Cryptography and Security

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

October 14, 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*.

References.

- Introduction to Modern Cryptography by Katz and Lidel (absolute best).
- MOOC by Dan Moneh available on Youtube.
- Mathematical Cryptography by Steven Galbraith (more on the algebra side).

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 Some famous examples of cryptographic hash functions are SHA-256, SHA-2, SHA-3 or MD-5 (this last one is outdated).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 Private key encryption.

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.

[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.

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

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

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

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

□

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

- ▷ KeyGen samples from $\mathcal{U}(\{0,1\}^\ell)$.
- ▷ $\mathrm{Enc}(k, m)$ 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 in the next section.

2.1 Pseudo-random generators.

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:

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

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. We will write, in this case, $\mathscr{D}_1 \simeq^{\mathrm{c}} \mathscr{D}_2$.

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 $(\mathscr{D}_{0,\lambda})_{\lambda \in \mathbb{N}}$ 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:

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

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

[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 $\mathrm{negl}(\lambda)$.

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 $i \in \{0, \dots, \ell(\lambda)\}$ and any efficient adversary $\mathcal{A} : \{0,1\}^n \to \{0,1\}$, we have that:

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

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 $\mathcal{A} : \{0,1\}^i \to \{0,1\}$ 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 $\mathcal{B}$?

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

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

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

is non-negligible, say $\mathrm{Adv}_{\mathcal{A}} \geq \varepsilon$. We then define $\ell+1$ distributions $(\mathcal{D}_i)_{i=0,\dots,\ell}$ as

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

We then have, by all the terms cancelling (this is a telescoping

sum), that:

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

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

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

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,

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

Finally, we can conclude that:

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

□

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:

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

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

If we have a PRG $G : \{0,1\}^n \to \{0,1\}^{n+1}$ 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.)[4]

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

$$\big(\mathrm{Enc}(k, m_0)\big)_{k \leftarrow \mathrm{KeyGen}()} \simeq^{\mathrm{c}} \big(\mathrm{Enc}(k, m_1)\big)_{k \leftarrow \mathrm{KeyGen}()}.$$

[4] 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 $n \to n+1$ construction 2^{23} times.

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

- ▷ 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 $\mathscr{C}$.
- ▷ Then, the challenger encrypt m_b using the key, and gives it to $\mathscr{A}$.
- ▷ Finally, $\mathscr{A}$ tries to "guess" b (*i.e.* which message was encrypted (m_0 or m_1).

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

$$\mathrm{Adv}(\mathscr{A}) = \left|2 \times \Pr[b^\star = b] - 1\right|.$$

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

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

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 fo the PRG. We will use the "encryption security game" analogy in this proof. We define two games:

- ▷ Let Hybrid_0 be the game where $\mathscr{C}$ uses m_0.
- ▷ Let Hybrid_4 be the game where $\mathscr{C}$ uses m_1.

which we then complete with three other "intermediate" games:

- ▷ Let Hybrid_1 be the game similar to 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 Hybrid_2 be the game similar to Hybrid_1 except that m_0 is changed with m_1 and thus $c = m_1 \oplus u$.
- ▷ Let Hybrid_3 be the game similar to Hybrid_2 except that $c = m_1 \oplus u$ is replaced with $c = m_1 \oplus G(k)$.

We define

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

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 Hybrid_0 and Hybrid_1. If $\mathscr{A}$ 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, $\mathscr{B}$, who will pretend to be $\mathscr{A}$ from the point of view of $\mathscr{C}$ and *vice-versa*.

  The players are then:

  - $\mathscr{A}$ is the encryption adversary;
  - $\mathscr{C}$ is the PRG challenger;
  - $\mathscr{B}$ 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 $\mathscr{A}$,

  - in the "PRG" case, it should be exactly as in Hybrid_0;
  - in the "Uniform" case, it should be exactly as in Hybrid_1.

  The game will take place in the following way:

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

Because $\mathscr{A}$'s view is consistent, it behaves as unexpected in Hybrid_0 or Hybrid_1. This means that:

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

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

▷ For the games Hybrid_1 and Hybrid_2, we will prove that $p_1 = p_2$. As u is chosen uniformly, then $\mathscr{A}$ receives a uniform cypher c in both games. Then, as $\mathscr{A}$ 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.

□

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

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

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

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

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 $(\mathbb{Z}/p\mathbb{Z})^\star$, 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 $((\mathbb{Z}/p\mathbb{Z})^\star, \times)$, the best known algorithms cost

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

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 $(\mathbb{Z}/p\mathbb{Z})^\star$. 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 $\mathrm{O}\big(\sqrt{|G|}\big)$ time algorithm for finding the DLP.[5]

[5] The intersection of two lists of length $\sqrt{|G|}$ can be easily done in $\mathrm{O}\big(\sqrt{|G|}\big)$ by using a hashmap, for example, to check if there are collisions.

Definition 2.10 (Computational Diffie-Hellman Problem, CDH)**.** The CDH problem is defined as taking as input g a generator of a group G, and two elements g^a and g^b. The goal is to find g^{ab}. This is not as easy as computing $g^a \cdot g^b$ as it'd give us g^{a+b} which is generally different from g^{ab}.

If DLP can be easily solved, then so is CDH: we say CDH *reduces* to DLP. The other direction is still an open problem, with partial results by Mauer and Wolf in '99.

Definition 2.11 (Decisional Diffie Hellman, DDH)**.** The DDH is the problem defined by the following. It takes as input three elements h_1, h_2 and h_3 of G either uniformly chosen (*i.e.* g^a, g^b, g^c for $a, b, c \leftarrow \mathcal{U}(\mathbb{Z}/p\mathbb{Z})$) or of the form (g^a, g^b, g^{ab}) for some $a, b \in \mathcal{U}(\mathbb{Z}/p\mathbb{Z})$. The goal is to distinguish between the two cases.

The advantage of an distinguisher $\mathcal{A}$ is given by

$$\mathrm{Adv}^{\mathrm{DDH}}(\mathcal{A}) := \Big| \Pr_{\substack{a,b,c\leftarrow\mathcal{U}(\mathbb{Z}/p\mathbb{Z}) \\ \mathrm{coins}(\mathcal{A})}}[\mathcal{A}(g^a, g^b, g^c) \text{ outputs } 1] - \Pr_{\substack{a,b\leftarrow\mathcal{U}(\mathbb{Z}/p\mathbb{Z}) \\ \mathrm{coins}(\mathcal{A})}}[\mathcal{A}(g^a, g^b, g^{ab}) \text{ outputs } 1]\Big|,$$

where the "coins($\mathcal{A}$)" means the internal randomness of $\mathcal{A}$.

If $\mathrm{Adv}^{\mathrm{DDH}}(\mathcal{A})$ is non-negligible then we say $\mathcal{A}$ *solves* the DDH problem.

If one can solve the CDH problem, then we can also solve the DDH. But we know groups for which DDH is easy and CDH is presumed/believed to be hard (elliptic curves with efficient pairings, which are very frequently used in blockchain or BLS signatures). The DDH direclty gives a PRG (we will use G for the PRG and G for the

group):

$$\begin{aligned} \mathsf{G} : (\mathbb{Z}/p\mathbb{Z}) \times (\mathbb{Z}/p\mathbb{Z}) &\longrightarrow G \times G \times G \\ (a, b) &\longmapsto (g^a, g^b, g^{ab}). \end{aligned}$$

It is expanding if the bit-size of the group elements is around $\log_2 p$. "It is indistinguishable from uniform by the DDH assumption."

The DDH problem only says that $(g^a, g^b, g^{ab})_{a,b\in\mathbb{Z}/p\mathbb{Z}} \simeq^{\mathrm{c}} \mathcal{U}(G^3)$ but not uniform over the bit-strings of the same length. It is not obvious how to do one-time encryption by XORing. To tackle this we can use a hash function H and output $H(g^a, g^b, g^{ab})$ (and we'd need to modify the DDH problem to put H in it).

The discrete logarithm problem is quantumly easy with Shor's algorithm (Shor '94). Thus DLP-based cryptography is not quantum-safe. This is the main assumption used for quantum-safe (or post-quantum) cryptography. For example, the LWE (learning with errors, 2005, Regev) is defined as in the following.

Definition 2.12 (Search LWE)**.** Fix n, m, q, B four integers. The inputs are a matrix A of size $n \times m$ of elements chosen uniformly in $\mathbb{Z}/q\mathbb{Z}$, a vector b of size m such that $b = As + e \mod q$ where s is chosen uniformly in $(\mathbb{Z}/q\mathbb{Z})^n$, e is chosen uniformly in $[\![-B, B]\!]^m$ with the assumption that $B \ll q$. The goal is to find such a vector s.

We call:

- s, the secret;
- e, the error;
- n, the LWE dimension;
- m, the number of samples;
- q, the modulus.

Remark 2.4.
- ▷ The integer q does not have to be prime, but to prove thins it's much easier when it is, as $\mathbb{Z}/q\mathbb{Z}$ is a field (we will restrict ourselves to this case).
- ▷ If $B = 0$ then LWE is easy as we can use probabilistic arguments to solve the systems.
- ▷ If B is large, LWE is trivially hard as (A, b) is uniform so the vector s is not even well-defined.
- ▷ Typically, m is a bit larger than n, q is a small polynomial in n and B is like $\sqrt{n}$. In this case, the best-known algorithms (classical and quantum) cost $2^{\Omega(n)}$.
- ▷ Very often, the error vector e is sampled from other distributions than $\mathcal{U}(\llbracket -B, B \rrbracket)$. The most frequent choice is the use of an integer Gaussian as it simplify proofs.

Definition 2.13 (Decision LWE). Extending from the definition of the search LWE, we define the Decision LWE problem as the following: Given (A, b) which is either uniform in $(\mathbb{Z}/q\mathbb{Z})^{m\times n} \times (\mathbb{Z}/q\mathbb{Z})^m$ or of the form $(A, As + b)$ as before, tell in which case you are with non-negligible advantage.

We define the advantage of some distinguisher $\mathcal{A}$ as:

$$\mathrm{Adv}^{\mathrm{LWE}}(\mathcal{A}) := \Big| \Pr_{\substack{(A,b) \text{ uniform} \\ \text{coins}(\mathcal{A})}} \mod q [\mathcal{A}(A, b) \text{ outputs } 1] - \Pr_{\substack{(A,s) \text{ uniform} \\ e\leftarrow\mathcal{U}(\llbracket -B,B \rrbracket) \\ \text{coins}(\mathcal{A})}} \mod q [\mathcal{A}(A, As + e) \text{ outputs } 1] \Big|.$$

A surprising fact is that search LWE and decision LWE are equivalent (in the sense that there are reductions to one another in polynomial time).

Also, the decision LWE gives us a PRG:

$$\begin{aligned}\mathsf{G} : (\mathbb{Z}/q\mathbb{Z})^{m\times n} \times (\mathbb{Z}/q\mathbb{Z})^n \times [\![-B, B]\!]^m &\longrightarrow (\mathbb{Z}/q\mathbb{Z})^{m\times n} \times (\mathbb{Z}/q\mathbb{Z})^m \\ (A, s, e) &\longmapsto (A, As + e).\end{aligned}$$

This really is a PRG as:

- the output is computationally indistinguishable from uniform (over $(\mathbb{Z}/q\mathbb{Z})^{m\times n} \times (\mathbb{Z}/q\mathbb{Z})^m$;
- it is expanding (if $m \gg n$ and $q \gg B$) as we have
  - $mn + \log q + m \log q + m(1 + \log B)$ bits for the input,
  - $mn \log q + m \log n$ bits for the output.

In a way, search LWE is a kind of DLP problem in additive notation... with some noise.

2.3 Pseudo Random Functions.

A PRG $G : \{0,1\}^s \to \{0,1\}^n$ with $n > s$ is such that $G(s)$ looks uniform when s is uniform. In terms of difficulty, a PRG-based encryption is only one-time secure.

Definition 2.14. A *pseudo random function* or *PRG* is a deterministic and efficiently computable function $F : \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^m$ such that $F(k, -)$ is computationally indistinguishable from a truly uniform function from $\{0,1\}^n$ to $\{0,1\}^m$ when k is uniform in $\{0,1\}^s$.

Remark 2.5. We can see this as a game between a challenger $\mathscr{C}$ and an adversary $\mathscr{A}$. The goal is to distinguish between two kind of games.

The "Real" experiment is:

- The challenger picks a key k uniformly over $\{0,1\}^s$.
- The attacker picks an input x_1 and sends it to $\mathscr{C}$.

- ▷ The challenger computes $y_1 := F(k, x_1)$ and sends it to $\mathcal{A}$.
- ▷ The attacker picks an input x_2 and sends it to $\mathcal{C}$.
- ▷ The challenger computes $y_2 := F(k, x_2)$ and sends it to $\mathcal{A}$.
- ▷ *etc*
- ▷ The attacker picks an input x_Q and sends it to $\mathcal{C}$.
- ▷ The challenger computes $y_Q := F(k, x_Q)$ and sends it to $\mathcal{A}$.
- ▷ Finally the attacker outputs a bit $b \in \{0, 1\}$.

The "Ideal" experiment is:

- ▷ The challenger picks a function f uniformly over $\{0,1\}^n \to \{0,1\}^m$.
- ▷ The attacker picks an input x_1 and sends it to $\mathcal{C}$.
- ▷ The challenger computes $y_1 := f(x_1)$ and sends it to $\mathcal{A}$.
- ▷ The attacker picks an input x_2 and sends it to $\mathcal{C}$.
- ▷ The challenger computes $y_2 := f(x_2)$ and sends it to $\mathcal{A}$.
- ▷ *etc*
- ▷ The attacker picks an input x_Q and sends it to $\mathcal{C}$.
- ▷ The challenger computes $y_Q := f(x_Q)$ and sends it to $\mathcal{A}$.
- ▷ Finally the attacker outputs a bit $b \in \{0, 1\}$.

The advantage is defined by:

$$\mathrm{Adv}(\mathcal{A}) := \Big| \Pr_{\mathrm{coins}(\mathcal{C}),\mathrm{coins}(\mathcal{A})}[\mathcal{A} \xrightarrow{\text{"Real" exp.}} 1] - \Pr_{\mathrm{coins}(\mathcal{C}),\mathrm{coins}(\mathcal{A})}[\mathcal{A} \xrightarrow{\text{"Ideal" exp.}} 1] \Big|.$$

We say that the PRF is *secure* if there is no poly-time adversary $\mathcal{A}$ such that $\mathrm{Adv}(\mathcal{A})$ is non-negligible.

Remark 2.6. This definitions as a game can be tricky to deal with, as the challenger $\mathscr{C}$ has to pick uniformly a function $f : \{0,1\}^n \to \{0,1\}^m$.

We can define a uniform function to be a function (if you query twice the same input, you get the same output) such that for each input x the output $f(x)$ is uniform.

To deal with such a function, we can use a table to store outputs for already-given inputs. If the input is in the table, we give the corresponding output. Otherwise, we sample uniformly in the output space.

Now it is clear that $\mathscr{C}$ is efficient as the number of queries is bounded by the runtime of $\mathscr{A}$.

The number of functions for the "Real" experiment is 2^s, one for each key. The number of functions for the "Ideal" experiment is $(2^m)^{(2^n)}$. There is a large difference in the number of such functions.

2.4 Equivalence between PRGs and PRFs.

We will show (in the following classes) that the existence of a PRG is equivalent to the existence of a PRF.

2.4.1 From PRF to PRG.

This is the "easy part." Let $F : \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^m$ be a PRF.

How to build a PRG $G : \{0,1\}^s \to \{0,1\}^{\ell m}$ for an arbitrary (but small) integer $\ell \geq 1$?

We can define

$$\begin{aligned} G : \{0,1\}^s &\longrightarrow \{0,1\}^{\ell m} \\ k &\longmapsto F(k,\bar{0})||F(k,\bar{1})||\cdots||F(k,\overline{\ell-1}), \end{aligned}$$

where $\bar{n}$ is the number n written in binary in $\{0,1\}^s$.

Lemma 2.2. The generator G is a PRG.

Proof. We proceed by reduction. Assume that we have a distinguisher $\mathscr{A}$ against G. I want to build a PRF distinguisher $\mathscr{B}$ against the PRF F, using $\mathscr{A}$.

Let $\mathscr{C}$ be a PRF challenger. The $\mathscr{B}$ will be a PRF attacker while also being a PRG challenger. Finally, $\mathscr{A}$ will be the PRG attacker.

- ▷ Initially, $\mathscr{C}$ chooses whether it'll play the "Real" or "Ideal".
- ▷ $\mathscr{B}$ will send $\bar{0}$ to $\mathscr{C}$.
- ▷ $\mathscr{C}$ will return some value y_0.
- ▷ $\mathscr{B}$ will send $\bar{1}$ to $\mathscr{C}$.
- ▷ $\mathscr{C}$ will return some value y_1.
- ▷ *etc*
- ▷ $\mathscr{B}$ will send $\overline{\ell - 1}$ to $\mathscr{C}$.
- ▷ $\mathscr{C}$ will return some value $y_{\ell-1}$.
- ▷ Then, $\mathscr{B}$ will send $\mathscr{A}$ the string

$$y_0||y_1||\cdots||y_{\ell-1}.$$

- ▷ Finally, $\mathscr{A}$ will send $\mathscr{B}$ a bit, that $\mathscr{B}$ will send $\mathscr{C}$.

We can consider two cases.

Case 1. Challenger $\mathscr{C}$ plays the "real" experiment, *i.e.* it samples k uniformly and sets $y_i = F(k, \bar{i})$. Thus, $\mathscr{A}$ gets

$$F(k, \bar{1})||\ldots||F(k, \overline{\ell - 1}) = G(k).$$

Case 2. Challenger $\mathscr{C}$ plays the "ideal" experiment, *i.e.* the replies y_i to queries on different is are chosen uniformly and independent. Thus, $\mathscr{A}$ gets $y_0||\cdots||y_{\ell-1}$ which is uniform.

We have that

$$\begin{aligned}\mathrm{Adv}(\mathscr{B}) &= \Big|\Pr[\mathscr{B} \xrightarrow{\text{"Real" exp.}} 1] - \Pr[\mathscr{B} \xrightarrow{\text{"Ideal" exp.}} 1]\Big| \\ &= \Big|\Pr_{k\leftarrow\mathscr{U}(\{0,1\}^s)}[\mathscr{A}(G(k)) \to 1] - \Pr_{y\leftarrow\mathscr{U}(\{0,1\}^{\ell m})}[\mathscr{A}(y) \to 1]\Big|,\end{aligned}$$

which is non-negligible as $\mathscr{A}$ is a PRG attacker for G. Thus $\mathscr{B}$ has a non-negligible advantage in breaking the PRF F. Finally, if $\mathscr{A}$ is poly-time then so is $\mathscr{B}$. □

2.4.2 From PRG to PRF.

This construction of PRF from a PRG is the first non-trivial construction (it is called the GGM construction, from Goldreich, Geldwesser and Micali in the mid '80s).

Consider a PRG $G : \{0,1\}^n \to \{0,1\}^{2m}$. We will define a PRF

$$F : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n.$$

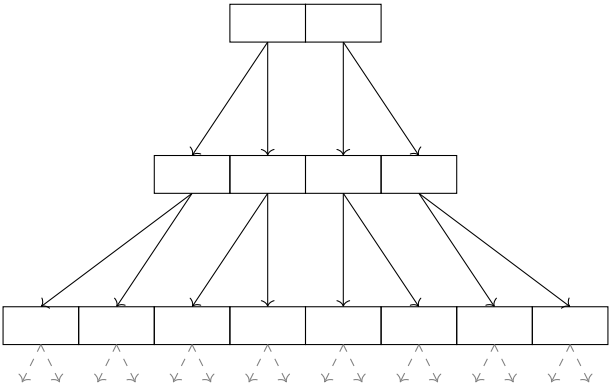
Let us do, as a warm-up, some small values of m.

Case $m = 1$**.** We define

$$F(k,0) := G(k)_{1\text{—}n} \qquad \text{and} \qquad F(k,1) := G(k)_{n+1\text{—}2n}.$$

In this case, the proof of the secureness of the PRF is tautological: the first half is uniform and independent from the second half, which is also uniform.

Case $m = 2$**.** For $m = 2$, I need a new idea as $G(k)$ has only $2n$ bits, but I need $4n$ pseudo-random bits (for inputs 00, 01, 10, 11). The idea is to apply G to itself. If the first bit of the input is 0, then apply G to the first half of $G(k)$ and use case $m = 1$ for that. Otherwise (the first bit of the input is 1), apply G to the second half and proceed like the case $m = 1$.



For larger values of m, we consider a tree of height m, where leaves corresponds to outputs of F in the sense that, if the path is labelled as $b_1, \dots, b_m$, the leaf corresponds to $F(k, b_1 \dots b_m)$.

As an algorithm, we get the following.

Algorithm 1 Construction of a PRF from a PRG

Require: $k, x \in \{0,1\}^m$
  $\mathsf{tmp} \leftarrow G(k) \in \{0,1\}^{2n}$
  for $i = 1 \dots m$ **do**
    $\mathsf{tmp} \leftarrow \begin{cases} G(\mathsf{tmp}_{1\text{—}n}) & \text{if } x_i = 0 \\ G(\mathsf{tmp}_{n+1\text{—}2n}) & \text{if } x_i = 1 \end{cases}$
  return tmp

It's indeed an efficient function $\{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$. But why is it a PRF?

Warning! We can use the assumption that G is a PRG only if its input is uniform.

Lemma 2.3. The constructed function is a PRF.

Proof. Define the following games.

- ▷ Hybrid_0 is the "Real" experience PRF game.

▷ Hybrid_j is defined with the following changes:

– at the outset of the game, the challenger initializes an empty table T,

– given a query $x_1 \dots x_m$, the challenger executes the following algorithm:

- it looks at whether $(x_1 \dots x_j, \mathsf{tmp}_{x_1 \dots x_j}) \in T$, if so, define $\mathsf{tmp} := \mathsf{tmp}_{x_1 \dots x_j}$, otherwise it samples $\mathsf{tmp} \leftarrow \mathcal{U}(\{0,1\}^{2n})$ and appends $(x_1 \dots x_j, \mathsf{tmp})$ to T. (this is ta lazy sampling of a uniform function $\{0,1\}^j \to \{0,1\}^{2n}$).
- For $i = j+1 \dots m$, it updates $\mathsf{tmp} \leftarrow G(\mathsf{tmp}_{1 \dots n})$ if x_{j+1} is 0, and the result of G on the other half if $x_{j+1} = 1$.

This way, the game Hybrid_j is the initial game except that we "cut off" the first j levels of the tree of calls to G, and replace them with uniform random values.

For $j = m$, the game Hybrid_m is exactly the "Ideal" experience of the PRF game. We can bound the advantage using the triangular inequality:

$$\begin{aligned}\text{Adv}_{\text{PRF}}(\mathcal{A}) &= \left| \Pr[\mathcal{A} \xrightarrow{\text{Real}} 1] - \Pr[\mathcal{A} \xrightarrow{\text{Ideal}} 1] \right| \\ &\leq \sum_{j=1}^{m} \left| \Pr[\mathcal{A} \xrightarrow{\text{Hybrid}_j} 1] - \Pr[\mathcal{A} \xrightarrow{\text{Hybrid}_{j+1}} 1] \right|.\end{aligned}$$

We have m hybrids.

Continuing with hybrid games, we define

▷ $\text{Hybrid}_{j,\ell}$ to be the same game as Hybrid_j except that, for the first ℓ prefixes $x_i \dots x_{j+1}$ that are queried, use Hybrid_{j+1}, for the next ones, use Hybrid_j. Here, $\ell \in [\![0, q_{j+1}]\!]$ where q_{j+1} is the number of $(n-j-1)$-prefixes that occur in the queries from the adversary.

The number of $\text{Hybrid}_{j,\ell}$'s games is no larger than the number of queries made by $\mathcal{A}$, and q is smaller than the run-time of $\mathcal{A}$, it is polynomial. We also have that

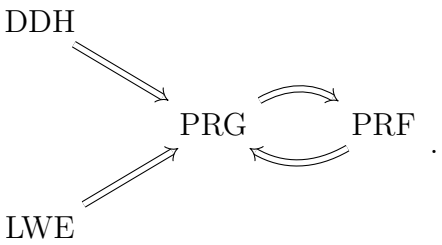
$$\text{Adv}_{\text{PRF}}(\mathcal{A}) \leq \sum_{j=1}^{m} \sum_{\ell=1}^{q_{j+1}} \left| \Pr[\mathcal{A} \xrightarrow{\text{Hybrid}_{j,\ell}} 1] - \Pr[\mathcal{A} \xrightarrow{\text{Hybrid}_{j,\ell-1}} 1] \right|.$$

The difference between $\text{Hybrid}_{j,\ell}$ and $\text{Hybrid}_{j,\ell+1}$ is exactly one call to G on a uniform input. Assuming G is a secure PRG, this advantage is $\text{negl}(n) = n^{-\omega(1)}$. Thus:

$$\text{Adv}_{\text{PRF}}(\mathcal{A}) \leq n^{-\omega(1)} \times m \times \max(q_{j+1}) \leq n^{-\omega(1)} \times \underbrace{m \times q}_{\text{poly}(n)} \leq n^{-\omega(1)}.$$

(The number of hybrid games must be less than a polynomial in n.) □

As a consequence, we have the following implications:

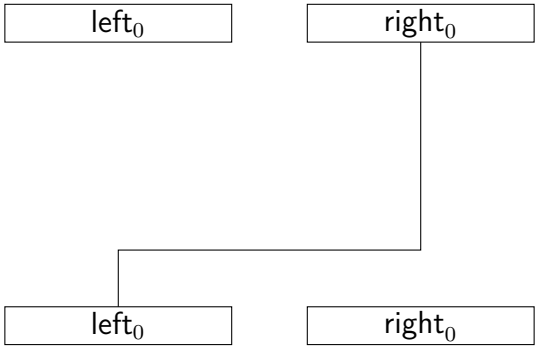


These PRFs are not used in practice: for basic PRFs, we use ad-hoc constructions, for PRFs with more advanced functionalities, we direction constructions from LWE and DDH can be interesting.

2.5 Key homomorphic PRF.

Can we construct a PRF such that with the following property?

$$F(k_1 \oplus k_2, x) = F(k_1, x) \oplus F(k_2, x).$$



We can make an ad-hoc construction, the old default PRF is DES (deterministic encryption standard). DES is based on a Feistel network with **16 rounds** (with distinct keys for every round, all derived from the "master" key).

A *Feistel network* gives a permutation even if f_k is not a permutation.

In the one presented above, we have $\mathsf{right}_0 = \mathsf{left}_1$, and $\mathsf{left}_0 = f_k(\mathsf{left}_1) \oplus \mathsf{right}_1$.

A PRP is a pseudo random permutation that come in two kinds: weak and strong.

- A *weak PRP* is defined like a PRF except that $F(k, -)$ is a permutation for any key k.
- A *strong PRP* is a weak PRP except that $F(k, -)^{-1}$ should be efficient and the adversary is given query access to both $F(k, -)$ and $F(k, -)^{-1}$.

We can consider five exercises:

- a PRF and a two-round Feistel network doesn't define a weak PRP ;
- a PRF and a three-round Feistel network does define a weak PRP ;
- a PRF and a four-round Feistel network doesn't define a strong PRP ;

- a PRF and a five-round Feistel network does define a strong PRP.

In practice today, people use AES which is not based on a Feistel network (the key size in AES is 128, 192 or 256).

How do we encrypt with a PRF $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$?

Given k and $m = m_1 m_2 m_3$, we could encrypt it as

$$F(k, m_1)||F(k, m_2)||F(k, m_3).$$

DON'T DO THIS!! The adversary can see of two m_i's are the same. To avoid this kind of attacks, it is necessary that the encryption is randomized.

An encryption scheme (Enc, Dec) is *IND-CPA* (*undistinguishable under chosen plain-text attacks*) secure for multiple messages of the advantage of any polynomial-time adversary $\mathcal{A}$ is negligible where the game is the following.

- The challenger gets a random uniform bit $b \in \{0,1\}$.
- The challenger chooses a key $k \in \{0,1\}^n$ uniformly.
- The adversary sends two messages (m_0^0, m_1^0) to $\mathcal{C}$.
- The challenger encodes m_b^0 using k and sends it to $\mathcal{A}$.
- The adversary can send as many two-messages packets (m_0^i, m_1^i) as it wants, and the challenger will give the cipher-text corresponding to m_b^i.
- Finally, the adversary will return a guess b' for b.

The advantage is defined as:

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

How do we formalize the previous attack? Consider three messages (m_0, m_1, m_2). Then the attacker will send (m_0, m_1) and (m_0, m_2). If the PRF F is deterministic, then $c_0 = c_1$ in the case $b = 0$. This gives a distinguisher.

The right way to encrypt with a PRF is the following.

Encryption. Given the inputs (k, m), take $r \leftarrow \mathcal{U}(\{0,1\}^n)$ and output $c = r||m \oplus F(k, r) \in \{0,1\}^{2n}$.

Decryption. Given $(k, c_1||c_2)$, output $F(k, c_1) \oplus c_2$.

We immediately have that $\mathrm{Dec}(k, \mathrm{Enc}(k, m)) = m$ for any (k, m).

The intuition for the security proof is the following. As the number of queries is less than a polynomial in n, it's very unlikely that two queries are being replied to with the same r. If the rs are distinct, then $F(k, r)$ plays the role of a mask and hides m.

If I encrypt a message of length ℓ plain-text by using this t times, then the output has size 2ℓ.

[**Something to fill here**...]

3 Message Authentication Codes, MACs

The goal of MACs is to provide *integrity* and *authenticity*.

Definition 3.1. A *MAC* is a triple of poly-time algorithms

$$(\text{KeyGen}, \text{Sign}, \text{Verif})$$

such that:

- ▷ $\text{KeyGen}(1^\lambda)$ takes as input the security parameter (in unary) and outputs a key $k \in \{0,1\}^s$;
- ▷ $\text{Sign}(k, \mu)$ takes as inputs a key k, and a message $\mu \in \{0,1\}^n$, and outputs a tag $t \in \{0,1\}^m$;
- ▷ $\text{Verify}(k, \mu, t)$ that takes as input a key k, a message μ and a tag t, and outputs a bit in $\{0,1\}$.

We say that a MAC is *correct* if, for every key k output by KeyGen, for all message μ,

$$\text{Verify}(k, \mu, \text{Sign}(k, \mu)) = 1.$$

The security is defined with an experiment:

- ▷ A challenger $\mathscr{C}$ creates a key k with KeyGen( ).
- ▷ An adversary $\mathscr{A}$ gives a message μ_1 to $\mathscr{C}$.
- ▷ Then $\mathscr{C}$ sends back $t_1 := \text{Sign}(k, \mu_1)$.

- ▷ After, $\mathcal{A}$ gives a message μ_2 to $\mathcal{C}$.
- ▷ And $\mathcal{C}$ sends back $t_2 := \text{Sign}(k, \mu_2)$.
- ▷ *etc.*
- ▷ Finally, $\mathcal{A}$ sends a pair $(\mu^\star, t^\star)$ to $\mathcal{C}$.

The goal of $\mathcal{A}$ is to create (forge) a new valid message-tag pair. The adversary $\mathcal{A}$ will win if $\text{Verify}(k, \mu^\star, t^\star) = 1$ and $(\mu^\star, t^\star) \neq (\mu_i, t_i)$ for every i.

The MAC is secure if, for any poly-time adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins is negligible. We call this *sEU-CMA security* (strong existential unforgeability under chosen message attacks).

We also define *EU-CMA* security: it is a variant where the success conditions are

$$\text{Verify}(k, \mu^\star, t^\star) = 1 \qquad \text{and} \qquad \mu^\star \neq \mu_i \quad \forall i.$$

We have that sEU-CMA security implies EU-CMA security.

PRF-base MAC for fixed-length messages.

We can proceed like the following:

- ▷ KeyGen( ), it samples $k \leftarrow \mathcal{U}(\{0,1\}^s)$;
- ▷ $\text{Sign}(k, \mu)$, it returns $t \leftarrow F(k, \mu)$;
- ▷ $\text{Verify}(k, \mu, t)$, it tests if $t \stackrel{?}{=} F(k, \mu)$.

This way, a PRF is a MAC.

Why is it a secure MAC ? Let's assume we have a sEU-CMA adversary $\mathcal{A}$ and see if we can use it to break the PRF.

Consider the experiment Exp_0—the genuine sEU-CMA experiment—where $\mathcal{C}$ samples a key $k \leftarrow \mathcal{U}(\{0,1\}^s)$, then $\mathcal{A}$ makes queries μ_i (than can depend on results of previous ones) and gets back $t_i \leftarrow F(k, \mu_i)$.

Finally $\mathcal{A}$ sends $\mathcal{C}$ a "forged signature" $(\mu^\star, t^\star)$. The adversary will win if $F(k, \mu^\star) = t^\star$ and $(\mu_i, t_i) \neq (\mu^\star, t^\star)$.

Now, consider experiment Exp_1, where $\mathcal{C}$ (lazily) gets a uniform $f : \{0,1\}^n \to \{0,1\}^m$. When answering $\mathcal{A}$'s queries, $\mathcal{C}$ will use $t_i \leftarrow f(\mu_i)$. Finally $\mathcal{A}$ sends $\mathcal{C}$ a "forged signature" $(\mu^\star, t^\star)$. The adversary will win if $f(\mu^\star) = t^\star$ and $(\mu_i, t_i) \neq (\mu^\star, t^\star)$.

I will stop taking notes for the Cryptography and Security course, as I will no longer be following it. Some great lecture notes can be found in the AliENS GitLab (ENS students only):

https://gitlab.aliens-lyon.fr/di-students/cours-m1/-/tree/2020-2021/s2/CS/2019-2020

Farewell everyone!