Introduction to Cryptology

Master 1 Lecture Notes

Alain Passelègue and Damien Stehlé

September 24, 2026

Contents

Chapter 1

Security and Perfect Secrecy

We first introduce the scope and methodology of modern cryptology. We then study perfect secrecy through the one-time pad and explain why practical encryption must replace information-theoretic guarantees with computational ones. This leads to the first definition of a secure pseudorandom generator.

1.1 What is cryptology?

Cryptology is the science of securing data against adversaries. Encryption is one of its central tools, but the topic is broader than encryption alone: data need not be in transit in order to require protection, and confidentiality is not the only security property we may want to attach to data. Conversely, cryptology is not the whole of computer security. Topics such as social engineering, implementation bugs, risk management, system security, hardware security, and digital forensics require methods that go beyond the mathematical models developed in this course.

Example 1.1 (Comparing distant files)**.** Suppose two parties hold large files and want to check whether the files are identical while communicating very little. Rather than transmitting the files themselves, they can compare short hashes. An adversary may wish to alter one of the files so that the hashes still match; cryptology provides ways to make such tampering computationally intractable. This illustrates a cryptographic task that is not, in itself, an encryption problem.

1.1.1 Secure communication over an open channel

A classical setting modeling communication in the presence of an eavesdropper has two honest parties, Alice and Bob, and an adversary, Eve, who can observe the channel (Figure 1.1). A simplified view of a connection to a web service is obtained by reading "Alice" as the user and "Bob" as the server.

At a high level, a TLS-style secure connection proceeds in three stages.

1. **Authentication.** Bob proves his identity to Alice.
2. **Handshake.** Alice and Bob establish shared secret key material.
3. **Protected communication.** They use the resulting shared keys to encrypt and decrypt their messages.

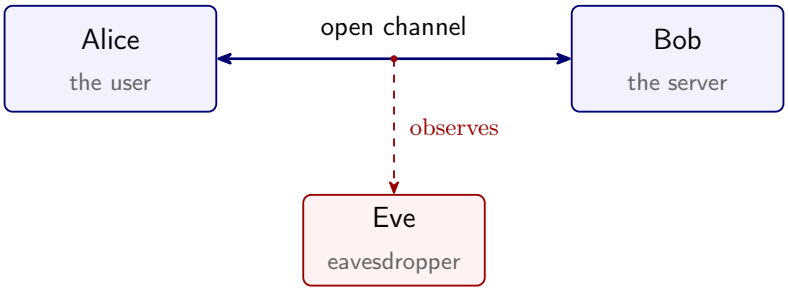

Figure 1.1: Communication over an open channel: Alice and Bob exchange messages while Eve observes the traffic.

The third stage uses *symmetric-key cryptography*: the communicating parties share the same secret. We study this stage in the first half of the course; authentication and key establishment are addressed in the second half.

Before constructing a protocol, we must say precisely what "secure" means. Three fundamental properties are:

Authenticity: Alice is assured that she is communicating with Bob.

Integrity: an adversary cannot modify, insert, or erase part of the communication without detection.

Confidentiality: Eve cannot learn the protected content of the communication.

A protocol is therefore specified not only by its functionality, but also by its desired security properties and by the capabilities granted to the attacker. Complex protocols are assembled from simpler cryptographic building blocks, called *primitives*. Examples of complex protocols include:

- anonymous communication, as provided by Tor, which seeks to conceal who communicates with whom, so that an observer cannot link senders to receivers;
- electronic cash, which seeks anonymity, authenticity, and protection against duplication;
- cryptocurrencies, which secure a shared transaction ledger through distributed consensus, so that ownership and the absence of double-spending can be verified publicly without a trusted authority; unlike electronic cash, they usually do not aim for payer anonymity;
- digital identity, where a user proves attributes about themselves (an age, a nationality, the possession of a valid credential) while revealing as little as possible and, ideally, without letting distinct uses be linked;
- electronic voting, where requirements may include ballot secrecy, resistance to ballot stuffing and coercion, and verifiability.

Cryptographic protection also applies to data at rest, as in disk encryption, and can even allow computations to be carried out on encrypted data. A related capability is secure multi-party computation (MPC), in which several parties jointly evaluate a function of their private inputs while learning nothing beyond the result. A classic illustration is Yao's millionaires' problem, in which two people determine who is richer without revealing how much either owns.

1.1.2 Cryptography as a science

A typical cryptographic development follows five steps.

1. Define the functionality of the protocol or primitive.
2. Define the attacker's goal.
3. Define the attacker's capabilities.
4. Propose a construction realizing the functionality.
5. Prove that any successful, efficient attack would yield an efficient solution to an algorithmic problem believed to be intractable.

The last step is typically achieved by a *reduction*. If breaking the construction provides an algorithm to solve a problem assumed to be hard, then, by contraposition, the construction inherits security from that assumption. This is one of the constructive faces of complexity theory. It turns out that many hardness assumptions used in cryptography are algebraic, which is why algebra plays such an important role in the subject.

Cryptography is not an art of hiding protocol descriptions. Designs should be public and open to analysis; the source of security should be concentrated in well-defined secret information, typically a randomly chosen key. This is Kerckhoffs's principle: security should not rely on obscurity.

1.2 Perfect secrecy

We begin with information-theoretic security. At this stage there are no efficiency requirements: even an adversary with unlimited computational power must learn nothing about the plaintext.

1.2.1 Finite encryption schemes

Definition 1.2 (Encryption scheme)**.** Let $\mathcal{K}$, $\mathcal{P}$, and $\mathcal{C}$ be finite sets of keys, plaintexts, and ciphertexts, respectively. An encryption scheme over $(\mathcal{K}, \mathcal{P}, \mathcal{C})$ consists of:

- a randomized key-generation algorithm KeyGen that samples a key $K \in \mathcal{K}$;
- a possibly randomized encryption algorithm $\mathsf{Enc}\colon \mathcal{K} \times \mathcal{P} \to \mathcal{C}$;
- a deterministic decryption algorithm $\mathsf{Dec}\colon \mathcal{K} \times \mathcal{C} \to \mathcal{P}$.

It is *correct* if, for every $k \in \mathcal{K}$, every $m \in \mathcal{P}$, and every possible outcome c of $\mathsf{Enc}(k, m)$, it holds that

$$\mathsf{Dec}(k, c) = m \ .$$

We may assume without loss of generality that $\mathcal{C}$ is the supported image of Enc; ciphertexts that can never occur may simply be removed. We treat the key K, the plaintext M, and the ciphertext $\mathsf{Enc}_K(M)$ as random variables. We further assume that M and K are statistically independent and that every element of $\mathcal{P}$ and $\mathcal{K}$ has nonzero probability. Any probability involving the Enc algorithm is also over its internal randomness.

Definition 1.3 (Perfect secrecy, Shannon (1949))**.** An encryption scheme is *perfectly secret* if, for every $m \in \mathcal{P}$ and every supported $c \in \mathcal{C}$,

$$\Pr\big[M = m \mid \mathsf{Enc}_K(M) = c\big] = \Pr[M = m] \ .$$

In words, revealing the ciphertext gives no more information about the plaintext than was available before it was seen: conditioning on the ciphertext leaves the distribution of M unchanged. Equivalently, the random variables M and $\mathsf{Enc}_K(M)$ are statistically independent.

Lemma 1.4 (Shannon's bound)**.** *Every perfectly secret encryption scheme satisfies*

$$|\mathcal{K}| \geq |\mathcal{P}| \ .$$

Perfect secrecy forces the key space to be at least as large as the plaintext space. Informally, the key must be at least as long as the plaintext.

Proof. Fix any supported ciphertext $c \in \mathcal{C}$. For each $m \in \mathcal{P}$, perfect secrecy gives $\Pr[M = m \mid \mathsf{Enc}_K(M) = c] = \Pr[M = m] > 0$. Since c is supported we also have $\Pr[\mathsf{Enc}_K(M) = c] > 0$, so the joint event $\{M = m,\ \mathsf{Enc}_K(M) = c\}$ has positive probability, namely $\Pr[M = m \mid \mathsf{Enc}_K(M) = c] \cdot \Pr[\mathsf{Enc}_K(M) = c] = \Pr[M = m] \cdot \Pr[\mathsf{Enc}_K(M) = c]$. In particular there is a key k_m for which c is a possible output of $\mathsf{Enc}(k_m, m)$; correctness, which quantifies over every possible outcome of Enc, then gives $\mathsf{Dec}(k_m, c) = m$. The map $m \mapsto k_m$ is injective, since $k_m = k_{m'}$ would force the single value $\mathsf{Dec}(k_m, c)$ to equal both m and m'. Hence $|\mathcal{P}| \leq |\mathcal{K}|$. □

1.2.2 The one-time pad

The *one-time pad*, also called the Vernam cipher, is defined as follows. For a fixed plaintext length n, we set $\mathcal{K} = \mathcal{P} = \mathcal{C} = \{0,1\}^n$. Key generation samples k uniformly from $\{0,1\}^n$, and Enc and Dec are defined by

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

where $\oplus$ denotes bitwise exclusive OR.

Theorem 1.5. *The one-time pad is correct and perfectly secret.*

Proof. Correctness follows immediately from

$$(m \oplus k) \oplus k = m \ .$$

For perfect secrecy, let $m, c \in \{0,1\}^n$. Independence of M and the uniform key K gives

$$\begin{aligned} \Pr[M = m \ \wedge\ \mathsf{Enc}_K(M) = c] &= \Pr[M = m \ \wedge\ \mathsf{Enc}_K(m) = c] \\ &= \Pr[M = m] \cdot \Pr[K = c \oplus m] \\ &= \Pr[M = m] \cdot 2^{-n} \ . \end{aligned}$$

Moreover, we have:

$$\begin{aligned} \Pr[\mathsf{Enc}_K(M) = c] &= \sum_{m' \in \{0,1\}^n} \Pr[M = m'] \cdot \Pr[K = c \oplus m'] \\ &= \sum_{m' \in \{0,1\}^n} \Pr[M = m'] \cdot 2^{-n} = 2^{-n} \ . \end{aligned}$$

Dividing the two equalities yields

$$\Pr[M = m \mid \mathsf{Enc}_K(M) = c] = \Pr[M = m] \ ,$$

completing the proof. □

Perfect secrecy comes at a substantial practical cost:

- The key is at least as long as the plaintext.
- A key must never be reused. If $c_1 = m_1 \oplus k$ and $c_2 = m_2 \oplus k$, then

$$c_1 \oplus c_2 = m_1 \oplus m_2 \ ,$$

  which exposes a direct relation between the two plaintexts.
- The parties must share as much secret material as the total amount of protected communication.
- The one-time pad protects confidentiality against eavesdropping, but it does not by itself provide authenticity or integrity. For instance, an adversary who intercepts $c = m \oplus k$ can replace it with $c \oplus \delta$ for any chosen mask δ; the receiver then decrypts $(c \oplus \delta) \oplus k = m \oplus \delta$, flipping exactly the bits of m selected by δ without ever learning m or k.

Chapter 2

Pseudorandom Generators

To avoid keys as long as the plaintexts in encryption, we replace the uniformly random pad with pseudorandom bits generated from a much shorter random seed. We seek a deterministic algorithm

$$G\colon \{0,1\}^s \longrightarrow \{0,1\}^n \ , \qquad n \gg s \ ,$$

whose output *looks* uniform when its seed is uniform. Further, the algorithm must be deterministic: all genuine randomness is placed in its input seed.

This suggests the following encryption scheme for n-bit plaintexts:

$$k \leftarrow\!\!\!\!\rightarrow \{0,1\}^s \ , \qquad \mathsf{Enc}_k(m) = m \oplus G(k) \ , \qquad \mathsf{Dec}_k(c) = c \oplus G(k) \ .$$

It cannot achieve perfect secrecy when $s < n$: the plaintext space has size 2^n, whereas the key space has size only 2^s, contradicting Lemma 1.4. The appropriate replacement is a *computational* rather than *statistical* notion of security.

In fact, the distributions $G(U(\{0,1\}^s))$ and $U(\{0,1\}^n)$ can always be told apart in principle: the image of G contains at most 2^s of the 2^n strings of $\{0,1\}^n$, so it is extremely sparse in the space where it lives, and an unbounded algorithm that tests membership in this image separates the two distributions almost perfectly. All we can hope for is that building such a distinguisher is computationally hard.

2.1 Indistinguishability of distributions

To formalize what it means for one distribution to *look* like another, we quantify how well an algorithm can tell them apart.

Definition 2.1 (Distinguishing advantage and indistinguishability)**.** Let D_0 and D_1 be distributions over $\{0,1\}^n$, and let $\mathcal{A}\colon \{0,1\}^n \to \{0,1\}$ be a possibly randomized algorithm. The *advantage* of $\mathcal{A}$ in distinguishing D_0 from D_1 is

$$\mathsf{Adv}_{\mathcal{A}}(D_0, D_1) = \left| \Pr_{X \leftarrow\!\!\!\!\rightarrow D_0}[\mathcal{A}(X) = 1] - \Pr_{X \leftarrow\!\!\!\!\rightarrow D_1}[\mathcal{A}(X) = 1] \right| \ ,$$

where the probabilities are also taken over the internal randomness of $\mathcal{A}$.

Now let $(D_0^{(n)})_{n\geq 1}$ and $(D_1^{(n)})_{n\geq 1}$ be sequences of distributions, with $D_0^{(n)}$ and $D_1^{(n)}$ over $\{0,1\}^n$; they are *computationally indistinguishable* if, for every efficient algorithm $\mathcal{A}$, the advantage $\mathsf{Adv}_{\mathcal{A}}\big(D_0^{(n)}, D_1^{(n)}\big)$ is negligible.

Intuitively, D_0 and D_1 are computationally indistinguishable when no efficient algorithm does noticeably better than random guessing at deciding which of the two a given sample was drawn from. This definition becomes fully formal only after fixing what "efficient" and "negligible" mean.

- In concrete security, one states a pair of bounds (t, ε): adversaries running for at most time t should have advantage at most ε. Illustrative targets use work factors such as 2^{128} or more and advantages such as 2^{-80} or 2^{-128}. Determining the correct pair for a concrete primitive is the problem of assessing its *bit security*.
- In asymptotic theory, one considers distributions over $\{0,1\}^n$ where $n(\lambda)$ is a function of a security parameter λ. An efficient adversary runs in time $\mathrm{poly}(\lambda)$, and a function is negligible if, for every constant $c > 0$, it is eventually smaller than $1/\lambda^c$; this is written $\lambda^{-\omega(1)}$.
- On an exponential security scale, cryptographers also commonly compare attackers of running time $2^{o(\lambda)}$ with advantages of order $2^{-\Omega(\lambda)}$.

The advantage in Definition 2.1 compares two fixed experiments, one for each distribution. An equivalent and often more convenient formulation folds both experiments into a single guessing game: the challenger secretly tosses a coin to select one of the two distributions, and the adversary must guess which one it drew from. This is illustrated in Figure 2.1.

Definition 2.2 (Guessing advantage)**.** Let D_0 and D_1 be distributions over $\{0,1\}^n$, and let $\mathcal{A}\colon \{0,1\}^n \to \{0,1\}$ be a possibly randomized algorithm. Consider the experiment in which the challenger samples a uniform bit $\beta \hookleftarrow U(\{0,1\})$, then samples $X \hookleftarrow D_\beta$ and sends X to $\mathcal{A}$, which returns a bit β'. The *guessing advantage* of $\mathcal{A}$ is

$$\mathsf{Adv}'_{\mathcal{A}}(D_0, D_1) = \left| \, 2\Pr[\beta' = \beta] - 1 \, \right| \ ,$$

where the probability is over β, the sampling of X, and the internal randomness of $\mathcal{A}$.

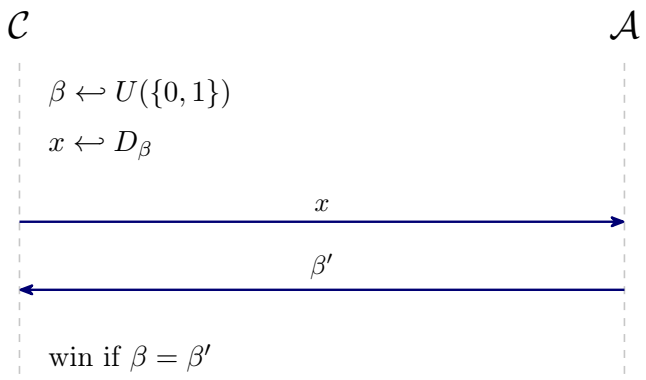

Figure 2.1: The guessing game behind $\mathsf{Adv}'_{\mathcal{A}}$.

Lemma 2.3. *For every algorithm $\mathcal{A}$ and all distributions D_0, D_1 over $\{0,1\}^n$, it holds that*

$$\mathsf{Adv}'_{\mathcal{A}}(D_0, D_1) = \mathsf{Adv}_{\mathcal{A}}(D_0, D_1) \ .$$

2.2 Pseudorandom generators

The image of G contains at most 2^s of the 2^n possible strings. Thus $G(U(\{0,1\}^s))$ is very far from uniform from an information-theoretic point of view. Indeed, an all-powerful algorithm can test membership in $\mathrm{Im}(G)$: it accepts $G(U(\{0,1\}^s))$ with probability 1, but accepts $U(\{0,1\}^n)$ with probability at most 2^{s-n}. This is $\ll 1$ when $n \gg s$. Security can therefore hold only against computationally bounded tests.

Definition 2.4 (Secure pseudorandom generator)**.** A deterministic function $G\colon \{0,1\}^s \to \{0,1\}^n$, with $n > s$, is a *secure pseudorandom generator* (PRG) if the distributions $G(U(\{0,1\}^s))$ and $U(\{0,1\}^n)$ are computationally indistinguishable.

A corresponding distinguishing experiment is pictured in Figure 2.2. The output bit β' may be interpreted as the distinguisher's guess about the nature of X (pseudorandom or uniform).

Input: a bit $\beta \in \{0,1\}$.

1. If $\beta = 0$, the challenger samples $X \leftarrow\!\!\shortmid U(\{0,1\}^n)$.
2. If $\beta = 1$, the challenger samples $S \leftarrow\!\!\shortmid U(\{0,1\}^s)$ and sets $X = G(S)$.
3. The challenger sends X to the distinguisher $\mathcal{A}$.
4. The distinguisher returns a bit β'.

Figure 2.2: The uniform-versus-PRG distinguishing experiment.

To make the asymptotic definition fully precise, one models a PRG as a pair (Setup, G) of algorithms. On input 1^λ, the probabilistic algorithm Setup outputs public parameters pp that fix the input and output lengths $s(\lambda)$ and $n(\lambda)$; the deterministic algorithm G maps pp and a seed $k \in \{0,1\}^{s(\lambda)}$ to $G(\mathsf{pp}, k) \in \{0,1\}^{n(\lambda)}$. The security parameter λ is fed in unary so that a running time polynomial in the input size is polynomial in λ. Security then asks that $G\big(\mathsf{pp}, U(\{0,1\}^{s(\lambda)})\big)$ and $U(\{0,1\}^{n(\lambda)})$ be computationally indistinguishable when pp is also given to the distinguisher.

Example 2.5 (Library random-number generators)**.** The everyday pseudorandom generators of a programming language illustrate the gap between merely *looking* random and being secure. The C library functions `rand` and `random` stretch a small seed into a long output that passes simple statistical tests, yet their internal recurrence is (essentially) linear: a short run of consecutive outputs suffices to reconstruct the internal state and thereby predict all subsequent outputs. They are *not* secure generators and must never be used for cryptography. Operating systems provide a separate cryptographic generator: on Linux, the source `/dev/urandom` and the `getrandom` system call are built on a secure stream cipher (Chacha20).

2.3 Unpredictability and the next-bit test

Definition 2.1 lets an adversary apply an arbitrary efficient test to a sample. A single, very concrete test turns out to capture the entire notion: can an efficient algorithm, shown a prefix of the output, guess the following bit better than by chance? A generator that resists this one *next-bit test* already resists every efficient test.

For $x \in \{0,1\}^n$ and $0 \le i \le n$, we write $x_{1\ldots i}$ for the length-i prefix of x (empty when $i = 0$) and x_{i+1} for its $(i{+}1)$-th bit.

Definition 2.6 (Unpredictability)**.** A generator $G\colon \{0,1\}^s \to \{0,1\}^n$ is *predictable* if there exist an efficient algorithm $\mathcal{A}$ and an index i with $0 \le i < n$ such that

$$\left| \Pr_{k \leftarrow U(\{0,1\}^s)} \big[\mathcal{A}\big(G(k)_{1\ldots i}\big) = G(k)_{i+1}\big] - \tfrac{1}{2} \right| \;\ge\; \varepsilon$$

for some non-negligible ε, the probability being taken also over the internal randomness of $\mathcal{A}$. Such an $\mathcal{A}$ is called a *next-bit predictor* at position i. The generator is *unpredictable* if no efficient next-bit predictor exists at any position.

When $i = 0$, the predictor receives no input and must guess the very first output bit; in general it sees the first i bits and tries to anticipate the next one.

Theorem 2.7 (Yao, 1982)**.** *A generator $G\colon \{0,1\}^s \to \{0,1\}^n$ with $n > s$ is a secure pseudorandom generator if and only if it is unpredictable.*

The forward direction is immediate: a predictor is a particular distinguisher. The converse, due to Yao, is the substantial part and rests on a *hybrid argument*, a technique we shall reuse throughout the course.

Proof. A predictor yields a distinguisher. Suppose G is predictable, and let $\mathcal{A}$ be a next-bit predictor at some position i with advantage ε. Define $\mathcal{A}'\colon \{0,1\}^n \to \{0,1\}$ by

$$\mathcal{A}'(x) = 1 \quad \text{if and only if} \quad \mathcal{A}\big(x_{1\ldots i}\big) = x_{i+1} \;.$$

It runs $\mathcal{A}$ once and compares its guess with the actual bit x_{i+1}, so it is efficient. On a pseudorandom input $x = G(k)$, we have:

$$\Pr\big[\mathcal{A}'(G(k)) = 1\big] = \Pr\big[\mathcal{A}\big(G(k)_{1\ldots i}\big) = G(k)_{i+1}\big] \;.$$

On a uniform input $x \leftarrow U(\{0,1\}^n)$, the bit x_{i+1} is uniform and independent of the prefix $x_{1\ldots i}$, so $\mathcal{A}$'s guess matches it with probability exactly $1/2$:

$$\Pr_{x \leftarrow U(\{0,1\}^n)} \big[\mathcal{A}'(x) = 1\big] = \tfrac{1}{2} \;.$$

Subtracting, we obtain

$$\mathsf{Adv}_{\mathcal{A}'}\big(G(U(\{0,1\}^s)), U(\{0,1\}^n)\big) = \left| \Pr\big[\mathcal{A}\big(G(k)_{1\ldots i}\big) = G(k)_{i+1}\big] - \tfrac{1}{2} \right| \ge \varepsilon \;,$$

which is non-negligible, so G is insecure. Moreover, algorithm $\mathcal{A}'$ runs $\mathcal{A}$ once and performs a single bit comparison, so its running time is essentially that of $\mathcal{A}$.

A distinguisher yields a predictor (hybrid argument). Suppose conversely that G is insecure, and let $\mathcal{A}\colon \{0,1\}^n \to \{0,1\}$ be an efficient distinguisher with

$$\mathsf{Adv}_{\mathcal{A}}\big(G(U(\{0,1\}^s)), U(\{0,1\}^n)\big) \ge \varepsilon$$

for some non-negligible ε. Introduce the *hybrid* distributions $H_0, \ldots, H_n$ over $\{0,1\}^n$: to sample from H_i, draw $k \leftarrow U(\{0,1\}^s)$ and $u \leftarrow U(\{0,1\}^{n-i})$ and output

$$G(k)_{1\ldots i} \,\|\, u \;,$$

so the first i bits are those of $G(k)$ and the remaining $n-i$ bits are uniform and independent. The extremes are the two distributions of interest,

$$H_0 = U(\{0,1\}^n) \;, \qquad H_n = G(U(\{0,1\}^s)) \;.$$

Write $p_i = \Pr[\mathcal{A}(H_i) = 1]$. A telescoping sum and the triangle inequality give

$$\sum_{0 \le i < n} \big|p_{i+1} - p_i\big| \;\ge\; \big|p_n - p_0\big| = \mathsf{Adv}_{\mathcal{A}}\big(H_n, H_0\big) \ge \varepsilon \;\;,$$

so there is an index i with $|p_{i+1} - p_i| \ge \varepsilon/n$. We do not need to identify i: since n is small we may build one candidate predictor per position and argue that at least one succeeds.

Fix such an i and define the predictor $\mathcal{A}'_i$. On input a prefix $w = G(k)_{1\ldots i}$, it samples a bit $g \hookleftarrow U(\{0,1\})$ and a string $u \hookleftarrow U(\{0,1\}^{n-i-1})$, forms $x = w \,\|\, g \,\|\, u$, and runs $\beta \leftarrow \mathcal{A}(x)$; it outputs g if $\beta = 1$ and $\overline{g}$ otherwise. The idea is that g is a random guess for the true next bit $G(k)_{i+1}$, and $\mathcal{A}$ is used to decide whether to trust it: if $\mathcal{A}$ replies 1 it "believes" the sample is more pseudorandom, hence that g is the correct bit. Note that the string x handed to $\mathcal{A}$ is distributed exactly as H_i.

Let E denote the event "$g = G(k)_{i+1}$", i.e., that the guess is correct. Since g is uniform and independent of the true bit $G(k)_{i+1}$, we have $\Pr[E] = 1/2$, and conditioned on E the string x is distributed as H_{i+1}. Because x is unconditionally distributed as H_i, we also have $\Pr[\beta = 1] = p_i$. The predictor is correct exactly when $\beta = 1$ and E holds, or $\beta = 0$ and E fails, so

$$\begin{aligned}
\Pr\big[\mathcal{A}'_i\big(G(k)_{1\ldots i}\big) = G(k)_{i+1}\big] &= \Pr[\beta = 1 \;\wedge\; E] + \Pr[\beta = 0 \;\wedge\; \overline{E}] \\
&= \Pr[\beta = 1 \;\wedge\; E] + \Pr[\overline{E}] - \Pr[\beta = 1 \;\wedge\; \overline{E}] \\
&= \tfrac{1}{2} + \Pr[\beta = 1 \;\wedge\; E] - \Pr[\beta = 1 \;\wedge\; \overline{E}] \;\;.
\end{aligned}$$

Here we have

$$\Pr[\beta = 1 \wedge E] = \Pr[E] \cdot \Pr[\beta = 1 | E] = \tfrac{1}{2}\Pr[\mathcal{A}(H_{i+1}) = 1] = \tfrac{1}{2} p_{i+1} \;\;,$$

whereas

$$\Pr[\beta = 1 \wedge E] + \Pr[\beta = 1 \wedge \overline{E}] = \Pr[\beta = 1] = p_i \;\;,$$

so that

$$\Pr[\beta = 1 \wedge \overline{E}] = p_i - \tfrac{1}{2} p_{i+1} \;\;.$$

Substituting, we obtain:

$$\Pr\big[\mathcal{A}'_i\big(G(k)_{1\ldots i}\big) = G(k)_{i+1}\big] = \tfrac{1}{2} + \big(p_{i+1} - p_i\big) \;\;,$$

and

$$\Big|\Pr\big[\mathcal{A}'_i\big(G(k)_{1\ldots i}\big) = G(k)_{i+1}\big] - \tfrac{1}{2}\Big| = \big|p_{i+1} - p_i\big| \;\ge\; \frac{\varepsilon}{n} \;\;.$$

Since n is polynomial and ε non-negligible, the quantity ε/n is non-negligible, so $\mathcal{A}'_i$ is an efficient next-bit predictor and G is predictable. □

2.4 Encrypting with a pseudorandom generator

We now return to the encryption scheme sketched at the start of the chapter and prove that a secure PRG makes it secure, once "secure" is given the right computational meaning. Recall the construction: for n-bit plaintexts,

$$k \hookleftarrow U(\{0,1\}^s) \;\;, \qquad \mathsf{Enc}_k(m) = m \oplus G(k) \;\;, \qquad \mathsf{Dec}_k(c) = c \oplus G(k) \;\;,$$

where $G\colon \{0,1\}^s \to \{0,1\}^n$ with $n \gg s$. This is a *stream cipher*: the short seed k is stretched into a long pseudorandom pad $G(k)$, which is then used exactly as a one-time pad.

2.4.1 The danger of reusing the key

The one-time pad forbids reusing a key, and the stream cipher inherits the prohibition. Encrypting two plaintexts under the same k produces

$$c_1 = m_1 \oplus G(k) \ , \qquad c_2 = m_2 \oplus G(k) \ ,$$

so that

$$c_1 \oplus c_2 = m_1 \oplus m_2 \ ,$$

leaking the exclusive-or of the plaintexts to any eavesdropper; the pseudorandom pad cancels and offers no protection. For this reason the pad $G(k)$ must be used only once, exactly as in the information-theoretic setting.

Example 2.8 (The WEP failure)**.** Early Wi-Fi confidentiality (WEP) used the stream cipher $c = m \oplus \mathrm{PRG}(\mathrm{IV} \,\|\, k)$ and transmitted the pair (IV, c), where the initialization vector IV was 24 bits long and the shared key k was 104 bits long, so that the generator was seeded with 128 bits. The seed changes with each frame only through the 24-bit IV, so after on the order of 2^{24} frames (and, by the birthday effect, far sooner) an IV repeats and the same pad is used twice, reinstating the two-time-pad relation above. A further weakness is that successive IVs are often produced by a counter and are therefore strongly correlated $(1, 2, 3, \ldots)$, whereas nothing in the design guarantees that the corresponding pads look independent. Although these attacks are decades old, WEP long survived in the field, remaining in use on a nontrivial fraction of wireless networks well after it had been thoroughly broken.

The lesson is that a construction is only as good as the model it is proved secure in. We therefore need a definition that captures precisely what the stream cipher does guarantee.

2.4.2 Security for a single encrypted message

It is instructive to first restate perfect secrecy in a form built for comparison, and then relax it from a statistical to a computational requirement.

Lemma 2.9 (Perfect secrecy, indistinguishability form)**.** *An encryption scheme over $(\mathcal{K}, \mathcal{P}, \mathcal{C})$ is perfectly secret in the sense of Definition 1.3 if and only if, for all plaintexts $m_0, m_1 \in \mathcal{P}$, the ciphertext distributions*

$$\{\mathsf{Enc}_K(m_0)\} \qquad \textit{and} \qquad \{\mathsf{Enc}_K(m_1)\}$$

induced by a random key $K \leftarrow\!\!\!\rightarrow \mathsf{KeyGen}$ are identical.

Proof. Perfect secrecy is equivalent to the statistical independence of M and $\mathsf{Enc}_K(M)$ (see Section 1.2). Independence means that the conditional distribution of $\mathsf{Enc}_K(M)$ given $M = m$ does not depend on m, which is precisely the stated equality of $\{\mathsf{Enc}_K(m_0)\}$ and $\{\mathsf{Enc}_K(m_1)\}$ for all m_0, m_1. Conversely, if all these conditional distributions coincide, they equal the marginal of $\mathsf{Enc}_K(M)$, so M and $\mathsf{Enc}_K(M)$ are independent. □

In this form, perfect secrecy says that no observer, however powerful, can tell apart the encryption of m_0 from that of m_1, because the two ciphertext distributions are literally the same. The computational analogue keeps the same shape but replaces "identical" with "computationally indistinguishable," and grants the adversary the right to choose the two plaintexts it wishes to be challenged on.

Definition 2.10 (Single-message chosen-plaintext security)**.** Let $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme. Consider the following guessing game played between a challenger and an adversary $\mathcal{A}$:

1. the challenger samples a key $k \leftarrow\!\!\!\rightarrow \mathsf{KeyGen}$ and a uniform bit $\beta \leftarrow\!\!\!\rightarrow U(\{0,1\})$;
2. the adversary chooses two distinct plaintexts $m_0, m_1 \in \mathcal{P}$ of equal bit-length and sends them to the challenger;
3. the challenger returns the challenge ciphertext $c \leftarrow\!\!\!\rightarrow \mathsf{Enc}(k, m_\beta)$;
4. the adversary outputs a bit β', its guess for β.

The *advantage* of $\mathcal{A}$ is

$$\mathsf{Adv}_{\mathcal{A}} = \left| \, 2\Pr[\beta' = \beta] - 1 \, \right| \ ,$$

and the scheme is *smCPA-secure* (secure under a single-message chosen-plaintext attack) if $\mathsf{Adv}_{\mathcal{A}}$ is negligible for every efficient $\mathcal{A}$. The experiment is depicted in Figure 2.3. By the equivalence of Lemma 2.3, applied to the distributions $D_\beta = \{\mathsf{Enc}(K, m_\beta)\}$, this is the same as asking that $\{\mathsf{Enc}(K, m_0)\}$ and $\{\mathsf{Enc}(K, m_1)\}$ be computationally indistinguishable—the computational analogue of the perfect-secrecy form in Lemma 2.9.

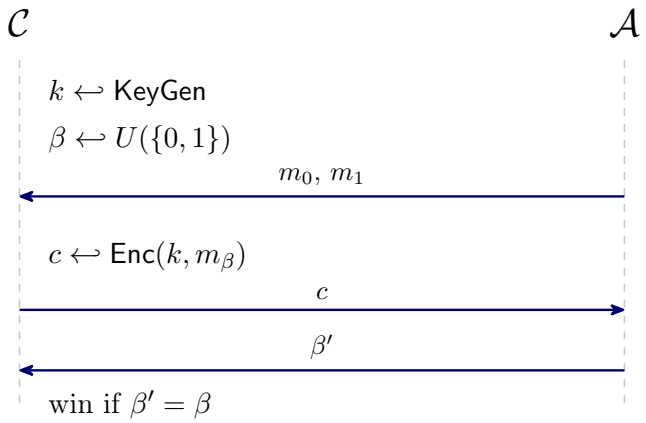

Figure 2.3: The single-message chosen-plaintext guessing game: the challenger encrypts m_β for a secret uniform bit β, and the adversary must guess β.

The qualifier "single-message" is a genuine restriction: the adversary sees exactly one challenge ciphertext. The two-time-pad attack of Section 2.4.1, which needs two ciphertexts under the same key, therefore falls outside this model and is *not* ruled out by smCPA-security. This is a deliberate illustration of the discipline "model, definition, proof": a proof certifies security only against the attacks the model permits, and using a stream cipher twice steps outside it.

2.4.3 A secure PRG yields a secure encryption scheme

We can now close the loop and prove that the stream cipher of the start of the chapter is smCPA-secure, provided the underlying generator is secure.

Theorem 2.11. *Let $G\colon \{0,1\}^s \to \{0,1\}^n$ be a secure pseudorandom generator, and define the encryption scheme with key space $\{0,1\}^s$ and message space $\{0,1\}^n$ by*

$$\mathsf{KeyGen}: \ k \leftarrow\!\!\!\rightarrow U(\{0,1\}^s) \ , \qquad \mathsf{Enc}_k(m) = m \oplus G(k) \ , \qquad \mathsf{Dec}_k(c) = c \oplus G(k) \ .$$

Then $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is smCPA-secure. More precisely, any adversary* $\mathcal{A}$ *breaking smCPA-security with advantage* ε *can be turned into a distinguisher* $\mathcal{B}$ *against* G *with advantage at least* $\varepsilon/2$ *and essentially the same running time.*

The proof is by *reduction*: assuming an attacker $\mathcal{A}$ against the encryption scheme, we build a distinguisher $\mathcal{B}$ against G. The idea is that $\mathcal{B}$ plays the challenger for $\mathcal{A}$ but uses its own challenge string as the pad. If that string is $G(k)$, then $\mathcal{A}$ faces a genuine encryption and, by assumption, guesses well; if the string is uniform, then $\mathcal{A}$ faces a true one-time pad and can do no better than chance. The gap between the two cases is exactly what $\mathcal{B}$ needs. The reduction is pictured in Figure 2.4.

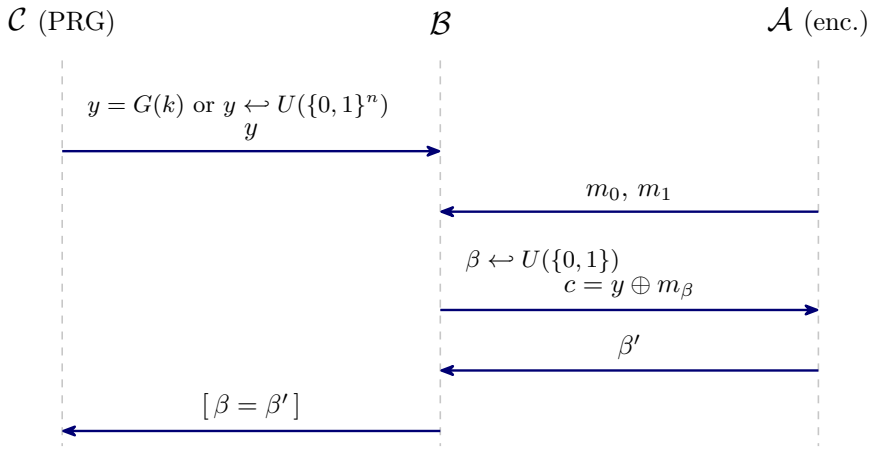

Figure 2.4: Reduction from distinguishing G to attacking the encryption scheme. The wrapper $\mathcal{B}$ uses its PRG challenge y as the pad and outputs 1 ("pseudorandom") exactly when $\mathcal{A}$ guesses β correctly.

Proof. Let $\mathcal{A}$ be an efficient smCPA-adversary with advantage $\mathsf{Adv}_{\mathcal{A}} \geq \varepsilon$. Construct a distinguisher $\mathcal{B}$ for G as follows. On input a string $y \in \{0,1\}^n$, algorithm $\mathcal{B}$ samples a uniform bit $\beta \leftarrow\!\!\!\rightarrow U(\{0,1\})$, receives the two plaintexts m_0, m_1 from $\mathcal{A}$, returns the ciphertext $c = y \oplus m_\beta$, obtains $\mathcal{A}$'s guess β', and outputs 1 if and only if $\beta' = \beta$. Clearly $\mathcal{B}$ runs in essentially the same time as $\mathcal{A}$.

Consider the two possible inputs to $\mathcal{B}$.

When $y = G(k)$ for a uniform seed k, the ciphertext $c = m_\beta \oplus G(k)$ is a genuine encryption of m_β, so $\mathcal{B}$ reproduces exactly the smCPA guessing game for $\mathcal{A}$. Since $\mathcal{B}$ outputs 1 precisely when $\mathcal{A}$ guesses correctly, we have

$$\Pr[\mathcal{B}(G(k)) = 1] = \Pr_{\text{real}}[\beta' = \beta] \ ,$$

where $\Pr_{\text{real}}$ denotes probability in this experiment.

When $y \leftarrow\!\!\!\rightarrow U(\{0,1\}^n)$, the pad y is uniform and independent of everything else, so $c = m_\beta \oplus y$ is uniform and independent of β: a genuine one-time pad. The adversary's view then carries no information about β, and its guess is correct with probability exactly $1/2$, so

$$\Pr[\mathcal{B}(U) = 1] = \tfrac{1}{2} \ .$$

Subtracting the two cases and recalling that $\mathsf{Adv}_{\mathcal{A}} = \left|2\Pr_{\text{real}}[\beta' = \beta] - 1\right|$,

$$\begin{aligned}
\mathsf{Adv}^{\text{PRG}}_{\mathcal{B}} &= \left|\Pr[\mathcal{B}(G(U(\{0,1\}^s))) = 1] - \Pr[\mathcal{B}(U(\{0,1\}^n)) = 1]\right| \\
&= \left|\Pr_{\text{real}}[\beta' = \beta] - \tfrac{1}{2}\right| \\
&= \tfrac{1}{2}\left|2\Pr_{\text{real}}[\beta' = \beta] - 1\right| \\
&= \tfrac{1}{2}\,\mathsf{Adv}_{\mathcal{A}} \\
&\geq \frac{\varepsilon}{2}\ .
\end{aligned}$$

If ε were non-negligible, so would be $\varepsilon/2$, contradicting the security of G. Hence $\mathsf{Adv}_{\mathcal{A}}$ is negligible for every efficient $\mathcal{A}$, and the scheme is smCPA-secure. □

Theorem 2.11 is a security *proof*: it shows that *if* the assumption holds (here, that G is a secure PRG) *then* the scheme is secure. It does not assert security unconditionally, and its value should be weighed along three axes.

- The *weakness of the assumption*: is it plausible, well studied, and standard? A scheme resting on a widely scrutinised assumption is more trustworthy than one resting on an ad hoc one.
- The *strength of the conclusion*: how useful is the guarantee? Here smCPA-security is a modest goal (one message only), and later chapters strengthen it.
- The *reduction loss*: how much advantage is lost in passing from the attack to the solved problem? Our reduction loses a factor of 2, which is essentially tight and translates into only a one-bit loss of security.

Chapter 3

Computational Hardness Assumptions

Chapter 2 reduced the security of a stream cipher to the existence of a secure pseudorandom generator, but it did not exhibit one. We now address the question left open there: how does one build a generator whose output is indistinguishable from uniform? Following the methodology laid out in Chapter 1, we tie pseudorandomness to a precisely stated *algorithmic* problem believed to be intractable, so that a proof of security rests on a single, well-studied hardness assumption rather than on the accumulated failure of cryptanalysts to break a bespoke design.

This chapter introduces two families of such intractability assumptions: the discrete logarithm problem together with its Diffie–Hellman relatives, and the learning-with-errors problem. We shall rely on these assumptions repeatedly throughout the remaining lectures, so the chapter also records the concrete groups and parameters in which they are posed and how hard the underlying problems are believed to be. Once the decisional Diffie–Hellman assumption is in place, it yields a pseudorandom generator almost for free. The construction is little more than a restatement of the assumption, a tautology that we make precise in Section 3.3.

3.1 The discrete logarithm problem

Let G be a finite cyclic group of order p, written multiplicatively, and let g be a generator, so that

$$G = \{\, g^0, g^1, \ldots, g^{p-1} \,\} \ .$$

Since g has order p, the powers cycle with period p: in particular $g^p = g^0 = 1$, the identity of G, and more generally g^i depends only on i taken modulo p. The map $x \mapsto g^x$ is then a bijection from $\mathbb{Z}_p$[1] onto G, and it is much more than a bijection. Because $g^x g^y = g^{x+y}$ and the exponents wrap around modulo p, we have

$$g^x \cdot g^y = g^{\,(x+y) \bmod p} \ ,$$

so *multiplying* two group elements amounts to *adding* their exponents modulo p. The map $x \mapsto g^x$ is thus a group isomorphism from $(\mathbb{Z}_p, +)$ onto $(G, \times)$: the additive world of the exponents and the multiplicative world of the group elements are two faces of the same object. Under this dictionary the neutral elements 0 and 1 correspond, squaring an element

[1] Here and throughout, $\mathbb{Z}_p$ is the standard cryptographic shorthand for the ring of integers modulo p, that is, $\mathbb{Z}/p\mathbb{Z}$ (and likewise $\mathbb{Z}_q$ for $\mathbb{Z}/q\mathbb{Z}$).

doubles its exponent (i.e., $g^x \cdot g^x = g^{2x}$), and raising an element to the power a multiplies its exponent by a (i.e., $(g^x)^a = g^{ax}$), the last operation being the exponentiation on which everything below rests. A guiding principle in group-based cryptography follows: to design an attack or analyse a scheme, look at what happens in the exponents, where the group is simply $(\mathbb{Z}_p, +)$.

The single operation this correspondence does *not* render transparent is the passage from a group element back to its exponent: recovering a from g^a is the discrete logarithm problem. The forward direction, by contrast, is easy in every group of cryptographic interest. The exponent x has about $\log_2 p$ bits, and *square and multiply* computes g^x using $O(\log p)$ group operations rather than the naive $x-1$ multiplications. Writing $x = \sum_i x_i 2^i$ in binary, we have:

$$g^x = \prod_{i:\, x_i = 1} g^{2^i} \ .$$

The powers $g^{2^i} = (g^{2^{i-1}})^2$ are obtained by repeated squaring, so a single left-to-right scan of the bits of x suffices.

Definition 3.1 (Discrete logarithm problem, DLP)**.** Let G be a cyclic group of known prime order p with generator g. Given g and $h = g^a$ for a uniform $a \leftarrow\!\!\$\ U(\mathbb{Z}_p)$, the *discrete logarithm problem* is to recover a. The assumption that no efficient algorithm succeeds with non-negligible probability is the *discrete logarithm assumption*.

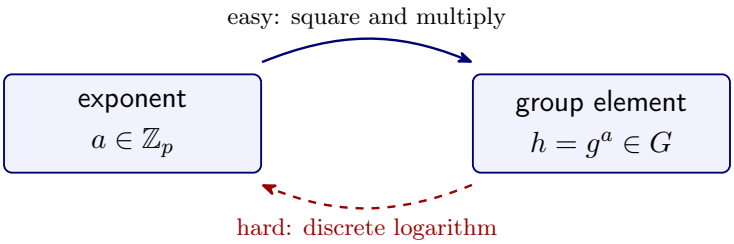

Figure 3.1: Exponentiation as a candidate one-way function. The forward map is efficient in every group used in practice; inverting it is the discrete logarithm problem.

Exponentiation is therefore a candidate *one-way function*: easy to compute, conjecturally hard to invert (Figure 3.1).

It is essential to realise that this hardness is not a property of the abstract group. Any two cyclic groups of the same prime order p are isomorphic, so as abstract structures they are indistinguishable; what differs is the concrete *representation* of their elements, and it is the representation that makes inversion easy or hard. A striking example is the additive group $(\mathbb{Z}/p\mathbb{Z}, +)$ itself, which is cyclic of order p. Taking the generator $g = 1$, the "exponentiation" $x \mapsto g \cdot x$ is just $x \mapsto x$, and even for a general generator g it is $x \mapsto g\,x \bmod p$; recovering the exponent from $h = g\,x$ then only requires one modular division, $x = h\,g^{-1} \bmod p$, which the extended Euclidean algorithm performs in polynomial time. In $(\mathbb{Z}/p\mathbb{Z}, +)$ the discrete logarithm is thus *trivial*, even though the group is a perfectly good cyclic group of order p. This is why we do not take $\mathbb{Z}/p\mathbb{Z}$ itself: a useful group must be represented so that multiplication is efficient while the isomorphism $x \mapsto g^x$ with the exponents is hard to invert.

3.1.1 Which group?

With that caveat in mind, the difficulty of the problem depends entirely on the concrete group in which it is posed. Three families have been used.

- **Multiplicative groups of prime fields.** The historical choice is a prime-order subgroup G of $\mathbb{Z}_q^\times$ for a prime q. One selects q so that $q-1$ has a large prime factor p and takes G to be the subgroup of order p. The group $\mathbb{Z}_q^\times$ is cyclic of order $q-1$, and a cyclic group has *exactly one* subgroup of each order dividing the group order, so this subgroup exists and is unique as soon as $p \mid q-1$. Here group elements are integers modulo q and multiplication is modular multiplication; the exponents live in $\mathbb{Z}_p$.
- **Multiplicative groups of other finite fields.** More generally, one may take a large prime-order subgroup of $(\mathbb{F}_q \setminus \{0\}, \times)$ for any finite field $\mathbb{F}_q$, for instance a field of characteristic two, $\mathbb{F}_{2^k}$. Such fields have convenient hardware arithmetic.
- **Elliptic curves.** Since the 1990s the preferred choice has been the group of rational points of an elliptic curve over a finite field. These groups support the same exponentiation-based schemes, with the group law playing the role of multiplication, while resisting the strongest known attacks, as discussed below.

3.1.2 Algorithms for the discrete logarithm

How hard is the problem? Any group admits generic attacks, which use only the group operations and ignore the representation of elements.

Exhaustive search computes $g^0, g^1, g^2, \ldots$ until the target h appears, using up to p group operations, which is prohibitive when p is large.

A *birthday* attack is quadratically faster.

Lemma 3.2 (Birthday discrete logarithm)**.** *There is a generic algorithm that solves the discrete logarithm problem in a group of prime order p using $O(\sqrt{p})$ group operations and $O(\sqrt{p})$ storage, and succeeds with constant probability.*

Proof. Let $h = g^a$; if h is the identity then $a = 0$, so assume $a \neq 0$. Put $t = \lceil 2\sqrt{p} \rceil$ (we may assume $t \leq p$). Fix $j_1, \ldots, j_t$ to be any t *pairwise distinct* elements of $\mathbb{Z}_p$ (their values are irrelevant, and their randomness is never used), and sample $i_1, \ldots, i_t$ independently and uniformly in $\mathbb{Z}_p$. Form the two lists

$$L = \left\{ h^{i_k} : 1 \leq k \leq t \right\} , \qquad R = \left\{ g \cdot h^{j_\ell} : 1 \leq \ell \leq t \right\} .$$

Suppose a collision $h^{i_k} = g\, h^{j_\ell}$ is found. Passing to the exponents, we have $g^{a i_k} = g^{\,1 + a j_\ell}$, and since g has order p this reads

$$a\,(i_k - j_\ell) \equiv 1 \pmod{p} .$$

The product $a\,(i_k - j_\ell)$ is congruent to 1; in particular $i_k - j_\ell \not\equiv 0$, and as p is prime this residue is invertible, so

$$a = (i_k - j_\ell)^{-1} \bmod p ,$$

which is the sought logarithm. The collision is located by sorting the two lists (or hashing one and probing with the other) in $O(t \log t)$ time.

It remains to argue that a collision is likely. Since $a \neq 0$, the map $j \mapsto g\, h^j = g^{1+aj}$ is a bijection from $\mathbb{Z}_p$ onto G, so the distinct j_ℓ give a set R of *exactly* t distinct group

elements; this is all we use about the j_ℓ. The map $i \mapsto h^i = g^{ai}$ is also a bijection, so for a uniform i the value h^i is uniform on G; hence L is a list of t *independent* uniform elements of G. Each of them lands outside the fixed t-element set R with probability exactly $1 - t/p$, so we obtain, by independence,

$$\Pr[\, L \cap R = \varnothing \,] = \left(1 - \tfrac{t}{p}\right)^t \le e^{-t^2/p} \le e^{-4} < 0.02 \ .$$

Therefore, a collision occurs with probability at least 0.98, and the algorithm uses $O(t) = O(\sqrt{p})$ exponentiations and $O(\sqrt{p})$ storage. Independent repetitions drive the failure probability to zero. □

The deterministic *baby-step giant-step* method achieves the same $O(\sqrt{p})$ time by writing $a = im + j$ with $m = \lceil \sqrt{p} \rceil$ and matching $h\, g^{-j}$ against $(g^m)^i$; Pollard's rho method reaches the same running time with only constant storage. Either way, the birthday bound is the generic barrier: in a group where nothing but the group law can be exploited, roughly $\sqrt{p}$ operations are both sufficient and necessary. Consequently, to reach a 2^λ security level against generic attacks one needs $\log_2 p \approx 2\lambda$.

The concrete groups of Section 3.1.1 are, however, not generic: their representation invites specialised algorithms that do far better.

- For a subgroup of $\mathbb{Z}_q^\times$, the *generalised number field sieve* solves the discrete logarithm in *sub-exponential* time

  $$\exp\Big(\widetilde{O}\big((\log p)^{1/3}\big)\Big) = L_p\big[\tfrac{1}{3},\, c\big] \ ,$$

  with the standard notation $L_p[\alpha, c] = \exp\big((c + o(1))(\log p)^\alpha (\log\log p)^{1-\alpha}\big)$. Because the exponent grows only as $(\log p)^{1/3}$, reaching a security level of 2^λ requires $\log p = \Omega(\lambda^3)$, far larger than the $\log_2 p \approx 2\lambda$ that suffices against generic attacks. Where this frontier currently lies is a matter of concrete cryptanalysis, and factoring records, which track discrete-logarithm records closely, give the sharpest data points: the 896-bit challenge modulus RSA-896 was factored on 19 September 2026, at an estimated cost on the order of 400,000 USD.

- For $\mathbb{F}_{2^k}$ and other small-characteristic fields, there are also sub-exponential algorithms, and in fact a *quasi-polynomial*-time algorithm of Barbulescu, Gaudry, Joux, and Thomé (2014). Small-characteristic fields are therefore a poor choice for discrete-logarithm cryptography.

- For elliptic curves, no sub-exponential algorithm is known, except for certain special families of curves. The best attacks on a well-chosen curve are the generic $O(\sqrt{p})$ methods above. This is why elliptic curves permit much smaller keys, with $\log_2 p \approx 2\lambda$ rather than $\Omega(\lambda^3)$, and correspondingly faster arithmetic.

Shor's algorithm computes discrete logarithms (and factors integers) in *quantum polynomial time*, so the discrete logarithm problem in any cyclic group, elliptic curves included, is broken once a large-scale quantum computer exists. This is the principal motivation for the lattice assumptions of Section 3.2, for which no such quantum collapse is known.

3.1.3 Problem variants: CDH and DDH

Cryptographic protocols rarely rely on the discrete logarithm problem directly. What they usually rest on is a stronger assumption: that the value g^{ab}, formed from two public

powers g^a and g^b, be hard to compute from them, or even just hard to tell apart from a random group element. The next two problems formalise these two requirements.

Definition 3.3 (Computational Diffie–Hellman, CDH)**.** Given G, p, g, and g^a, g^b for uniform and independent $a, b \leftarrow\!\!\shortmid U(\mathbb{Z}_p)$, compute g^{ab}.

Definition 3.4 (Decisional Diffie–Hellman, DDH)**.** Given G, p, g, distinguish the two distributions

$$\left(g^a,\, g^b,\, g^{ab}\right) \qquad \text{and} \qquad \left(g^a,\, g^b,\, g^c\right)\ ,$$

where $a, b, c \leftarrow\!\!\shortmid U(\mathbb{Z}_p)$ are uniform and independent. The *DDH assumption* states that these two distributions are computationally indistinguishable.

A triple of the first kind is called a *Diffie–Hellman triple*; the second distribution is simply three independent uniform group elements. The three problems are ordered by difficulty, and the ordering is proved by two elementary reductions (Figure 3.2).

Lemma 3.5 (CDH reduces to DLP)**.** *An efficient algorithm for DLP yields an efficient algorithm for CDH. Equivalently, if CDH is hard, then so is DLP.*

Proof. Let $\mathcal{S}$ solve DLP. On a CDH instance (g, g^a, g^b), run $\mathcal{S}$ on (g, g^a) to recover a, and output $(g^b)^a = g^{ab}$. The wrapper performs one call to $\mathcal{S}$ and one exponentiation. □

Lemma 3.6 (DDH reduces to CDH)**.** *An efficient algorithm for CDH yields an efficient distinguisher for DDH with advantage $1 - 1/p$. Equivalently, if DDH is hard, then so is CDH.*

Proof. Let $\mathcal{S}$ solve CDH. On a DDH challenge (g^a, g^b, T), compute $z = \mathcal{S}(g^a, g^b)$, which equals g^{ab}, and output 1 if and only if $T = z$. On a Diffie–Hellman triple $T = g^{ab} = z$, the distinguisher outputs 1 with probability 1. On a random triple $T = g^c$ with c uniform and independent, we have $T = z$ only when $c \equiv ab \pmod{p}$, which happens with probability $1/p$. The advantage is therefore $1 - 1/p$. □

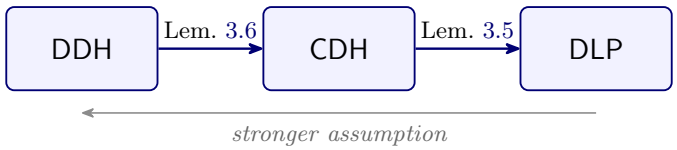

Figure 3.2: The Diffie–Hellman hierarchy. Each arrow $X \to Y$ means "an algorithm for Y breaks X", so assuming X hard is at least as strong as assuming Y hard: DDH hardness implies CDH hardness, which implies DLP hardness.

Assuming DDH is thus the strongest of the three assumptions and assuming DLP the weakest. The gaps between them are genuine, and can even be extreme: there are groups for which DDH is *easy* while CDH still appears hard. This happens when the group carries an efficiently computable bilinear *pairing* $e\colon G \times G \to G_T$ with $e(g^x, g^y) = e(g, g)^{xy}$: one can then test whether (g^a, g^b, T) is a Diffie–Hellman triple by checking $e(g^a, g^b) = e(g, T)$, which decides DDH, yet computing g^{ab} remains conjecturally hard. Far from being a defect, such groups are the foundation of *pairing-based cryptography*, which builds identity-based encryption, short signatures, and much else.

3.2 The learning-with-errors problem

The discrete-logarithm family is efficient and well understood, but it is broken by quantum computers. A very different assumption, introduced by Regev in 2005, is stated purely in terms of linear algebra over the ring $\mathbb{Z}_q$ and has so far resisted quantum attacks. Its hardness comes from adding small *errors* to an otherwise solvable linear system.

Definition 3.7 (Learning with errors, $\mathsf{LWE}_{m,n,q,B}$)**.** Fix dimensions $m \geq n \geq 2$, a modulus $q \geq 2$, and a bound $B < q/2$. The $\mathsf{LWE}_{m,n,q,B}$ assumption states that the distributions

$$(\mathbf{A},\, \mathbf{u}) \qquad \text{and} \qquad (\mathbf{A},\, \mathbf{A}\mathbf{s} + \mathbf{e})$$

are computationally indistinguishable, where $\mathbf{A} \hookleftarrow U\big(\mathbb{Z}_q^{m\times n}\big)$, $\mathbf{u} \hookleftarrow U\big(\mathbb{Z}_q^{m}\big)$, $\mathbf{s} \hookleftarrow U\big(\mathbb{Z}_q^{n}\big)$, and the error vector $\mathbf{e} \hookleftarrow U\big((-B, B]^m\big)$ has small integer entries reduced modulo q. All arithmetic is over $\mathbb{Z}_q$.

The problem is best read off a picture (Figure 3.3): a tall random matrix $\mathbf{A}$ multiplies a short secret $\mathbf{s}$, and a small error $\mathbf{e}$ is added; the task is to tell the perturbed product $\mathbf{A}\mathbf{s} + \mathbf{e}$ apart from an unrelated uniform vector $\mathbf{u}$, given $\mathbf{A}$.

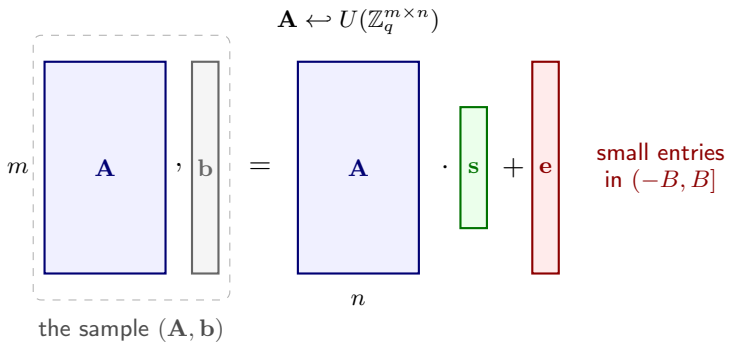

Figure 3.3: The learning-with-errors distribution. A uniform matrix $\mathbf{A}$ times a uniform secret $\mathbf{s}$, perturbed by a short error $\mathbf{e}$, yields $\mathbf{b} = \mathbf{A}\mathbf{s}+\mathbf{e} \in \mathbb{Z}_q^m$. LWE asks to distinguish $(\mathbf{A}, \mathbf{b})$ from $(\mathbf{A}, \mathbf{u})$ with $\mathbf{u}$ uniform.

The role of the error bound B deserves comment, since it interpolates between two degenerate extremes.

- If $B = 0$ there is no error and the problem is *easy*: one is handed a consistent linear system $\mathbf{b} = \mathbf{A}\mathbf{s}$ over $\mathbb{Z}_q$ and solves it for $\mathbf{s}$. Recovery works whenever $\mathbf{A}$ is injective as a map $\mathbb{Z}_q^n \to \mathbb{Z}_q^m$. When q is prime, the ring $\mathbb{Z}_q$ is a field and $m = n$ already suffices: a uniform square $\mathbf{A}$ is invertible modulo q with probability $\prod_{j=1}^{n}(1 - q^{-j}) \approx 1 - 1/q$, which is close to 1 unless q is very small. For composite q, and in particular smooth q, this is no longer automatic (modulo a small prime factor of q, a square matrix is singular with constant probability), so one takes m a little larger than n; injectivity modulo q then fails only with probability decaying like $p^{-(m-n)}$ in the smallest prime factor p of q. Once $\mathbf{s}$ is found one checks whether $\mathbf{b} = \mathbf{A}\mathbf{s}$: a genuine LWE sample is recognised with certainty, while a uniform $\mathbf{u}$ is consistent only with negligible probability.
- If $B = q/2$ the problem is *trivially hard*, but for an uninteresting reason: the error $\mathbf{e}$ is then (essentially) uniform over $\mathbb{Z}_q^m$, so $\mathbf{A}\mathbf{s} + \mathbf{e}$ is itself uniform and *identical* to the

distribution of $\mathbf{u}$. The two distributions coincide, and no adversary, however powerful, can separate them. Security here is information-theoretic and useless, because the sample carries no information about $\mathbf{s}$.

- The interesting regime lies in between. For B ranging from a constant up to (but below) $q/2$, the problem appears genuinely hard while the samples still determine $\mathbf{s}$ information-theoretically. This is the range used in cryptography.

The middle regime is exactly where $\mathbf{As} + \mathbf{e}$ can serve as a source of pseudorandomness, and a counting argument makes the point (this is the observation exploited in Section 3.3). The uniform vector $\mathbf{u}$ carries $m \log_2 q$ bits of entropy. The vector $\mathbf{As} + \mathbf{e}$, on the other hand, is a deterministic function of the pair $(\mathbf{s}, \mathbf{e})$, which carries only

$$\approx n \log_2 q + m\,(1 + \log_2 B)$$

bits: $n \log_2 q$ from the secret and about $\log_2(2B)$ per coordinate of the error. When $m - n$ is large, the second quantity is much smaller than $m \log_2 q$, so $\mathbf{As} + \mathbf{e}$ occupies a tiny, sparse region of $\mathbb{Z}_q^m$, exactly as the image of a generator was sparse in Chapter 2. That the two can nonetheless not be told apart efficiently is the content of the assumption.

A word on algorithms and parameters. The naive attacks are exponential: exhaustive search over the secret costs q^n, and guessing the first n error coordinates and solving costs $\approx (2B)^n$. A more clever, purely algebraic attack, due to Arora and Ge, linearises a polynomial system. Writing $\mathbf{a}_i$ for the i-th row of $\mathbf{A}$ and b_i for the i-th entry of $\mathbf{b}$, the error $e_i = b_i - \langle \mathbf{a}_i, \mathbf{s} \rangle$ takes one of the $2B$ values in $(-B, B]$, so

$$\prod_{v=-B+1}^{B} \big(\langle \mathbf{a}_i, \mathbf{s} \rangle - b_i + v\big) \equiv 0 \pmod{q} \ ,$$

a polynomial equation of degree $2B$ in the entries of $\mathbf{s}$. Treating every monomial of degree at most $2B$ as a fresh unknown turns these into *linear* equations, which can be solved once $m \gtrsim n^{2B}$. For $B \geq n$ and arbitrary m, the best known algorithm runs in time roughly

$$\exp\left[O\left(\frac{n \log q}{\log^2(q/B)} \log\left(\frac{n \log q}{\log^2(q/B)}\right)\right)\right].$$

Crucially, no quantum algorithm improves on this beyond reducing the constant hidden in the $O(\cdot)$. A typical parameter choice takes $n = \lambda$ and $B < q = \mathrm{poly}(n)$, which places one squarely in the hard regime.

Definition 3.7 is a deliberately simplified, pedagogical variant. In the standard formulation, the adversary may request arbitrarily many rows, i.e., increase m at will, and the error is drawn from a discrete Gaussian rather than a uniform distribution on an interval (Figure 3.4). The two versions reduce to each other in both directions, up to losses in the parameters, so nothing essential is lost by working with the simplified one. With Gaussian errors one can moreover manufacture additional samples out of a fixed batch, by taking random combinations of the given rows, again at the price of some parameter degradation.

3.3 Building a pseudorandom generator

We now turn the hardness assumptions of this chapter into secure pseudorandom generators. Each family gives a construction whose security is, almost by its very shape, a rephrasing of the underlying assumption.

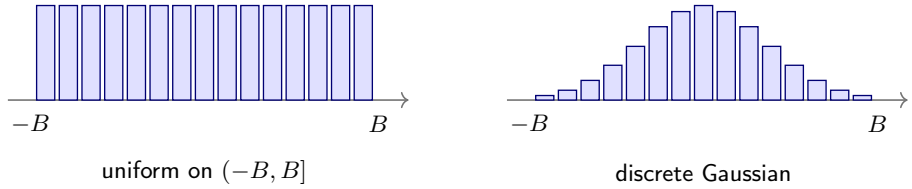

Figure 3.4: Two error distributions on the integers: the uniform distribution on $(-B, B]$ used in Definition 3.7 (left), and the discrete Gaussian of the standard formulation (right).

3.3.1 From decisional Diffie–Hellman

The decisional Diffie–Hellman assumption is a pseudorandomness statement almost verbatim. Fix public parameters: a cyclic group G of prime order p and a generator g, and recall that $x \mapsto g^x$ is a bijection between $\mathbb{Z}_p$ and G. Define

$$G_{\mathrm{DH}} \colon\ \mathbb{Z}_p \times \mathbb{Z}_p \longrightarrow G^3 \ , \qquad G_{\mathrm{DH}}(a, b) = \big(g^a,\, g^b,\, g^{ab}\big) \ .$$

The seed is the pair (a, b) of exponents, drawn uniformly from $\mathbb{Z}_p \times \mathbb{Z}_p$, and the output is the corresponding Diffie–Hellman triple (g^a, g^b, g^{ab}). The generator simply publishes the first two powers together with the "combined" power g^{ab}; a uniform element of G^3 is instead a triple (g^a, g^b, g^c) with an unrelated third component.

Before stating the result, a word on what "pseudorandom" means here. The generator of Chapter 2 maps *bits* to *bits*, whereas G_{DH} maps pairs in $\mathbb{Z}_p^2$ to triples in G^3, so we read security in the way natural for these alphabets: the output should be indistinguishable from the uniform distribution over its own range G^3. Casting G_{DH} as a genuine bit-to-bit generator would require two encodings, neither entirely free. On the input side, a uniform seed in $\{0,1\}^s$ must be turned into a uniform pair $(a, b) \in \mathbb{Z}_p^2$, for instance by reducing $s \gg 2 \log_2 p$ uniform bits modulo p (a negligible bias) or by rejection sampling. On the output side, group elements must be encoded as bitstrings. But a prime-order subgroup of $\mathbb{Z}_q^\times$ (if we make that choice of group) has only p of the q possible integer encodings, so the encoded output is *not* uniform over bitstrings and may even be recognisable as a subgroup element. A clean bit-to-bit generator from group assumptions therefore takes extra care, typically heuristically, by relying on a hash function. Over the native alphabets, by contrast, security needs no work at all: telling a Diffie–Hellman triple from a uniform triple is *by definition* the DDH problem.

Theorem 3.8 (A pseudorandom generator from DDH)**.** *Under the DDH assumption in (G, g, p), the map G_{DH} is a secure pseudorandom generator: the distributions $G_{\mathrm{DH}}\big(U(\mathbb{Z}_p^2)\big)$ and $U(G^3)$ are computationally indistinguishable.*

Proof. We first observe that the "uniform" target of a generator is exactly the random-triple distribution of DDH. Since $x \mapsto g^x$ is a bijection from $\mathbb{Z}_p$ to G, if $a, b, c \hookleftarrow U(\mathbb{Z}_p)$ are uniform and independent then g^a, g^b, g^c are uniform and independent in G; hence

$$U(G^3) \text{ is the distribution of } \big(g^a, g^b, g^c\big) \ .$$

On the other side, by construction $G_{\mathrm{DH}}\big(U(\mathbb{Z}_p^2)\big)$ is the distribution of $\big(g^a, g^b, g^{ab}\big)$ for uniform independent a, b. Thus the two distributions to be separated are precisely the Diffie–Hellman triple and the random triple of Definition 3.4.

The reduction is therefore the identity on adversaries (Figure 3.5). Given any efficient distinguisher $\mathcal{A} \colon G^3 \to \{0,1\}$ against the generator, the same $\mathcal{A}$ is a DDH distinguisher, and

$$\mathsf{Adv}_{\mathcal{A}}\big(G_{\mathrm{DH}}(U(\mathbb{Z}_p^2)),\, U(G^3)\big) = \mathsf{Adv}_{\mathcal{A}}\big((g^a, g^b, g^{ab}),\, (g^a, g^b, g^c)\big) \ ,$$

with identical running time. By the DDH assumption the right-hand side is negligible, so the left-hand side is too; conversely a successful attack on the generator is a successful attack on DDH. Hence G_{DH} is secure if and only if DDH holds. □

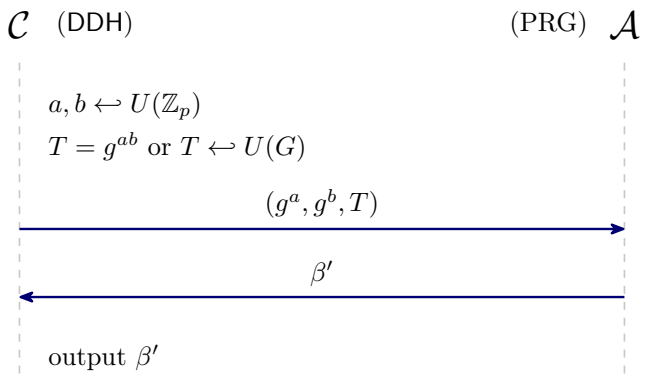

Figure 3.5: The reduction of Theorem 3.8. A DDH challenge (g^a, g^b, T) is handed verbatim to the generator's distinguisher: a Diffie–Hellman triple is a pseudorandom output, a random triple is a uniform one, so the two games coincide.

The generator G_{DH} is genuinely expanding. Its seed (a, b) encodes $2\log_2 p$ bits, while its output, three elements of a group of order p, carries $3\log_2 p$ bits: the construction converts $2\log_2 p$ bits of true randomness into $3\log_2 p$ bits of *pseudo*-randomness, an expansion factor of $3/2$.

3.3.2 From learning with errors

The learning-with-errors assumption yields a generator in exactly the same spirit, and one that lives natively over the integers. Fix public parameters (m, n, q, B) together with a uniform matrix $\mathbf{A} \leftarrow U(\mathbb{Z}_q^{m\times n})$, and define

$$G_{\mathrm{LWE}}\colon\ \mathbb{Z}_q^n \times (-B, B]^m \longrightarrow \mathbb{Z}_q^m\ , \qquad G_{\mathrm{LWE}}(\mathbf{s}, \mathbf{e}) = \mathbf{A}\mathbf{s} + \mathbf{e}\ .$$

The seed is the pair $(\mathbf{s}, \mathbf{e})$: a secret $\mathbf{s} \leftarrow U(\mathbb{Z}_q^n)$ together with a short error $\mathbf{e} \leftarrow U\big((-B, B]^m\big)$. The output is the perturbed product $\mathbf{A}\mathbf{s} + \mathbf{e} \in \mathbb{Z}_q^m$.

Theorem 3.9 (A pseudorandom generator from LWE)**.** *Under the $\mathsf{LWE}_{m,n,q,B}$ assumption, the map G_{LWE} is a secure pseudorandom generator: the distributions $\big(\mathbf{A},\, G_{\mathrm{LWE}}(\mathbf{s}, \mathbf{e})\big)$, for $\mathbf{s} \leftarrow U(\mathbb{Z}_q^n)$ and $\mathbf{e} \leftarrow U((-B, B]^m)$, and $\big(\mathbf{A},\, U(\mathbb{Z}_q^m)\big)$ are computationally indistinguishable.*

The proof is again the identity reduction and we omit it: the two distributions $(\mathbf{A}, \mathbf{A}\mathbf{s}+\mathbf{e})$ and $(\mathbf{A}, \mathbf{u})$ are exactly those of Definition 3.7, so a distinguisher against G_{LWE} *is* a distinguisher against LWE, with the same advantage.

For the map to expand, its output must carry more bits than its seed. The seed $(\mathbf{s}, \mathbf{e})$ has entropy $\approx n\log_2 q + m\,(1 + \log_2 B)$ bits, whereas the output lies in $\mathbb{Z}_q^m$ and is written in $m\log_2 q$ bits; taking m large enough relative to n and $\log_2 B$ makes the output the longer of the two, so the generator stretches its seed.

Here the encoding friction of the previous subsection is absent, which is one reason lattice assumptions are such convenient building blocks. Seeds and outputs are already vectors of integers modulo q, encoded coordinate by coordinate into $\lceil \log_2 q \rceil$ bits apiece; the only mismatch is that q need not be a power of two, so obtaining uniform bits from a

coordinate calls for rejection sampling. This loses a small fraction of the outputs, which is harmless once m is large enough, and disappears entirely when q is chosen a power of two. There is nothing like the sparse-subgroup obstruction met for $\mathbb{Z}_q^\times$, where only a p/q fraction of encodings were valid.