

# Cryptography and Security

*Based on the lectures of  
Alain PASSELÈGUE and Damien STEHLÉ  
Notes written by Hugo SALOU*



*September 19, 2025*

# Contents

- 1 Introduction: What is crypto? 4**
  - 1.1 Some basic examples. . . . . 4
    - 1.1.1 How to check that two distant files are the same? 4
    - 1.1.2 Secure communication over the Internet. . . . . 5
  - 1.2 Summary. . . . . 5
    - 1.2.1 TL;DR. . . . . 6
  - 1.3 Some more applications of crypto. . . . . 6
  - 1.4 Conclusion. . . . . 7
- 2 A perfectly secure symmetric encryption scheme: ONE-TIME PAD 9**
  - 2.1 Pseudo-random generators. . . . . 12
    - 2.1.1 How to get PRGs? Cryptographic assumptions. 20

This course will mostly be about crypto, with a little amount of security. Before we start, some administrative informations: the class is on Friday at 1:30PM and the tutorials are on Monday at 8AM. There will be two written exams.

This course will be split in two parts: the first third will be on *symmetric crypto* and the other two thirds will be on *public key crypto*.

# 1 Introduction: What is crypto?

Crypto can be understood as *cryptography* or *cryptology*; the two are distinct notions, but this won't matter to us in this course.

*Crypto is the science of securing data  
against adversaries.*

This is not just encryption, not just data transit (*e.g.* managing data), not all information security. Security is a lot broader than crypto: it contains crypto but also social engineering, finding implementation bugs, risk management, system security.

This first chapter will be more informal, to give a high-level overview of crypto.

## 1.1 Some basic examples.

### 1.1.1 How to check that two distant files are the same?

Instead of comparing the two files byte by byte, we can compare *cryptographic hashes*. They are a lot smaller in size compared to the original data. Some famous examples of cryptographic hash functions are SHA-256, SHA-2, SHA-3 or MD-5 (this last one is outdated).

### 1.1.2 Secure communication over the Internet.

HTTPS is a secure protocol for browsing the Internet. The “S” in HTTPS means *secure*. The underlying protocol is called TLS/SSL. When browsing, for example, Google, we want that our communication is secured against *eavesdropper*; usually this means encrypting the communications. This communication goes in multiple parts.

1. **Authentication.** Alice asks Bob some kind of identification, and Bob will send Alice a certificate of authentication.
2. **Handshake protocol.** Alice and Bob then create a shared secret key.
3. **Encrypted communication.** Finally, Alice and Bob can communicate without risking being eavesdropped. This is done with symmetric encryption.

All of these steps can be done while being eavesdropped, and this does not compromise the communication between Alice and Bob. This course will cover all the ingredients for this protocol:

1. Digital signature (for step 1)
2. Public-key encryption (for step 2)
3. Symmetric encryption (for step 3).

We will see all of this in the reverse order.

## 1.2 Summary.

The *core* of crypto is the three following ideas.

**Authenticity.** Am I certain to whom I am talking?

**Integrity.** Am I certain what I am saying is what the other person hears?

**Confidentiality.** Am I certain that our communication is private?

Encryption, what most people have in mind when talking about crypto, sits in the third step “Confidentiality.” However, without the other two, there is no point to encryption for secure communication.

As we can see from SSL/TSL, defining a secure protocol is complicated. We should define: functionality, security properties, who is the adversary, what the adversary is capable of. This last question can be divided in two subquestions:

- ▷ What is its computational resources?

Crypto experts try to define protocols that require at least  $2^{128}$  operations for the adversary to access your secured communication. Even with a limited computational power on the user side, the adversary would require exponentially more computational power.

- ▷ What kind of access does it have to your data?

Having access to someone’s phone or not having access to it makes a lot of difference in terms of security.

### 1.2.1 TL;DR.

Protocols are complex: we break them into simpler pieces called “cryptographic primitives” (*e.g.* digital signatures, hash functions, symmetric encryption, *etc*). The security of a complex system can be reduced to the security of its building blocks. One of the goal of this course is to define all these primitives and prove that they are secure.

## 1.3 Some more applications of crypto.

VPNs or the TOR network are more common examples of applications of crypto. One important part is to *define* what you are trying to hide. For example, your internet provider can still see, even while using a VPN, that you are watching videos as a lot of data is flowing (the ISP does not know the exact videos, but still know you are watching something).

Another very common example is cryptocurrencies, for example the famous BitCoin. Hashes and digital signatures are used a lot with these cryptocurrencies. E-cash is a little different, but with the same idea. How can you check that someone did not spend some amount of money twice? Guaranteeing that you cannot duplicate money.

Then, crypto is also used by E-voting. An E-voting system needs to make sure of a few properties: assure that there is no ballot stuffing, verifiability at the end of the protocol, anonymity of the vote, authentication.

Also, the French app *TousAntiCovid* from the Covid pandemic used crypto. Here, functionality is important: in itself the protocol is not sure, and can be easily manipulated to leak its data, no matter how perfectly encrypted the data is.

Disk encryption is also using crypto. Is there a way to compute over encrypted data? This can be used as, for examples, LLMs such as ChatGPT are manipulating lots of data, and some of it can be confidential.

## 1.4 Conclusion.

Cryptography is a science that can be split in five steps:

1. Define functionality;
2. Define the goal of the attacker;
3. Define its capabilities;
4. Propose a realisation;
5. Prove the security of the realisation.

This last step is usually done by proving that, if an attack succeeds in breaking the protocol, then some unrealistic (computational) conditions are true. This is done via reduction, *e.g.* prove that attacking implies being able to solve famous intractable problem, for example the FACTORING problem. **NP**-complete problems, like SAT, are not always the answer as it is difficult to get a SAT instance that

is hard to solve; to get a hard-to-solve instance for FACTORING, you only need to pick two large prime numbers which is a lot easier to do.

Crypto is related to complexity theory and the intractable problems we rely on are usually algebraic problems. Beautiful math is involved.

Crypto is not an art! Non-public protocols must be considered insecure. Protocols should be public and sources of security should be clearly identified and concentrated. If the protocol is not open-source, then it usually means that something sketchy is going on. The implementation can be private as a lot of hard work can be done in how to implement this protocol effectively, but the protocol should be public.



## 2 A perfectly secure symmetric encryption scheme: ONE-TIME PAD

This encryption scheme achieves information-theoretic security.

**Definition 2.1 (Symmetric encryption).** Let  $\mathcal{K}$  be a key space,  $\mathcal{P}$  be a plain-text space and let  $\mathcal{C}$  be a ciphertext space. These three spaces are finite spaces.

A *symmetric encryption* scheme over  $(\mathcal{K}, \mathcal{P}, \mathcal{C})$  is a tuple of three algorithms (KeyGen, Enc, Dec) :

- ▷ KeyGen provides a sample  $k$  of  $\mathcal{K}$ ;
- ▷  $\text{Enc} : \mathcal{K} \times \mathcal{P} \rightarrow \mathcal{C}$ ;
- ▷  $\text{Dec} : \mathcal{K} \times \mathcal{C} \rightarrow \mathcal{P}$ .

Without loss of generality, we will assume that  $\text{im Enc} = \mathcal{C}$ . We want to ensure **Correctness**: for any key  $k \in \mathcal{K}$  and message  $m \in \mathcal{P}$ , we have that:

$$\text{Dec}(k, \text{Enc}(k, m)) = m.$$

The elements  $m$  and  $k$  are independent random variables and all the elements in  $\mathcal{K}$  and  $\mathcal{P}$  have non-zero probability.

**Remark 2.1.** The algorithm Enc could (and should<sup>1</sup>) be probabilistic. However, the algorithm Dec is deterministic.

So far, we did not talk about efficiency of these algorithms.

**Definition 2.2 (Shannon, 1949).** A symmetric encryption scheme is said to have *perfect security* whenever, for any  $\bar{m}$  and any  $\bar{c}$ ,

$$\Pr_{k,m}[m = \bar{m} \mid \text{Enc}_k(m) = \bar{c}] = \Pr_m[m = \bar{m}].$$

The intuition is that knowing the encrypted message tells me *nothing* about the message.

**Lemma 2.1 (Shannon).** Given a symmetric encryption scheme (KeyGen, Enc, Dec) has perfect security then  $|\mathcal{K}| \geq |\mathcal{P}|$ .

**Proof.** Let  $\bar{c} \in \mathcal{C}$  and define

$$\mathcal{S} := \{\bar{m} \in \mathcal{P} \mid \exists \bar{k} \in \mathcal{K}, \bar{m} = \text{Dec}(\bar{k}, \bar{c})\}.$$

Let  $N := |\mathcal{S}|$ . We have that  $N \leq |\mathcal{K}|$  as Dec is deterministic. We also have that  $N \leq |\mathcal{P}|$  as  $\mathcal{S} \subseteq \mathcal{P}$ . Finally, assume  $N < |\mathcal{P}|$ . This means, there exists  $\bar{m} \in \mathcal{P}$  such that  $\bar{m} \notin \mathcal{S}$ . Then,

$$\Pr[m = \bar{m} \mid \text{Enc}_k(m) = \bar{c}] = 0,$$

but by assumption,  $\Pr[m = \bar{m}] \neq 0$ . So this is not a perfectly secure scheme. We can conclude that

$$N = |\mathcal{P}| \leq |\mathcal{K}|.$$

□

<sup>1</sup>If the algorithm is deterministic, if we see two identical ciphers we know that the messages are identical, and this can be seen as a vulnerability of this protocol.

**Example 2.1 (One-Time PAD).** Let  $\mathcal{K} = \mathcal{C} = \mathcal{P} = \{0, 1\}^\ell$ . Here are the algorithms used:

- ▷ KeyGen samples from  $\mathcal{U}(\{0, 1\}^\ell)$ .
- ▷ Enc( $k, m$ ) we compute the XOR  $c = m \oplus k$ .
- ▷ Dec( $k, m$ ) we compute the XOR  $m = c \oplus k$ .

**Theorem 2.1.** The One-Time PAD is a perfectly-secure symmetric encryption.

**Proof. Correctness.** We have that

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

**Security.** We have, by independence of  $m$  and  $k$  we have that

$$\begin{aligned} \Pr[m = \bar{m} \mid \text{Enc}(k, m) = \bar{c}] &= \Pr[m = \bar{m} \mid k \oplus m = \bar{c}] \\ &= \Pr[m = \bar{m}]. \end{aligned}$$

□

**Remark 2.2.** This example is not practical:

- ▷ keys need to be larger than the message;
- ▷ you cannot encrypt twice: for example,  $c_1 = m_1 \oplus k$  and  $c_2 = m_2 \oplus k$ , then we have  $c_1 \oplus c_2 = m_1 \oplus m_2$ .

This last part is why that protocol is called a *One-Time secure encryption*.

We want to be able to encrypt arbitrarily long messages! We will have to make a trade-off and we choose to not care about *perfect* security. Why? In real life, we don't care about proving that something is proven to be absolutely infeasible, we only want to believe it is

infeasible in practice.

**Computational complexity** is sufficient in practice.

Let us be more precise in the next section.

## 2.1 Pseudo-random generators.

**Definition 2.3.** Let  $\mathcal{D}_0$  and  $\mathcal{D}_1$  be two distributions over  $\{0, 1\}^n$ .

An algorithm  $\mathcal{A} : \{0, 1\}^n \rightarrow \{0, 1\}$  is called a *distinguisher* between  $\mathcal{D}_0$  and  $\mathcal{D}_1$ . We define its *distinguishing advantage* as:

$$\text{Adv}_{\mathcal{A}} := \left| \underbrace{\Pr_{x \leftarrow \mathcal{D}_1} [\mathcal{A}(X) = 1]}_{\text{probability of being right}} - \underbrace{\Pr_{x \leftarrow \mathcal{D}_0} [\mathcal{A}(X) = 1]}_{\text{probability of being mistaken}} \right|.$$

We say that  $\mathcal{D}_0$  and  $\mathcal{D}_1$  are *computationally indistinguishable* if for any efficient distinguisher  $\mathcal{A}$  its advantage  $\text{Adv}_{\mathcal{A}}$  is small. We will write, in this case,  $\mathcal{D}_1 \simeq^c \mathcal{D}_2$ .

This definition is not very formal yet, we have not defined “efficient” and “small.” This can be formalized by introducing a parameter  $\lambda \in \mathbb{N}$  called the *security parameter*.

**Definition 2.4.** Let  $(\mathcal{D}_{0,\lambda})_{\lambda \in \mathbb{N}}$  and  $(\mathcal{D}_{1,\lambda})_{\lambda \in \mathbb{N}}$  be two distributions over  $\{0, 1\}^{n(\lambda)}$  for a non-decreasing polynomial  $n(\lambda)$ . The value of  $\lambda \in \mathbb{N}$  is called the *security parameter*.

An algorithm  $\mathcal{A} : \{0, 1\}^{n(\lambda)} \rightarrow \{0, 1\}$  is called a *distinguisher* between the distributions  $\mathcal{D}_{0,\lambda}$  and  $\mathcal{D}_{1,\lambda}$ . We define its *distinguishing advantage* as:

$$\text{Adv}_{\mathcal{A}}(\lambda) := \left| \underbrace{\Pr_{x \leftarrow \mathcal{D}_{1,\lambda}} [\mathcal{A}(X) = 1]}_{\text{probability of being right}} - \underbrace{\Pr_{x \leftarrow \mathcal{D}_{0,\lambda}} [\mathcal{A}(X) = 1]}_{\text{probability of being mistaken}} \right|.$$

We say that  $\mathcal{D}_{0,\lambda}$  and  $\mathcal{D}_{1,\lambda}$  are *computationally indistinguishable* if for any distinguisher  $\mathcal{A}$  running in  $O(\lambda^c)$  for some  $c > 0$ <sup>2</sup> its advantage  $\text{Adv}_{\mathcal{A}}$  is a  $o(1/\lambda^c)$  for some  $c > 0$ .<sup>3</sup>

Our goal now is to extend the One-Time PAD to messages  $m$  larger than the key  $k$ . We want to construct some function  $G$  that takes as input the key  $k \in \{0, 1\}^n$  and expand it to a string  $G(k) \in \{0, 1\}^\ell$  for some  $\ell > n$  that is computationally hard to distinguish from a uniform random string. This is called a *PGR* or *pseudo-random generator*.

**Definition 2.5.** A *pseudo-random generator* is a pair of poly-time algorithms  $(\text{Setup}, G)$  such that:

- ▷  $\text{Setup}$  is an algorithm that takes as input a security parameter  $\lambda$  (taken as a string  $1^\lambda$  of length  $\lambda$ , *i.e.* we write  $\lambda$  in unary) and returns a public parameter;
- ▷  $G_\lambda : \{0, 1\}^{n(\lambda)} \rightarrow \{0, 1\}^{\ell(\lambda)}$  is an algorithm which takes a string  $k$  of length  $n(\lambda)$  and return a string  $G(k)$  of length  $\ell(\lambda)$  with  $\ell(\lambda) > n(\lambda)$ .

such that

- ▷  $G$  is deterministic;
- ▷  $\ell(\lambda) > n(\lambda)$  (we say that it is *expanding*)
- ▷ the distributions  $\{\mathcal{U}(\{0, 1\}^{\ell(\lambda)})\}_{\lambda \in \mathbb{N}}$  and  $\{G(\mathcal{U}(\{0, 1\}^{n(\lambda)}))\}_{\lambda \in \mathbb{N}}$  are computationally indistinguishable (we call it *pseudo-randomness*).

Another way of defining a pseudo-random generator is with *unpredictability* instead of *pseudo-randomness*.

**Definition 2.6.** This is the same definition as before but replacing pseudo-randomness with *unpredictability*.

<sup>2</sup>This means it is polynomial in  $\lambda$ , which we will write  $\text{poly}(\lambda)$

<sup>3</sup>This means it is negligible in terms of  $\lambda$ , which we will write  $\text{negl}(\lambda)$ .

A PRG (Setup,  $G$ ) is *unpredictable* if, for any index  $i \in \{0, \dots, \ell(\lambda)\}$  and any efficient adversary  $\mathcal{A} : \{0, 1\}^n \rightarrow \{0, 1\}$ , we have that:

$$\left| \Pr_{k \leftarrow \mathcal{U}(\{0,1\}^{n(\lambda)})} [\mathcal{A}(G(k)|_i) = G(k)_{i+1}] - \frac{1}{2} \right| = \text{negl}(\lambda).$$

We can now prove that the two definitions are equivalent.

**Theorem 2.2.** The two definitions of a PRG are equivalent.

**Proof.** To simplify, we will remove the security parameter from the notations.

On one side, assume we have a predictor  $\mathcal{A} : \{0, 1\}^i \rightarrow \{0, 1\}$  that succeeds in guessing  $G(k)_{i+1}$  with non-negligible probability. We then construct a distinguisher  $\mathcal{B}$  against pseudo-randomness as  $\mathcal{B}$  receive a sample  $x$  from either  $\mathcal{D}_0 = \mathcal{U}(\{0, 1\}^\ell)$  or  $\mathcal{D}_1 = G(\mathcal{U}(\{0, 1\}^n))$ : algorithm  $\mathcal{B}$  runs  $\mathcal{A}$  on input  $x|_i$  and checks if  $\mathcal{A}(x|_i) \stackrel{?}{=} x_{i+1}$ . In that case,  $\mathcal{B}$  will return 1; otherwise it returns 0. What is the advantage of  $\mathcal{B}$ ?

$$\begin{aligned} \text{Adv}_{\mathcal{B}} &= \left| \Pr_{x \leftarrow \mathcal{D}_1} [\mathcal{B}(x) = 1] - \overbrace{\Pr_{x \leftarrow \mathcal{D}_0} [\mathcal{B}(x) = 1]}^{1/2} \right| \\ &= \left| \Pr_{x \leftarrow \mathcal{D}_1} [\mathcal{A}(x|_i) = x_{i+1}] - \frac{1}{2} \right|. \end{aligned}$$

This is the definition of the predictability advantage of  $\mathcal{A}$  (which is non-negligible by assumption).

Next, we will use a technique called an *Hybrid Argument* (due to Yao in '82). Assume we have a distinguisher  $\mathcal{A}$  such that

$$\text{Adv}_{\mathcal{A}} = \left| \Pr_{x \leftarrow \mathcal{D}_1} [\mathcal{A}(x) = 1] - \Pr_{x \leftarrow \mathcal{D}_0} [\mathcal{A}(x) = 1] \right|$$

is non-negligible, say  $\text{Adv}_{\mathcal{A}} \geq \varepsilon$ . We then define  $\ell+1$  distributions

$(\mathcal{D}_i)_{i=0,\dots,\ell}$  as

$$\mathcal{D}_i := \left\{ x \in \{0, 1\}^\ell \mid \begin{array}{l} x_{|i} = G(k)_{|i} \text{ for } k \leftarrow \mathcal{U}(\{0, 1\}^n) \\ x_{|i+1,\dots,\ell} \leftarrow \mathcal{U}(\{0, 1\}^{\ell-i}) \end{array} \right\}.$$

We then have, by all the terms cancelling (this is a telescoping sum), that:

$$\begin{aligned} \varepsilon \leq \text{Adv}_{\mathcal{A}}(\mathcal{D}_0, \mathcal{D}_n) &= \left| \sum_{i=0}^{\ell} \left( \Pr_{x \leftarrow \mathcal{D}_{i+1}} [\mathcal{A}(x) = 1] - \Pr_{x \leftarrow \mathcal{D}_i} [\mathcal{A}(x) = 1] \right) \right| \\ &\leq \sum_{i=0}^{\ell} \left| \Pr_{x \leftarrow \mathcal{D}_{i+1}} [\mathcal{A}(x) = 1] - \Pr_{x \leftarrow \mathcal{D}_i} [\mathcal{A}(x) = 1] \right| \\ &\leq \sum_{i=0}^{\ell} \text{Adv}_{\mathcal{A}}(\mathcal{D}_i, \mathcal{D}_{i+1}). \end{aligned}$$

By the pigeonhole principle, we have that there exists an  $i \in \{0, \dots, \ell\}$ , such that

$$\left| \Pr_{x \leftarrow \mathcal{D}_{i+1}} [\mathcal{A}(x) = 1] - \Pr_{x \leftarrow \mathcal{D}_i} [\mathcal{A}(x) = 1] \right| \geq \frac{\varepsilon}{\ell + 1}.$$

As  $\varepsilon$  is non-negligible and  $\ell + 1$  being polynomial in  $\lambda$ , we have that  $\varepsilon/(\ell + 1)$  is non-negligible. How to turn this into a predictor for  $i$ ? Let us define  $\mathcal{B}_i$  as a predictor which is given  $G(k)_{|i}$  and supposed to predict  $G(k)_{i+1}$ . Algorithm  $\mathcal{B}_i$  will compute  $x \in \{0, 1\}^\ell$  with  $x \leftarrow G(k)_{|i} || y$  where  $y \leftarrow \mathcal{U}(\{0, 1\}^{\ell-i})$ . Then  $\mathcal{B}_i$  runs algorithm  $\mathcal{A}$  on input  $x$ , and  $\mathcal{A}$  returns a bit  $b \in \{0, 1\}$  and  $\mathcal{B}_i$  outputs a prediction  $x_{i+1}$  for  $G(k)_{i+1}$  if  $b = 1$  and  $1 - x_{i+1}$

otherwise. What is the prediction advantage of  $\mathcal{B}_i$ ?

$$\begin{aligned}
 & \Pr[\mathcal{B}_i(G(k)|_i) = G(k)_{i+1}] \\
 &= \Pr \left[ \begin{array}{c} \mathcal{A}(x) = 0 \wedge x_{i+1} = 1 - G(k)_{i+1} \\ \vee \\ \mathcal{A}(x) = 1 \wedge x_{i+1} = G(k)_{i+1} \end{array} \right] \\
 &= \Pr_{x \leftarrow \mathcal{D}_i} [\mathcal{A}(x) = 0 \wedge x_{i+1} = 1 - G(k)_{i+1}] \\
 &\quad + \Pr_{x \leftarrow \mathcal{D}_i} [\mathcal{A}(x) = 1 \wedge x_{i+1} = G(k)_{i+1}] \\
 &= \frac{1}{2} \Pr_{x \leftarrow \mathcal{D}_{i+1}} [\mathcal{A}(x) = 0] + \frac{1}{2} \Pr_{x \leftarrow \mathcal{D}_i} [\mathcal{A}(x) = 1] \\
 &= \frac{1}{2} \left( \Pr_{x \leftarrow \mathcal{D}_{i+1}} [\mathcal{A}(x) = 1] + 1 - \Pr_{x \leftarrow \mathcal{D}_{i+1}} [\mathcal{A}(x) = 1] \right)
 \end{aligned}$$

where we write  $\bar{\mathcal{D}}_{i+1}$  is the “flipped” of  $\mathcal{D}_{i+1}$ . We have that:

$$\begin{aligned}
 & \Pr_{x \leftarrow \mathcal{D}_i} [\mathcal{A}(x) = 1] \\
 &= \Pr_{x \leftarrow \mathcal{D}_i} [\mathcal{A}(x) = 1 \wedge x_{i+1} = G(k)_{i+1}] \\
 &\quad + \Pr_{x \leftarrow \mathcal{D}_i} [\mathcal{A}(x) = 1 \wedge x_{i+1} = 1 - G(k)_{i+1}] \\
 &= \frac{1}{2} \left( \Pr_{x \leftarrow \mathcal{D}_i} [\mathcal{A}(x) = 1] + \Pr_{x \leftarrow \bar{\mathcal{D}}_{i+1}} [\mathcal{A}(x) = 1] \right),
 \end{aligned}$$

thus

$$\Pr_{x \leftarrow \bar{\mathcal{D}}_{i+1}} [\mathcal{A}(x) = 1] = 2 \Pr_{x \leftarrow \mathcal{D}_i} [\mathcal{A}(x) = 1] - \Pr_{x \leftarrow \mathcal{D}_{i+1}} [\mathcal{A}(x) = 1].$$

Hence,

$$\begin{aligned}
 & \Pr[\mathcal{B}_i(G(k)|_i) - G(k)_{i+1}] = \\
 & \frac{1}{2} \Pr_{x \leftarrow \mathcal{D}_{i+1}} [\mathcal{A}(x) = 1] + 1 - 2 \Pr_{x \leftarrow \mathcal{D}_i} [\mathcal{A}(x) = 1] + \Pr_{x \leftarrow \mathcal{D}_{i+1}} [\mathcal{A}(x) = 1].
 \end{aligned}$$



Finally, we can conclude that:

$$\text{Adv}_{\mathcal{A}}(\mathcal{D}_i, \mathcal{D}_{i+1}) = \left| \Pr[\mathcal{B}_i(G(k)|_i) = G(k)_{i+1}] - \frac{1}{2} \right| \geq \frac{\varepsilon}{n}.$$

□

**Example 2.2.** Let us go back to the One-Time PAD example. As said before, to get information-theoretic security, one needs the key's bit length to be no smaller than the message's length.

Now, how do we use the PRG to have a secure protocol? We encode using the PRG:

$$\text{Enc}_k(m \in \{0, 1\}^\ell) = m \oplus G(k) \in \{0, 1\}^\ell.$$

We can use a key of length 128 bits but encode a 1 Gb message.

If we have a PRG  $G : \{0, 1\}^n \rightarrow \{0, 1\}^{n+1}$  where  $n$  is the length of the key, then we can call  $G$  on itself a lot of times to get a string of any length  $\ell > n$ . (This is likely to be proven in the tutorials.)<sup>4</sup>

As seen before with the One-Time PAD, this kind of encryption can only be used once: you cannot re-use the key to encrypt multiple messages.

**Definition 2.7.** An encryption scheme (KeyGen, Enc, Dec) is called *secure against a single message chosen plain-text attack* if, for all polynomial-time adversary  $\mathcal{A}$ , and all  $m_0, m_1$  chosen by  $\mathcal{A}$ , we have that the two distributions are computationally indistinguishable:

$$\left( \text{Enc}(k, m_0) \right)_{k \leftarrow \text{KeyGen}(\cdot)} \simeq^c \left( \text{Enc}(k, m_1) \right)_{k \leftarrow \text{KeyGen}(\cdot)}.$$

<sup>4</sup>The teacher gave us an explanation on how we can double the length of a string, then it is easy to go from 128 to  $2^{30}$  bits. However, that construction is still using the  $n \rightarrow n + 1$  construction  $2^{23}$  times.

**Remark 2.3.** Another way of thinking about this kind of security is to imagine two players, the adversary  $\mathcal{A}$  and the challenger  $\mathcal{C}$ .

- ▷ The challenger generates a secret key  $k \in \{0, 1\}^n$  (which we assume to be uniform) and a uniform bit  $b \leftarrow \mathcal{U}(\{0, 1\})$ .
- ▷ The adversary give two messages  $m_1$  and  $m_2$  to  $\mathcal{C}$ .
- ▷ Then, the challenger encrypt  $m_b$  using the key, and gives it to  $\mathcal{A}$ .
- ▷ Finally,  $\mathcal{A}$  tries to “guess”  $b$  (*i.e.* which message was encrypted ( $m_0$  or  $m_1$ )).

Writing  $b^*$  for the guess of the adversary, we obtain a different formulation for the advantage of  $\mathcal{A}$  :

$$\text{Adv}(\mathcal{A}) = \left| 2 \times \Pr[b^* = b] - 1 \right|.$$

This definition of the advantage is equivalent (*c.f.* tutorials) to the one used before :

$$\text{Adv}(\mathcal{A}) = \left| \Pr[\mathcal{A} \text{ guesses } 1 \mid b = 0] - \Pr[\mathcal{A} \text{ guesses } 1 \mid b = 1] \right|.$$

**Proposition 2.1.** The PRG-based construction is secure against a single message chosen plain-text attack.

**Proof.** We want to show that, if there is an attacker against the PRG-based scheme, then there is a distinguisher for the PRG. We will use the “encryption security game” analogy in this proof. We define two games:

- ▷ Let  $\text{Hybrid}_0$  be the game where  $\mathcal{C}$  uses  $m_0$ .
- ▷ Let  $\text{Hybrid}_4$  be the game where  $\mathcal{C}$  uses  $m_1$ .

which we then complete with three other “intermediate” games:

- ▷ Let  $\text{Hybrid}_1$  be the game similar to  $\text{Hybrid}_0$  except that  $c =$

$m_0 \oplus G(k)$  is replaced by  $c = m_0 \oplus u$  where  $u \leftarrow \mathcal{U}(\{0, 1\}^\ell)$ .

- ▷ Let  $\text{Hybrid}_2$  be the game similar to  $\text{Hybrid}_1$  except that  $m_0$  is changed with  $m_1$  and thus  $c = m_1 \oplus u$ .
- ▷ Let  $\text{Hybrid}_3$  be the game similar to  $\text{Hybrid}_2$  except that  $c = m_1 \oplus u$  is replaced with  $c = m_1 \oplus G(k)$ .

We define

$$p_n := \Pr[\mathcal{A} \text{ guesses } 1 \text{ in the game } \text{Hybrid}_n].$$

The goal is to show that  $|p_0 - p_4|$  is negligible. To prove that we will prove that  $|p_0 - p_1|$ ,  $|p_1 - p_2|$ ,  $|p_2 - p_3|$  and  $|p_3 - p_4|$  are all negligible (we will then conclude by the triangle inequality). This strategy is called *Game Hopping*. By symmetry, we only need to consider  $|p_0 - p_1|$  and  $|p_1 - p_2|$ .

- ▷ Consider the games  $\text{Hybrid}_0$  and  $\text{Hybrid}_1$ . If  $\mathcal{A}$  can see the difference between the two cyphers, then it can be used to break the PRG. To prove this, we proceed by reduction. We introduce a new player,  $\mathcal{B}$ , who will pretend to be  $\mathcal{A}$  from the point of view of  $\mathcal{C}$  and *vice-versa*.

The players are then:

- $\mathcal{A}$  is the encryption adversary;
- $\mathcal{C}$  is the PRG challenger;
- $\mathcal{B}$  is both the encryption challenger and the PRG adversary.

We consider two cases: the “PRG” case and the “Uniform” case (depending on the choice for the key used to cypher the message. From the point of view of  $\mathcal{A}$ ,

- in the “PRG” case, it should be exactly as in  $\text{Hybrid}_0$ ;
- in the “Uniform” case, it should be exactly as in  $\text{Hybrid}_1$ .

The game will take place in the following way:

- $\mathcal{A}$  will give  $\mathcal{B}$  two messages  $m_0$  and  $m_1$ ;
- $\mathcal{C}$  will give  $\mathcal{B}$  a key  $y$  with the required length (either generated uniformly in the “Uniform” case, or with the PRG in the “PRG” case).
- $\mathcal{B}$  encrypts the message  $m_0$  using the key  $y$ , and gives it to  $\mathcal{A}$ .
- $\mathcal{A}$  sends its guess  $b^*$  to  $\mathcal{B}$ , who directly sends it to  $\mathcal{C}$ .

Because  $\mathcal{A}$ 's view is consistent, it behaves as unexpected in  $\text{Hybrid}_0$  or  $\text{Hybrid}_1$ . This means that:

- in the “PRG” case,  $\mathcal{B}$  outputs 1 iff  $\mathcal{A}$  outputs 1, which happens with probability  $p_0$ .
- in the “PRG” case,  $\mathcal{B}$  outputs 1 iff  $\mathcal{A}$  outputs 1, which happens with probability  $p_1$ .

If  $|p_0 - p_1| = |\Pr[\mathcal{B} \leftarrow 1 \mid \text{PRG case}] - \Pr[\mathcal{B} \leftarrow 1 \mid \text{Uniform case}]|$  is non-negligible, then  $\mathcal{B}$  breaks the PRG. And, if  $\mathcal{A}$  is efficient, so is  $\mathcal{B}$ . Thus, if the PRG is secure, then  $|p_0 - p_1|$  is negligible.

- ▷ For the games  $\text{Hybrid}_1$  and  $\text{Hybrid}_2$ , we will prove that  $p_1 = p_2$ . As  $u$  is chosen uniformly, then  $\mathcal{A}$  receives a uniform cypher  $c$  in both games. Then, as  $\mathcal{A}$  has the same view, it has the same behavior. The rest of the proof is exactly the one for the perfect security of the One-Time PAD.

□

### 2.1.1 How to get PRGs? Cryptographic assumptions.

One example of a PRG is called RC4 (defined by Rivest in '87). It has some weaknesses. This PRG was used in WEP, an very old WiFi protocol (still used by 2 % of WiFi routers), and it has been totally broken (the WEP protocol added weaknesses on top of RC4's). It is

also used by Bittorent. The state of the art is Salsa20 (software) or Trivium (hardware).

**Definition 2.8.** A function  $f : \{0, 1\}^k \rightarrow \{0, 1\}^\ell$  is called *one way* (with no relation between  $\ell$  and  $k$ ) if it is computable in polyomial-time and for any polynomial-time adversary  $\mathcal{A}$ , its advantage

$$\text{Adv}(\mathcal{A}) = \Pr_{x \leftarrow \mathcal{U}(\{0,1\}^k)} [\mathcal{A}(f(x)) = x' \text{ where } f(x) = f(x')]$$

is negligible.

We have that:

- ▷ if there exists a PRG, then there is a one-way function
- ▷ if there is a one-way function, then there is a PRG (Goldreich-Levin hard-cord bits).

There also exists explicit universal functions: if a one-way function exists, then the universal function is one way.

This problem is connected to the **P** vs. **NP** problem (existence of one-way function implies  $\mathbf{P} \neq \mathbf{NP}$ ).

**Definition 2.9 (Discrete Logarithm Problem, DLP).** The DLP is defined relative to a prime-order cyclic group  $G$  with a generator  $g \in G$ . This means that

$$G = \{g^k \mid k = 0, \dots, p-1\},$$

where  $p = |G|$  is a prime number. The group  $G$  and the element  $g$  are publicly known. The goal is, given  $h \in G$ , find a  $x$  such that  $g^x = h$ .

**Example 2.3.** In  $(\mathbb{Z}/p\mathbb{Z}, +)$ , the DLP problem is quite easy.

In  $G_p := ((\mathbb{Z}/p\mathbb{Z})^\times, \times)$  is cyclic of order  $p - 1$ , but  $p - 1$  is not necessarily a prime! We take a prime  $p$  such that  $p = 2q + 1$  where  $q$  is prime (such primes are called *safe primes*). We have that

$$G_p = \{g, g^2, g^3, \dots, g^{p-1}\}$$

and

$$G_q = \{(g^2)^0, (g^2)^2, \dots, (g^2)^k, \dots, \overbrace{(g^2)^{(p-1)/2}}^{p^q}\}.$$

The group  $G_q$  is cyclic with prime order  $q$ . To find a generator for  $G_q$ , we simply sample uniformly an element  $g_0$  of  $(\mathbb{Z}/p\mathbb{Z})^\times$ , then take  $h := g_0^2$ . This is in fact a generator as long as  $g_0 \notin \{-1, 1\}$ .

In the 2000s, cryptographers started using the group of elements of an elliptic curve over a finite field. For prime order subgroups of  $((\mathbb{Z}/p\mathbb{Z})^\times, \times)$ , the best known algorithms cost

$$\exp(\tilde{O}(\sqrt[3]{\ln |G|})) \ll \exp(O(\ln |G|)).$$

The other cost is for generic “black box” groups (hardness of DLP). This blackbox algorithm is the best known algorithm for elliptic curves with  $\log_2 |G| \approx 256$ .

Thus,  $p$  and  $q$  have to be quite large to be hard-to-solve (around 4 096 bits) on the case of prime order subgroups of  $(\mathbb{Z}/p\mathbb{Z})^\times$ . Given  $h = g^x$ , there is a baby-step-giant-step algorithm to find  $x$ .

- ▷ We start by computing  $\{g^0, g, g^1, \dots, g^{\sqrt{q}}\}$  (baby steps);
- ▷ Then, we compute  $\{hg^{-\sqrt{q}}, hg^{-2\sqrt{q}}, hg^{-3\sqrt{q}}, \dots, hg^{-\sqrt{q}\sqrt{q}}\}$  (giant steps).

The cost for each step is around  $\sqrt{q}$ . As  $h = g^x = g^{x_0 + \sqrt{q}x_1}$ , then we have that  $g^{x_0} = h \cdot g^{\sqrt{q}x_1}$ . Each of these elements is in one set.

Then, if we find two elements  $g^{x_0}$  in the baby steps and  $h \cdot g^{\sqrt{q}x_1}$  in the giant steps that are equal, then we get  $h = g^{x_0} \cdot g^{\sqrt{q}x_1} = g^{x_0 + \sqrt{q}x_1}$ , thus we solve the DLP solution.

That's a  $O(\sqrt{|G|})$  time algorithm for finding the DLP.<sup>5</sup>

