the protocol the simulation is identically distributed to the view of all the parties and adding shares will not change anything in that regards.

Suppose that the next evaluated gate is a multiplication. The simulator works analogously to the addition case.

Finally, the shares $\{w_i\}_{i=1}^n$ of the output gate can be simulated correctly as the simulator is given the output w_f of the function. The first t shares $\{w_i\}_{i=1}^t$ will be uniform and independent. Simulator can then find the unique polynomial r with $\deg(r) \leq t$ such that $r(0) = w_f$ and $\forall i \in [t] : r(i) = w_i$. From here the rest of the shares can be found as $\{w_i := r(i)\}_{i=t+1}^n$.

Therefore, the output of the simulator and the views are identically distributed. By our security definition, the BGW protocol is secure against semi-honest adversaries. □

It turns out that if $t \geq n/2$, even some simple functions cannot be computed in a perfect secure MPC protocol. We will now prove this by stating such a function AND.

Theorem 2.5 (Impossibility of perfect security for $t \geq \frac{n}{2}$)**.** *Let $t \geq \frac{n}{2}$. There does not exist a perfect t-secure MPC scheme Π for every function $f : \mathbb{F}^n \to \mathbb{F} \in F_{all}$.*

Proof Sketch of Theorem 2.5. Let us assume that the two parties Alice and Bob are trying to compute the AND of their input bits X_A and X_B respectively, using a communication protocol Π. Let the transcript of the messages exchanged between them in Π be represented by the distribution $\Pi(X_A, X_B)$.

From (perfect) correctness we have that when $X_A = 1$, Alice should learn X_B perfectly. Therefore, the distributions $\Pi(1,0)$ and $\Pi(1,1)$ should have disjoint supports. Similarly $\Pi(0,1)$ and $\Pi(1,1)$ should have disjoint supports.

From (perfect) secrecy, when $X_A = 0$, Alice should learn nothing about X_B. Hence $\Pi(0,0) = \Pi(0,1)$. Similarly $\Pi(0,0) = \Pi(1,0)$.

From above we get that $\Pi(0,0)$ and $\Pi(1,1)$ have disjoint supports.

However, in a communication protocol Π, if $\Pi(0,0)$ and $\Pi(1,1)$ have disjoint supports, then $\Pi(1,0)$ and $\Pi(0,1)$ must also have disjoint supports (why? show this as homework).

This is a contradiction.

□

It is important to note that B's strategy above is not necessarily computationally efficient. In fact, with some computational assumptions, we can get a protocol for AND with computational security. Nonetheless, the impossibility of achieving perfect security when the majority of parties are adversarial remains.

3 Oblivious transfer

Definition 3.1. (Oblivious transfer) An oblivious transfer (OT) protocol is a cryptographic protocol between a sender S and a receiver R, where S holds two bits (x_0, x_1), and R holds one bit b. At the end of the protocol, R learns x_b but nothing about x_{1-b}, and S learns nothing about b.

3.1 Two-party AND from OT

An OT protocol can be used to construct a two-party MPC protocol for $f(x_A, x_B) = x_A \wedge x_B$, where two parties A and B each have a bit x_A and x_B, respectively, and wish to compute the bit-wise AND of their inputs. Party A takes on the sender's role in the OT protocol and sets $x_0 = 0$ and $x_1 = x_A$. Party B plays the receiver with $b = x_B$. After the OT protocol, party B learns only $x_A \wedge x_B$. For $x_B = 1$, A returns x_1, and $x_1 = x_A \wedge x_B$ holds by definition. For $x_B = 0$, A returns $x_0 = 0 = x_A \wedge x_B$. Hence party B only learns $x_A \wedge x_B$ by the properties of the OT protocol.

3.2 Secure MPC protocol

Oblivious transfer can be used to construct a computationally t-secure MPC protocol that is secure against even $t = (n-1)$ adversarial (semi-honest) parties colluding with each other, as shown by Goldreich-Micali-Wigderson [GMW91], as long as parties obey the protocol.

3.3 Malicious adversaries

Till now, we only considered semi-honest adversaries, adversaries that obey the protocol. For malicious adversaries which deviate from the protocol, it is more difficult to define security. However many of these notions can still be guaranteed assuming the OT functionality. Furthermore, we also know how to transform any protocol that is secure against a semi-honest adversary into one that protects against a malicious adversary. The idea behind this is that all parties run an instance of the semi-honest protocol as well as a *zero-knowledge proof* verifying that they ran the protocol correctly. More details on this can be found in the GMW compiler paper [GMW91].

National University of Singapore
CS4230/CS5430
Foundations of Modern Cryptography
AY 2025/26, Sem 2

Lecture 11 (Apr 16, 2026)
Lecture: Rahul Jain
Notes: Courtesy many

Lattice-Based Cryptography

---

1 Introduction

Lattice-based cryptography is a branch of constructions of cryptographic primitives that has gained significant attention in recent years due to widely believed assumptions of its strong security properties and resistance against quantum computing attacks. It relies on the hardness of several mathematical problems under lattice structure – regular, discrete, and infinite arrangements of points in n-dimensional Euclidean space – to construct cryptographic primitives such as encryption schemes, digital signatures, and key exchange protocols.

The advent of quantum computers and Shor's Algorithm [Sho94] pose a significant threat to the security of traditional cryptographic systems, such as RSA, Diffie-Hellman and elliptic curve cryptosystems (ECC), which rely on the computational intractability of problems like integer factorization and discrete logarithms. Lattice-based cryptography offers an attractive alternative, as it is believed to be resistant to attacks by both classical and quantum computers due to the hardness of the underlying problems it is based on, such as the Shortest Vector Problem (SVP) and the Learning With Errors (LWE) problem. Additionally, lattice-based schemes tend to have efficient and parallelizable algorithms, making them suitable for practical applications.

2 Lattice

A lattice is a discrete subgroup of an n-dimensional Euclidean space, denoted as $\mathbb{R}^n$. Formally, a lattice can be defined as follows:

Definition 2.1 (Lattice)**.** Let $\mathbf{B} = \{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_n}\}$ be a set of n linearly independent vectors in $\mathbb{R}^n$. The lattice $L(\mathbf{B})$ generated by the basis $\mathbf{B}$ is the set of all integer linear combinations of the basis vectors $\{\mathbf{b_i}\}$:

$$L(\mathbf{B}) = \left\{ \sum_{i=1}^{n} a_i \mathbf{b_i} \mid a_i \in \mathbb{Z} \right\}.$$

In this definition, the set $\mathbf{B}$ is called the basis of the lattice $L(\mathbf{B})$, and the linearly independent vectors $\{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_n}\}$ are referred to as the basis vectors. Note that a lattice can have multiple bases, as the basis vectors can be linear combinations of each other.

The dimension of a lattice is the number of its linearly independent basis vectors, which is also equal to the dimension of the Euclidean space it is embedded in. In the case of Fig. 3, we had a two-dimensional lattice, as it was defined by two linearly independent basis vectors.

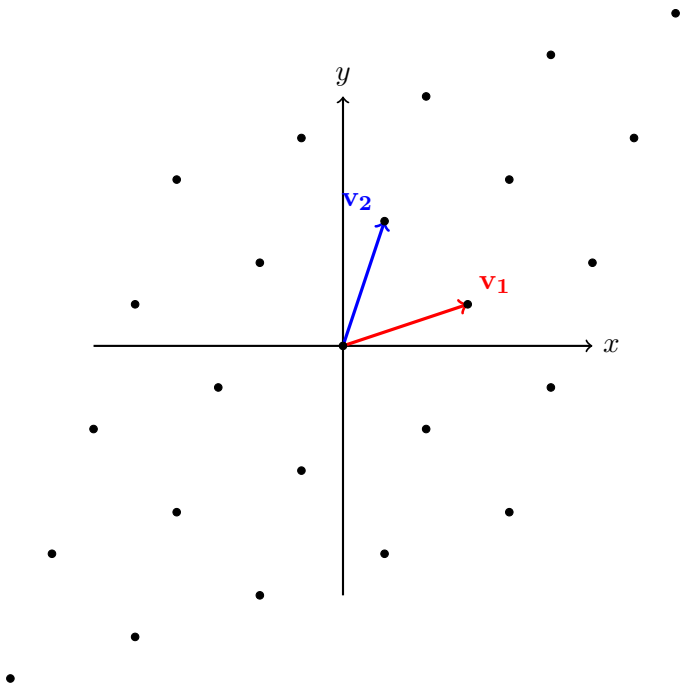


Figure 3: A simple two-dimensional lattice generated by the basis vectors $\mathbf{v_1}$ and $\mathbf{v_2}$.

A significant property of lattices is their periodicity. Due to the linear combination of integer coefficients, lattices exhibit a regular and repeating pattern. The spacing and orientation of this pattern are determined by the basis vectors of the lattice.

3 Hard Lattice Problems

3.1 Shortest Vector Problem (SVP)

Definition 3.1 (Shortest Vector Problem)**.** Given a lattice $L(\mathbf{B})$ generated by a basis $\mathbf{B}$, the Shortest Vector Problem (SVP) is to find a non-zero lattice vector $\mathbf{v} \in L(\mathbf{B})$ with the smallest Euclidean norm:

$$\mathbf{v} = \arg \min_{\mathbf{x} \in L(\mathbf{B}), \mathbf{x} \neq \mathbf{0}} \|\mathbf{x}\|_2.$$

3.2 Closest Vector Problem (CVP)

Definition 3.2 (Closest Vector Problem)**.** Given a lattice $L(\mathbf{B})$ generated by a basis $\mathbf{B}$ and a target vector $\mathbf{t} \in \mathbb{R}^n$, the Closest Vector Problem (CVP) is to find a lattice vector $\mathbf{v} \in L(\mathbf{B})$ that minimizes the Euclidean distance to the target vector:

$$\mathbf{v} = \arg \min_{\mathbf{x} \in L(\mathbf{B})} \|\mathbf{x} - \mathbf{t}\|_2.$$

These two problems, SVP and CVP, are assumed to be computationally hard in the worst-case, even for quantum computers, which makes them an appropriate foundation for the security of lattice-based cryptographic schemes.

3.3 Reduction between SVP and CVP (Optional)

Some works have demonstrated the relationship between the Shortest Vector Problem (SVP) and the Closest Vector Problem (CVP), establishing reductions between these problems in various l_p norms (SVP_p and CVP_p). More details can be found in the [ACK+21], which presents several $2^{\epsilon m}$-time reductions for $1 \leq p \leq q \leq \infty$. More specifically, the reductions between the following approximate version of SVP and CVP are showcased, which increase the rank n and dimension m of the input lattice by at most one.

1. A reduction from $\tilde{O}(1/\varepsilon^{1/p})\gamma$-approximate SVP_q to γ-approximate SVP_p.
2. A reduction from $\tilde{O}(1/\varepsilon^{1/p})\gamma$-approximate CVP_p to γ-approximate CVP_q.
3. A reduction from $\tilde{O}(1/\varepsilon^{1+1/p})$-$\text{CVP}_q$ to $(1+\varepsilon)$-unique SVP_p (which in turn trivially reduces to $(1+\varepsilon)$-approximate SVP_p).

These results provide a better understanding of the relationship between SVP and CVP, demonstrating their interconnections in various l_p norms. The paper builds upon previous work by combining techniques from Eisenbrand and Venzin's breakthrough research [EV20] with sparsification-based techniques.

4 Learning with Errors

The Learning With Errors (LWE) problem is a fundamental cryptographic problem that is at the core of various lattice-based cryptographic primitives. LWE is believed to be hard to solve, even for quantum computers, which makes several LWE-based constructions attractive candidates for post-quantum cryptography.

Definition 4.1 (Learning With Errors)**.** The Learning With Errors (LWE) problem is defined as follows. Let $n, m, q \in \mathbb{N}$ with $m = \mathsf{poly}(n)$, $m \gg n$ and q being a prime number, $q = \mathsf{poly}(n)$, and let χ be a probability distribution on $\mathbb{Z}_q$. We are given a matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ and a vector $\mathbf{b} \in \mathbb{Z}_q^m$. The LWE problem asks to find a secret vector $\mathbf{s} \in \mathbb{Z}_q^n$ such that:

$$\mathbf{b} \approx \mathbf{A}\mathbf{s} \pmod{q},$$

where the approximation is defined with respect to the noise distribution χ. Specifically, there exists an error vector $\mathbf{e} \in \mathbb{Z}_q^m$ sampled from χ^m such that:

$$\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \pmod{q}.$$

This definition captures the most general form of LWE. In the lecture, χ is simplified to a uniform distribution over $[-\eta \cdot q, \eta \cdot q]$, where $\eta \in [0, 1]$ is the noise magnitude. In other literature, instead of η, the domain is directly determined by a parameter $B < \frac{q}{4}$ i.e. $[-B, B]$.

We can easily see that if we remove any ingredient in this structure, the problem will become easy. For example, if the noise e is not added, $\mathbf{s}$ can be found by Gaussian elimination. If $\mathbf{b}$ is released without $\pmod{q}$, then we can solve it by *linear regression*. Magically, adding noise and $\pmod{q}$ together make the problem very hard.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \quad \mathbf{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} \quad \mathbf{e} = \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_m \end{pmatrix} \qquad \mathbf{b} = \mathbf{As} + \mathbf{e} \pmod{q}$$

Figure 4: Illustration of the LWE problem.

4.1 Hardness Assumption

The Learning with Errors (LWE) problem is assumed to be hard due to several reasons.

1. **Worst-case to average-case reduction:** LWE is considered hard because it has a strong connection to worst-case lattice problems, such as the Shortest Vector Problem (SVP) and the Closest Vector Problem (CVP). In his 2005 paper, Oded Regev introduced the LWE problem and showed that the hardness of solving LWE on average is at least as hard as solving worst-case lattice problems in the quantum setting [Reg09]. This worst-case to average-case reduction is a key factor in establishing LWE as a foundation for secure cryptographic schemes.

2. **Resilience to known attacks:** Another reason why LWE is assumed to be hard is that it has resisted all known attacks, including classical lattice reduction algorithms such as LLL [LLL82] and BKZ [SE94], as well as algorithms developed specifically for LWE, like the Blum-Kalai-Wasserman (BKW) algorithm [BKW03]. Even when considering the advent of quantum computing, LWE is believed to remain hard.

4.2 Setting Parameters

Over the past two decades, cryptanalysis has provided insights into safe parameter settings.

1. **Security parameter:** The security parameter n should be chosen between 1 and $10,000$, with larger values providing stronger security guarantees.

2. **Number of equations:** m can be an arbitrary polynomial in n.

3. **Error domain:** η should be chosen to make $\eta \cdot q$ as a small polynomial in n.

4. **Modulus:** The modulus, q, should be chosen as a polynomial in n and should be larger than B. It could be as large as a sub-exponential value, such as $2^{0.99n}$. Choosing a larger q can offer better security but may also affect the efficiency of the cryptographic scheme.

4.3 Search-LWE and Decisional-LWE

LWE has two main variants: the Search-LWE problem and the Decisional-LWE problem.

Definition 4.2 (Search-LWE problem)**.** Given a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ chosen uniformly at random and a noisy linear equation $\mathbf{b} = \mathbf{As} + \mathbf{e} \in \mathbb{Z}_q^m$, where $\mathbf{s} \in \mathbb{Z}_q^n$ is a secret vector and $\mathbf{e} \in \mathbb{Z}_q^m$ is an error vector with entries sampled from a noise distribution χ, the goal is to find the secret vector $\mathbf{s}$.

Definition 4.3 (Decisional-LWE problem)**.** Given a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ chosen uniformly at random, a noisy linear equation $\mathbf{b} = \mathbf{As} + \mathbf{e} \in \mathbb{Z}_q^m$ with the same notations as in the Search-LWE problem, and a uniformly random vector $\mathbf{u} \in \mathbb{Z}_q^m$, the goal is to distinguish between the distribution of $(\mathbf{A}, \mathbf{b})$ and the distribution of $(\mathbf{A}, \mathbf{u})$.

Assumption 4.1 (LWE Assumption)**.** *There exist polynomials $a(n)$ and $b(n)$ such that for all integers m that are polynomial in n, if $q > a(n)$ and $\eta = \frac{1}{b(n)}$, then for all PPT algorithms $\mathcal{A}$, we have:*

$$\Pr\left[\mathcal{A}(\mathbf{A}, \mathbf{As} + \mathbf{e}) = \mathbf{s}\right] = \mathrm{negl}(n),$$

where $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ is chosen uniformly at random, $\mathbf{s} \in \mathbb{Z}_q^n$ is the secret, and $\mathbf{e} \in \mathbb{Z}_q^m$ is a vector with entries chosen independently from a noise distribution with parameter η.

4.3.1 Reductions between SLWE and DLWE (Optional)

Now we discuss reductions between Search-LWE and Decisional-LWE.

Proposition 4.2 (Decision-to-Search reduction)**.** *It can be shown that if there exists an algorithm that solves the Search-LWE problem with noticeable advantage, then one can construct an algorithm to solve the Decisional-LWE problem. Given a search-LWE oracle, one can feed the oracle and obtain an $\mathbf{s}$ with $100n^3$ equations from the given LWE instance, and test $\mathbf{s}$ over other $100n^3$ equations to evaluate $\mathbf{s}$.*

Proposition 4.3 (Search-to-Decision reduction)**.** *Let the input be*

$$(\mathbf{a}_1, \langle \mathbf{a}_1, \mathbf{s} \rangle + e_1), ..., (\mathbf{a}_m, \langle \mathbf{a}_m, \mathbf{s} \rangle + e_m).$$

For each $i \in [n]$ (sequentially in the increasing order) do:

1. *Guess the i-th element of $\mathbf{s}$, $\mathbf{s}[i] = c_i$, where $c_i \leftarrow \mathbb{Z}_q$.*
2. *For each $j \in [m]$, define $\mathbf{a}'_j := \mathbf{a}_j + [0, 0, ..., r_i, ..., 0, 0]^T$, where $r_i \leftarrow \mathbb{Z}_q$ and*

$$b'_j := b_j + r_i \cdot c_i = \langle \mathbf{a}_j, \mathbf{s} \rangle + r_i \cdot c_i + e_j = \langle \mathbf{a}'_j, \mathbf{s} \rangle + e_j + r_i \cdot (c_i - \mathbf{s}[i]).$$

3. *Call DLWE with input $(\mathbf{A}', \mathbf{b}')$.*

Note,

- *If $c_i = \mathbf{s}[i]$, then $\forall j \in [m]$*

  $$b'_j = \langle \mathbf{a}'_j, \mathbf{s} \rangle + e_j.$$

  Hence $(\mathbf{A}', \mathbf{b}')$ is distributed according to $(\mathbf{A}', \mathbf{A}'\mathbf{s} + \mathbf{e})$.

- *If $c_i \neq \mathbf{s}[i]$, then $\forall j \in [m]$*

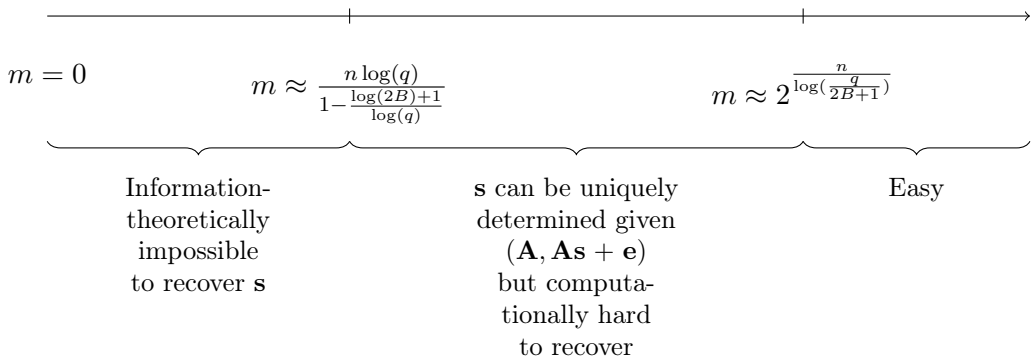
  $$b'_j = \langle \mathbf{a}'_j, \mathbf{s} \rangle + e_j + r_i \cdot (c_i - \mathbf{s}[i]).$$

  Since $(c_i - \mathbf{s}[i]) \in \mathbb{Z}_q^\star$, we get that $r_i(c_i - \mathbf{s}[i])$ is random in $\mathbb{Z}_q$ and hence b'_j is random in $\mathbb{Z}_q$. Hence $(\mathbf{A}', \mathbf{b}')$ is distributed according $(\mathbf{A}', \mathbf{u})$.

Hence the DLWE oracle helps to verify whether the guess is correct. There are total $n \cdot q$ calls to the DLWE oracle.

4.4 Information-Computation Gap

Given fixed and appropriate values of $n, q = \mathsf{poly}(n) > 4B, B = \mathsf{poly}(n) \approx \sqrt{n}$, the hardness of SLWE varies according to the value of m. Specifically, we have the following hardness regions.



4.5 One-Way Function Based on LWE

Clearly, we can build an OWF based on the assumption that SLWE is hard. Let (n, m, q, η) be the LWE parameters. We define a one-way function F as follows:

$$F(\mathbf{s}, \mathbf{e}) = (\mathbf{A}, \mathbf{As} + \mathbf{e}),$$

where

- $\mathbf{s} \in \mathbb{Z}_q^n$ is the input (secret key),
- $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ is chosen uniformly at random,
- $\mathbf{e} \in \mathbb{Z}_q^m$ is a noise vector with entries chosen independently from $[-\eta \cdot q, \eta \cdot q]$.

5 LWE-Based Secret-Key Encryption

In this section, we introduce a secret-key encryption scheme based on the Learning with Errors (LWE) problem. The construction relies on the hardness of LWE, and the scheme's security can be proven under the assumption that LWE is hard to solve.

Let's first show the formal definition of the LWE-based secret-key encryption scheme.

- **Key Generation** (Gen): Given security parameter n, a secret key $\mathbf{s} \in \mathbb{Z}_q^n$ is chosen uniformly at random.
- **Encryption** (Enc): To encrypt a one-bit message $c \in \{0, 1\}$, the sender does the following.
  1. Choose a random vector $\mathbf{a} \leftarrow \mathbb{Z}_q^n$.
  2. Choose a small noise term $e \leftarrow [-\eta \cdot q, \eta \cdot q]$ according to some noise distribution χ.
  3. Compute the ciphertext as $(\mathbf{a}, b = \langle \mathbf{a}, \mathbf{s} \rangle + e + m \cdot \frac{q+1}{2} \pmod q) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$.

- **Decryption** (Dec): To decrypt a ciphertext $(\mathbf{a}, b)$, the recipient computes $c' = b - \langle \mathbf{a}, \mathbf{s} \rangle$ and recovers the original message c by rounding c' to nearest point of $\{0 \text{ (or } q), q/2\}$ and dividing it by $q/2$.

5.1 Correctness

When the decryption is performed, we have

$$c' = b - \langle \mathbf{a}, \mathbf{s} \rangle = \left(\langle \mathbf{a}, \mathbf{s} \rangle + e + c \cdot \frac{q+1}{2} \right) - \langle \mathbf{a}, \mathbf{s} \rangle = e + c \cdot \frac{q+1}{2}.$$

Since e is small ($e < q/4$), rounding c' to the nearest multiple of $q/2$ and taking its remainder modulo q and dividing it by $q/2$ will recover the original bit c.

5.2 Security

The security of this LWE-based secret-key encryption scheme relies on the hardness of the LWE problem. Specifically, the semantic security of the scheme can be proven under the assumption that Decision-LWE is hard and then we use $\langle \mathbf{a}, \mathbf{s} \rangle + e$ as a one-time pad to encrypt the message. Based on the DLWE assumption, given an encryption of a random message, any efficient adversary cannot distinguish the encrypted message from a random element in $\mathbb{Z}_q^n \times \mathbb{Z}_q$.

6 Public-Key Encryption Scheme Based on LWE

6.1 Construction

We now describe a public-key encryption scheme based on LWE. Let (n, m, q, η) be the LWE parameters, and let $\mathcal{M} = \{0, 1\}$ be the message space.

- **Key Generation** (Gen):
  1. Choose a random matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$.
  2. Choose a random vector $\mathbf{s} \in \mathbb{Z}_q^n$.
  3. Choose a noise vector $\mathbf{e} \in \mathbb{Z}_q^m$ with entries chosen independently from $[-\eta \cdot q, \eta \cdot q]$.
  4. Compute $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e}$.
  5. The public key is $(\mathbf{A}, \mathbf{b})$ and the private key is $\mathbf{s}$.
- **Encryption** (Enc):
  1. Given a message $c \in \{0, 1\}$ and a public key $(\mathbf{A}, \mathbf{b})$, choose a random binary vector $\mathbf{r} \in \{0, 1\}^m$.
  2. Compute the ciphertext $\left(\mathbf{r}^T \mathbf{A}, \left(\mathbf{r}^T \mathbf{b} + m \cdot \lfloor \frac{q+1}{2} \rfloor \right) \mod q \right)$.
- **Decryption** (Dec):
  1. Given a ciphertext $(\mathbf{u}, v)$ and a private key $\mathbf{s}$, compute $c' = (v - \langle \mathbf{u}, \mathbf{s} \rangle) \mod q$.
  2. If $|c'| < \frac{q}{4}$ then output 0. Otherwise, output 1.

6.2 Correctness

We need to show that for any message $c \in \mathcal{M}$, we have

$$\mathsf{Dec}(\mathbf{s}, \mathsf{Enc}(\mathbf{A}, \mathbf{b}, c)) = c.$$

Recall that during encryption, we computed ciphertext

$$(\mathbf{u}, v) = \left(\mathbf{r}^T\mathbf{A}, \left(\mathbf{r}^T\mathbf{b} + c \cdot \lfloor \frac{q+1}{2} \rfloor\right) \mod q\right).$$

Then, during decryption, we have

$$\begin{aligned} m' &= (v - \langle \mathbf{u}, \mathbf{s} \rangle) \mod q \\ &= \left(\mathbf{r}^T\mathbf{A}\mathbf{s} + \mathbf{r}^T\mathbf{e} + c \cdot \lfloor \frac{q+1}{2} \rfloor\right) - \mathbf{r}^T\mathbf{A}\mathbf{s} \mod q \\ &= \mathbf{r}^T\mathbf{e} + c \cdot \lfloor \frac{q+1}{2} \rfloor \mod q. \end{aligned}$$

Since $\|\mathbf{e}\|$ is small, we can recover the original message c perfectly.

6.3 Security

The security of this encryption scheme is based on the hardness of the LWE problem. Specifically, the IND-CPA security of the scheme can be reduced to the Decisional-LWE problem. By the DLWE assumption, we claim that

$$\begin{aligned} (\mathbf{A}, \mathbf{b}, \mathbf{r}^T\mathbf{A}, \mathbf{r}^T\mathbf{b}) &\approx_c (\mathbf{A}, \mathbf{u}, \mathbf{r}^T\mathbf{A}, \mathbf{r}^T\mathbf{u}) \\ &\approx_s (\mathbf{A}, \mathbf{u}, \mathbf{r}^T\mathbf{A}, z). \end{aligned}$$

where $\mathbf{u} \leftarrow \mathbb{Z}_q^m$, and $z \leftarrow \mathbb{Z}_q$. The last approximation follows from the *Leftover-hash Lemma*. Due to this $\mathbf{r}^T\mathbf{b}$ can be used as one-time-pad to mask c.

7 Lattice-based Cryptography and Fully Homomorphic Encryption

Since we will cover FHE in some later lectures, we will not give too many details here.

Fully homomorphic encryption (FHE) allows computation on encrypted data without the need for decryption, thereby providing privacy and security guarantees for sensitive computations. In essence, FHE schemes enable arbitrary computations on encrypted data while revealing no information about the inputs or the intermediate results.

Lattice-based cryptography has emerged as a promising approach to achieving FHE. The hardness of lattice problems, particularly the LWE problem, serves as a foundation for constructing FHE schemes. Craig Gentry [Gen09], in his groundbreaking work, introduced the first FHE scheme, which was based on a lattice problem known as the *ideal lattice problem*.

In summary, lattice-based cryptography, due to the hardness of lattice problems and the ability to perform arithmetic operations on lattice points, has become a powerful tool for constructing FHE schemes. This has far-reaching implications for secure and private computation in various domains, such as cloud computing, data analysis, and machine learning.

References

[ACK+21] Divesh Aggarwal, Yanlin Chen, Rajendra Kumar, Zeyong Li, and Noah Stephens-Davidowitz. Dimension-preserving reductions between svp and cvp in different p-norms. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2444–2462. SIAM, 2021.

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM (JACM)*, 50(4):506–519, 2003.

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10, Chicago, Illinois, USA, 1988. ACM.

[BS06] Mihir Bellare and Amit Sahai. Non-malleable encryption: Equivalence between two notions, and an indistinguishability-based characterization. Cryptology ePrint Archive, Paper 2006/228, 2006. https://eprint.iacr.org/2006/228.

[BS15] Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography*. 2015.

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. Cryptology ePrint Archive, Paper 1998/006, 1998. https://eprint.iacr.org/1998/006.

[EV20] Friedrich Eisenbrand and Moritz Venzin. Approximate CVP_p in time $2^{0.802n}$. *arXiv preprint arXiv:2005.04957*, 2020.

[Gam84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.

[GB01] Shafi Goldwasser and Mihir Bellare. *Lecture Notes on Cryptography*. 2001.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. Stanford university, 2009.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, aug 1986.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*, pages 25–32. ACM, 1989.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity. *Journal of the ACM*, 38(3):691–729, 1991.

[Gol00] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, USA, 2000.

[HILL99a] Johan Hastad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. pages 1–46, 1999.

[HILL99b] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[IL89] Russell Impafliazzo and Michael Luby. One-way functions are essential for complexity based cryptography (extended abstract). page 9, 1989.

[IN89] R. Impagliazzo and M. Naor. Efficient cryptographic schemes provably as secure as subset sum. In *30th Annual Symposium on Foundations of Computer Science*, pages 236–241, 1989.

[Kal03] Adam Tauman Kalai. Generating random factored numbers, easily. *Journal of Cryptology*, 16(4):287–289, 2003.

[LLL82] Arjen K Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische annalen*, 261(ARTICLE):515–534, 1982.

[Mer74] Ralph Merkle. First cs244 project proposal. pages 1–7, 1974.

[NY89a] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 33–43, 1989.

[NY89b] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*. ACM, 1989.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 427–437, 1990.

[PS98] Sarvar Patel and Ganapathy S. Sundaram. An efficient discrete log pseudo random generator. *Advances in Cryptology — CRYPTO '98*, page 304–317, 1998.

[Rab79] M. O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical report, USA, 1979.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[Sch96] Bruce Schneier. *Applied cryptography - protocols, algorithms, and source code in C, 2nd Edition*. Wiley, 1996.

[SE94] Claus-Peter Schnorr and Martin Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66:181–199, 1994.

[Seu12] Yannick Seurin. On the exact security of schnorr-type signatures in the random oracle model. Cryptology ePrint Archive, Paper 2012/029, 2012. https://eprint.iacr.org/2012/029.

[Sha48] Claude E. Shannon. *A Mathematical Theory of Communication.* The Bell System Technical Journal, 1948.

[Sha49] Claude E. Shannon. Communication theory of secrecy systems. *Bell Syst. Tech. J.*, 28(4):656–715, 1949.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[Sho94] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[SHO97] PW SHOR. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM journal on computing (Print)*, 26(5):1484–1509, 1997.