Cryptography and Security

Based on the lectures of
Alain Passelègue and Damien Stehlé
Notes written by Hugo Salou

ENS DE LYON

September 12, 2025

Contents

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 understanding 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 Some famous examples of cryptographic hash functions are SHA-256, SHA-2, SHA-3 or MD-5 (this last one is outdated).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. **Authentification.** Alice asks Bob some kind of identification, and Bob will send Alice a certificate of authentification.
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. Publick-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 ressources?

  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, authentification.

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 irrealistic (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-theoric security.

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

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

- ▷ KeyGen provides a sample k of $\mathscr{K}$;
- ▷ $\mathrm{Enc} : \mathscr{K} \times \mathscr{P} \to \mathscr{C}$;
- ▷ $\mathrm{Dec} : \mathscr{K} \times \mathscr{C} \to \mathscr{P}$.

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

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

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

Remark 2.1. The algorithm Enc could (and should[1]) 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 $\bar{c}$,

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

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

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

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

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

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

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

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

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

□

[1]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 $\mathscr{K} = \mathscr{C} = \mathscr{P} = \{0,1\}^\ell$. Here are the algorithms used:

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

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

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

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

□

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.

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

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

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

We say that $\mathscr{D}_0$ and $\mathscr{D}_1$ are *computationally indistinguishable* if for any efficient distinguisher $\mathscr{A}$ its advantage $\mathrm{Adv}_{\mathscr{A}}$ is small.

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 $(\mathscr{D}_{0,\lambda})_{\lambda\in\mathbb{N}}$ and $(\mathscr{D}_{1,\lambda})_{\lambda\in\mathbb{N}}$ 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 $\mathscr{A} : \{0,1\}^{n(\lambda)} \to \{0,1\}$ is called a *distinguisher* between the distributions $\mathscr{D}_{0,\lambda}$ and $\mathscr{D}_{1,\lambda}$. We define its *distinguishing advantage* as:

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

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

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 expend it to a string $G(k) \in \{0,1\}^\ell$ for some $\ell > k$ 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 (Setup, G) such that:

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

A PRG (Setup, G) is *unpredictable* if, for any index $i \in \{0, \dots, \ell(\lambda)\}$

[2]This means it is polynomial in λ, which we will write $\mathrm{poly}(\lambda)$

[3]This means it is negligible in terms of λ, which we will write $\mathrm{negl}(\lambda)$.

and any efficient adversary $\mathscr{A} : \{0,1\}^n \to \{0,1\}$, we have that:

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

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

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

This is the definition of the predictability advantage of $\mathscr{A}$ (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 $\mathscr{A}$ such that

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

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

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

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

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

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

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

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

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

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

.

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

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

,

thus

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

Hence,

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

Finally, we can conclude that:

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

$\square$