"SAPIENZA" UNIVERSITY OF ROME
FACULTY OF INFORMATION ENGINEERING, INFORMATICS AND STATISTICS
DEPARTMENT OF COMPUTER SCIENCE

Cryptography

Lecture notes integrated with the book "Introduction to Modern Cryptography", J. Katz, Y. Lindell

Author
Simone Bianco

December 23, 2025

Contents

Information and Contacts

Personal notes and summaries collected as part of the *Cryptography* course offered by the degree in Computer Science of the University of Rome "La Sapienza".

Further information and notes can be found at the following link:
https://github.com/Exyss/university-notes. Anyone can feel free to report inaccuracies, improvements or requests through the Issue system provided by GitHub itself or by contacting the author privately:

- Email: **bianco.simone@outlook.it**
- LinkedIn: **Simone Bianco**

The notes are constantly being updated, so please check if the changes have already been made in the most recent version.

Suggested prerequisites:

Algebra and Computational Complexity

Licence:

These documents are distributed under the **GNU Free Documentation License**, a form of copyleft intended for use on a manual, textbook or other documents. Material licensed under the current version of the license can be used for any purpose, as long as the use meets certain conditions:

- All previous authors of the work must be **attributed**.
- All changes to the work must be **logged**.
- All derivative works must be **licensed under the same license**.
- The full text of the license, unmodified invariant sections as defined by the author if any, and any other added warranty disclaimers (such as a general disclaimer alerting readers that the document may not be accurate for example) and copyright notices from previous versions must be maintained.
- Technical measures such as DRM may not be used to control or obstruct distribution or editing of the document.

<div style="text-align: right; font-size: 4em; color: gray;">1</div>

Introduction to modern cryptography

1.1 Definitions, assumptions and notation

Cryptography is the branch of computer science that practices and studies techniques for secure communication in the presence of adversarial behavior (e.g. altering the integrity of the message, reading the content of the message, ...). Cryptography focuses on two main goals:

1. **Confidential communication**: the property of making the original message impossible to read even if an outsider finds out the *ciphertext*, i.e. the encrypted message

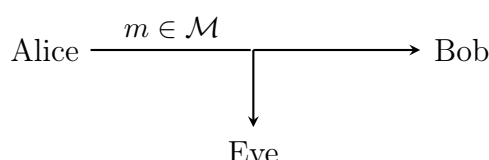$$\text{Alice} \xrightarrow{\quad m \in \mathcal{M} \quad} \text{Bob} \\ \downarrow \\ \text{Eve}$$

Figure 1.1: Without an encryption scheme, Eve – an evildoer – may be able to read the message that Alice is sending to Bob.

2. **Message integrity**: the capability of ensuring that the original message has not been altered even if an outsider intercepts the ciphertext

$$\text{Alice} \xrightarrow{\quad m \in \mathcal{M} \quad} \dashrightarrow \text{Bob} \\ \downarrow \qquad \qquad \uparrow \\ \text{Eve} \xrightarrow{\quad m' \in \mathcal{M} \quad}$$
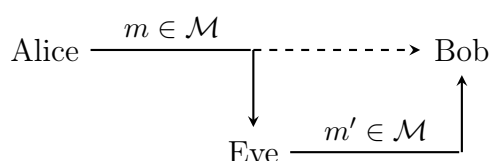
Figure 1.2: Without an encryption scheme, Eve may be able to intercept and alter the message that Alice is sending to Bob.

Before the 50's, cryptography was considered an *art* for geniouses capable of encrypting and decrypting messages written by other geniouses of the field. In modern days, cryptography became a *mathematical science* with precise definitions and proofs. These mathematical tools separate in two types: **unconditional proofs** and **conditional proofs**.

The former type referres to proofs where no assumptions are made, focusing on showing that something is possible without caring about it being inefficient (hence we have theoretically infinite resources at our disposal). The latter, instead, uses real-world assumptions that are believed to be true in order to prove real-world results. The typical example is the $P \neq NP$ assumption, i.e. that there are some problems that are verifiable in polynomial time but not solvable in polynomial time. Many cryptosystems are based on the assumption that prime factorization is hard to solve but easy to verify. This is equivalent to assuming that $FACTORING \in NP - P$ (only $FACTORING \in NP$ has been proven), making the assumption $P \neq NP$ fundamental. Making a crytosystem easy to verify allows us to make it impossible to access a resource without using knowing the solution to the authentication phase.

> **Theorem 1.1**
>
> Every cryptosystem based on prime factorization is "secure" if $FACTORING \notin P$

Proof (sketch). By contrapositive, suppose that there is a cryptosystem Π that is not "secure", meaning that there is a polynomial time algorithm A that is capable of "breaking" Π. Then, for each number $n \in \mathbb{N}$ we can forge a message m based on n and use the output $A(m)$ to find the prime factors of n, concluding that $FACTORING \in P$. $\square$

These two goals will be discussed under two main types of cryptosystems:

- **Symmetric cryptography**: both ends of the communication share a single common key that is random and unknown to any outsider.

- **Asymmetric cryptography**: both ends of the communication have each their own key pair (pk, sk) where pk is *public*, i.e. known by everyone, and sk is *secret*, i.e. known only by the owner.

Throughout this work we'll use the following notation to talk about cryptographic systems:

- $\mathcal{M}$ is the message space, i.e. the set of all strings of messages that can be sent between two parties.

- $\mathcal{C}$ is the ciphertext space, i.e. the set of all strings of ciphertexts that can be produced by a cryptosystem.

- $\mathcal{K}$ is the key space, i.e. the set of all strings of keys that can be used in a cryptographic system.

- $\mathcal{H}$ is the hashing function space, i.e. the set of all hash functions that can be used by a cryptosystem.

2

Information-theoretic cryptography

2.1 Perfect secrecy and Shannon's theorem

For now, we'll focus on symmetric cryptography, i.e. cryptosystems where a shared secret key is used for both encryption and decryption. It is widely used due to its speed and efficiency, especially in encrypting large volumes of data. A core model of this approach is **Secret Key Encryption (SKE)**, which includes three main components:

- A shared *secret key* $K \in_R \mathcal{K}$ chosen uniformly at random

- An *encryption function* $\mathrm{Enc} : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ that transforms plaintext into ciphertext

- A *decryption function* $\mathrm{Dec} : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$ that transforms a ciphertext into plaintext

In order to be functional, SKEs must be **correct**, meaning that if a message $m \in \mathcal{M}$ gets encrypted with $K \in_R \mathcal{K}$ obtaining the ciphertext $c \in \mathcal{C}$, the decryption process over c using the same key K must give back the original message.

> **Definition 2.1: Correctness in SKEs**
>
> An SKE $\Pi = (\mathrm{Enc}, \mathrm{Dec})$ is said to be correct if $\forall m \in \mathcal{M}, \forall K \in_R \mathcal{K}$ it holds that:
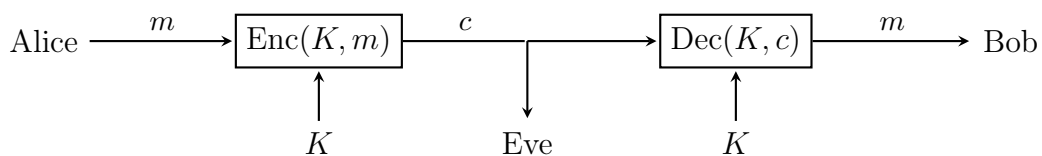>
> $$\mathrm{Dec}(K, \mathrm{Enc}(K, m)) = m$$

Figure 2.1: Example of SKE with perfect secrecy. Even if Eve intercepts the ciphertext, she cannot obtain the original message since she doesn't know the key.

Originally proposed in the 19th century by the homonym cryptographer, **Kerckhoffs's principle** is a foundational concept in cryptography which asserts that the security of a cryptographic system should depend <u>only</u> on the secrecy of the key. In other words, a system should be secure even if everything about the system is known publicly, except for the secret key. The principle was later succinctly restated by Claude Shannon as "the enemy knows the system".

During his work, Shannon also proposed a formal definition of **perfect secrecy**, i.e. a property that fully respects the concept behind Kerckhoff's principle. Shannon's formulation states that the probability of m being the communicated message is equal to the probability of m being the communicated message even when the corresponding ciphertext c is known. In other words, no ciphertext reveals additional information over any message.

> **Definition 2.2: Perfect secrecy**
>
> Let $\Pi = (\text{Enc}, \text{Dec})$ be an SKE. Let M be a random variable over $\mathcal{M}$ and let C be another random variable defined as $C = \text{Enc}(K, M)$, for some key $K \in_R \mathcal{K}$. We say that Π has perfect secrecy when $\forall m \in \mathcal{M}$ and $\forall c \in \mathcal{C}$ it holds that:
>
> $$\Pr[M = m] = \Pr[M = m \mid C = c]$$

Shannon proved that such definition is achievable by some cryptosystems, but it comes with inheritent practical limitations. Suprisingly, even a simple SKE as the **One Time Pad (OTP)** system has perfect secrecy. In this system we assume that everything is a binary string of the same length, i.e. that $\mathcal{M} = \mathcal{K} = \mathcal{C} = \{0,1\}^n$ for some $n \in \mathbb{N}$. The encryption and decryption functions are defined as follows:

$$\text{Enc}(K, m) = K \oplus m \qquad \text{Dec}(K, c) = K \oplus c$$

By properties of the bit-wise XOR function, it's easy to see that OTP is complete:

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

In order to prove that OTP has perfect secrecy, we start by proving two equivalent definitions of perfect secrecy. In particular, states that for every pair of messages every ciphertext has the same probability of being the output of the encoding function when applied of both messages.

Lemma 2.1: Perfect secrecy (eq. definitions)

Let $\Pi = (\text{Enc}, \text{Dec})$ be an SKE. Let M be a random variable over $\mathcal{M}$ and let C be another random variable defined as $C = \text{Enc}(K, M)$, for some key $K \in_R \mathcal{K}$. The following statements are equivalent:

1. Π has perfect secrecy

2. M and C are independent

3. $\forall m, m' \in \mathcal{M}$ and $\forall c \in \mathcal{C}$ it holds that:
$$\Pr_{K \in_R \mathcal{K}}[\text{Enc}(K, m) = c] = \Pr_{K \in_R \mathcal{K}}[\text{Enc}(K, m') = c]$$

Proof.

1. $\implies$ 2. Suppose that Π has perfect secrecy. Then, through definition of conditional probability we get that:
$$\Pr[M = m] = \Pr[M = m \mid C = c] = \frac{\Pr[M = m, C = c]}{\Pr[C = c]}$$

which implies that:
$$\Pr[M = m] \cdot \Pr[C = c] = \Pr[M = m, C = c]$$

concluding that M and C are independent

2. $\implies$ 3. Suppose that M and C are independent. Fix $m, m' \in \mathcal{M}$ and $c \in \mathcal{C}$. Through event manipulation and independency of M and C we obtain that:
$$\Pr_{K \in_R \mathcal{K}}[\text{Enc}(K, m) = c] = \Pr_{K \in_R \mathcal{K}}[\text{Enc}(K, M) = c \mid M = m]$$
$$= \Pr_{K \in_R \mathcal{K}}[C = c \mid M = m]$$
$$= \Pr_{K \in_R \mathcal{K}}[C = c]$$

Proceding in the same way with $\text{Enc}(K, m')$, we conclude that:
$$\Pr_{K \in_R \mathcal{K}}[\text{Enc}(K, m) = c] = \Pr_{K \in_R \mathcal{K}}[\text{Enc}(K, m') = c]$$

3. $\implies$ 1. Assume the third statement holds. Fix $m \in \mathcal{C}$ and $c \in \mathcal{C}$.

   Claim: $\Pr[C = c] = \Pr[C = c \mid M = m]$

Proof of the claim. Through the probability sum principle, we get that:

$$\Pr[C = c] = \sum_{m' \in \mathcal{M}} \Pr[C = c, M = m]$$

$$= \sum_{m' \in \mathcal{M}} \Pr[C = c \mid M = m'] \cdot \Pr[M = m']$$

$$= \sum_{m' \in \mathcal{M}} \Pr[\text{Enc}(K, M') = c \mid M = m'] \cdot \Pr[M = m']$$

$$= \sum_{m' \in \mathcal{M}} \Pr[\text{Enc}(K, m') = c] \cdot \Pr[M = m']$$

Through the hypothesis we also get that:

$$\Pr[C = c] = \sum_{m' \in \mathcal{M}} \Pr[\text{Enc}(K, m') = c] \cdot \Pr[M = m']$$

$$= \sum_{m' \in \mathcal{M}} \Pr[\text{Enc}(K, m) = c] \cdot \Pr[M = m']$$

$$= \Pr[\text{Enc}(K, m) = c] \cdot \sum_{m' \in \mathcal{M}} \Pr[M = m']$$

$$= \Pr[\text{Enc}(K, m) = c] \cdot 1$$
$$= \Pr[\text{Enc}(K, M) = c \mid M = m]$$
$$= \Pr[C = c \mid M = m]$$

$\square$

By definition of conditional probability, we know that:

$$\Pr[M = m \mid C = c] \cdot \Pr[C = c] = \Pr[M = m, C = c] = \Pr[C = c \mid M = m] \cdot \Pr[M = m]$$

which implies that:

$$\Pr[M = m] = \frac{\Pr[M = m \mid C = c] \cdot \Pr[C = c]}{\Pr[C = c \mid M = m]}$$

Finally, through the claim we conclude that:

$$\Pr[M = m] = \frac{\Pr[M = m \mid C = c] \cdot \Pr[C = c]}{\Pr[C = c \mid M = m]} = \Pr[M = m \mid C = c]$$

$\square$

Proposition 2.1

OTP has perfect security.

Proof. We'll prove that the third definition of Lemma 2.1 holds for OTP. Fix two messages $m, m' \in \mathcal{M}$ and a ciphertext $c \in \mathcal{C}$. Through definition of the OTP system and by the properties of the XOR function, we have that:

$$\Pr_{K \in_R \mathcal{K}}[\text{Enc}(K, m) = c] = \Pr_{K \in_R \mathcal{K}}[K \oplus m = c] = \Pr_{K \in_R \mathcal{K}}[K = m \oplus c] = 2^{-n}$$

Through the same argument, we also get that:

$$\Pr_{K \in_R \mathcal{K}}[\text{Enc}(K, m') = c] = 2^{-n}$$

$\square$

We'll now show the inheritent limitations of perfect secrecy. The key idea is simple: in order for M and C to be independent, the key cannot be shorter than the message. Otherwise, there will be some ciphertexts that are unreachable by some messages, revealing information on the system.

> ### Theorem 2.1: Shannon's perfect secrecy theorem
>
> Let $\Pi = (\text{Enc}, \text{Dec})$ be a non-trivial perfectly secret SKE. Then, it holds that $|\mathcal{K}| \geq |\mathcal{M}|$

Proof. Suppose that Π is a non-trivial perfectly secret. Fix any $c \in \mathcal{C}$ such that $\Pr[C = c] > 0$ (this is where non-triviality is required). Let $\mathcal{M}'$ be the set of possible decryptions over c, i.e. $\mathcal{M}' = \{\text{Dec}(K, c) \mid K \in \mathcal{K}\}$. We observe that $\mathcal{M}'$ contains at most one decryption for each key (some keys may yield the same decryption). By way of contradiction, suppose that $|\mathcal{K}| < |\mathcal{M}|$. Then, we get that $|\mathcal{M}'| \leq |\mathcal{K}| < |\mathcal{M}|$.

This implies that $\exists m \in \mathcal{M} - \mathcal{M}'$. Thus, there is a message that cannot be the result of applying the decryption function on c, meaning that $\Pr[M = m \mid C = c] = 0$. However, we know that, when no additional information is given, every message is uniform, hence $\Pr[M = m] = \frac{1}{\mathcal{M}}$, contradicting the fact that Π has perfect secrecy. $\square$

2.2 Message authentication codes

We'll now focus on the second key goal of cryptography: *message integrity*. The simplest way to reach such goal is through **Message Authentication Codes (MACs)**, an additional piece of information sent with the message that enables the receiver to assert that the message hasn't been altered.

For now, we'll start with a simple model that doesn't care about secrecy, but only about integrity. This type of MACs use a deterministic **tagging function** (usually implemented as an *hash function*) $\text{Tag} : \mathcal{K} \times \mathcal{M} \to \mathcal{T}$, where $\mathcal{T}$ is the tag space, i.e. the set of all tag strings.
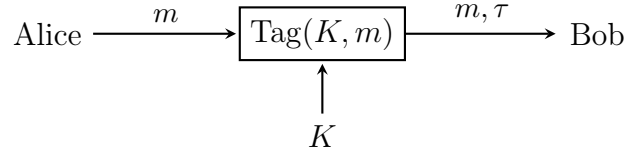
Figure 2.2: Example of an integrity-only MAC.

The idea behind tagging functions is simple: once the message and the tag have been received, Bob can re-compute the tag using the same key-message pair and compare it to the received tag. If the two tags are equal, Bob is sure that the message hasn't been altered. However, this simple idea can only work under the assumption of **unforgeability**:

- It should be hard to forge a valid tag τ for a message m when the key K is not known

- It should be hard to forge a valid tag τ for a message m even when a pair (m', τ') is known

In other words, unforgeability states that no pair should reveal no information about how the tags are computed. Without this property, an adversarial entity may be able to infer information on the shared key and/or the tagging function, using them to forge valid tags. If an entity is capable of forging a valid tag, it may intercept the message-tag pair, alter the message, forge a valid tag for the new message and send a new pair, fooling the receiver. We give a formal definition of a property that reflects this unforgeability aspect.

Definition 2.3: t-time ε-statistical security

We say that a MAC $\Pi = (\text{Tag})$ has t-time ε-statistical security when $\forall m, m_1, \ldots, m_t \in \mathcal{M}$ and $\forall \tau, \tau_1, \ldots, \tau_t \in \mathcal{T}$, where $m \neq m_i$ and $m_i \neq m_j$ for each $i \neq j$, it holds that:
$$\Pr_{K \in_R \mathcal{K}}[\text{Tag}(K, m) = \tau \mid \text{Tag}(K, m_1) = \tau_1, \ldots, \text{Tag}(K, m_t) = \tau_t] \leq \varepsilon$$

In simple terms, the above property states that, even when t message-tag pairs $(m_1, \tau_1), \ldots, (m_t, \tau_t)$ are known, the probability of a message-tag pair (m, τ) being possible is at most ε. Optimally, we want ε to be as small as possible and t to be as large as possible. However, it's easy to see that $\forall t \in \mathbb{N}$ it is impossible to get $\varepsilon = 0$ since a random $\tau \in \mathcal{T}$ always has probability at least $\frac{1}{|\mathcal{T}|}$ of being correct. Just as we did with perfect secrecy, we'll show that the notion of good statistical security is achievable, but it's highly inefficient in terms of key size.

Theorem 2.2

Any t-time $2^{-\lambda}$-statistically secure MAC must have a key of size $(t+1)\lambda$

Proof. Omitted. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\square$

A good enough 1-time statistically secure MAC is achievable through **pairwise independent hash functions**, i.e. a family of hash functions where each pair of functions forms a pair of independent random variables. This idea can also be expressed through joint uniform distributions, as in the below definition.

> **Definition 2.4: Pairwise independent hash functions**
>
> Let $\mathcal{H} = \{h_K : \mathcal{M} \to \mathcal{T}\}_{K \in \mathcal{K}}$ be a family of hash functions. We say that $\mathcal{H}$ is pairwise independent if $\forall m, m' \in \mathcal{M}$ with $m \neq m'$ it holds that the distribution $(h_K(m), h_K(m'))$ is uniform over $\mathcal{T} \times \mathcal{T}$ when $K \in_R \mathcal{K}$.
>
> *Note*: $h_K(m)$ and $h_K(m')$ denote two random variables in this context.

> **Theorem 2.3**
>
> Let $\mathcal{H} = \{h_K : \mathcal{M} \to \mathcal{T}\}_{K \in \mathcal{K}}$ be a family of pairwise indepependent hash functions. Then, $\mathcal{H}$ induces a 1-time $\frac{1}{|\mathcal{T}|}$-statistically secure MAC.

Proof. Fix $m \in \mathcal{M}$ and $\tau \in \mathcal{T}$. Let $\Pi = (\text{Tag})$ be the MAC defined as $\text{Tag}(K, m) = h_K(m)$. Since the joint probability of each pair of hash functions in $\mathcal{H}$ is pairwise uniformly distributed, we get that each hash function is also individually uniformly distributed. Hence, we derive that:

$$\Pr[\text{Tag}(K, m) = \tau] = \Pr[h_K(m) = \tau] = \frac{1}{|\mathcal{T}|}$$

Similarly, by pairwise independence $\forall m, m' \in \mathcal{M}$ with $m \neq m'$ and $\forall \tau, \tau' \in \mathcal{T}$ it holds that:

$$
\begin{aligned}
\Pr_{K \in_R \mathcal{K}}[\text{Tag}(K, m') = \tau', \text{Tag}(K, m) = \tau] &= \Pr_{K \in_R \mathcal{K}}[\text{Tag}(K, m') = \tau'] \cdot \Pr_{K \in_R \mathcal{K}}[\text{Tag}(K, m) = \tau] \\
&= \Pr_{K \in_R \mathcal{K}}[h_K(m') = \tau'] \cdot \Pr_{K \in_R \mathcal{K}}[h_K(m) = \tau] \\
&= \frac{1}{|\mathcal{T}|} \cdot \frac{1}{|\mathcal{T}|}
\end{aligned}
$$

Putting the two results together we get that:

$$\Pr[\text{Tag}(K, m') = \tau' \mid \text{Tag}(K, m) = \tau] = \frac{\Pr_{K \in_R \mathcal{K}}[\text{Tag}(K, m') = \tau', \text{Tag}(K, m) = \tau]}{\Pr[\text{Tag}(K, m) = \tau]} = \frac{1}{|\mathcal{T}|}$$

$$\square$$

After proving that pairwise independent hash function families form a good enough MAC, we're left with proving that such families exist. Suprisingly, these families can be easily constructed through prime numbers.

> **Proposition 2.2**
>
> Given a prime $p \in \mathbb{P}$, let $\mathcal{M} = \mathcal{T} = \mathbb{Z}_p$ and $\mathcal{K} = \mathbb{Z}_p^2$. Then, the family $\mathcal{H} = \{h_{(a,b)}\}_{(a,b) \in \mathbb{Z}_p^2}$ where $h_{(a,b)}(m) = am + b \pmod{p}$ is pairwise independent.

Proof. Fix $m, m' \in \mathbb{Z}_p$ with $m \neq m'$ and $\tau, \tau' \in \mathbb{Z}_p$. We'll show that the joint probability over the hash functions is uniform. First, we observe that:

$$\Pr_{(a,b) \in_R \mathbb{Z}_p^2}[h_{(a,b)}(m) = \tau, h_{(a,b)}(m') = \tau'] = \Pr_{(a,b) \in_R \mathbb{Z}_p^2}[am + b = \tau, am' + b = \tau']$$

$$= \Pr_{(a,b) \in_R \mathbb{Z}_p^2}\left[\begin{pmatrix} m & 1 \\ m' & 1 \end{pmatrix}\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \tau \\ \tau' \end{pmatrix}\right]$$

Since $m \neq m'$, we know that $\det \begin{pmatrix} m & 1 \\ m' & 1 \end{pmatrix} = m - m' \neq 0$, thus the matrix is invertible. This allows us to further manipulate the event and rewrite it in terms of the key:

$$\Pr_{(a,b) \in_R \mathbb{Z}_p^2}[h_{(a,b)}(m) = \tau, h_{(a,b)}(m') = \tau'] = \Pr_{(a,b) \in_R \mathbb{Z}_p^2}\left[\begin{pmatrix} m & 1 \\ m' & 1 \end{pmatrix}\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \tau \\ \tau' \end{pmatrix}\right]$$

$$= \Pr_{(a,b) \in_R \mathbb{Z}_p^2}\left[\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} m & 1 \\ m' & 1 \end{pmatrix}^{-1}\begin{pmatrix} \tau \\ \tau' \end{pmatrix}\right]$$

$$= \frac{1}{|\mathbb{Z}_p^2|}$$

concluding that $\mathcal{H}$ is pairwise independent. $\qquad\square$

2.3 Randomness extraction

We discussed how randomness can be used to generate secret keys over a probability distribution, but how a random value be generated? As we will discuss later, randomness is a <u>crucial</u> concept for secure cryptography. Clearly, there is no such concept as *real randomness*: even if the whole universe may appear chaotic, everything is technically deterministic. The process of generating a random-enough value is called **randomness extraction** and it is typically achieved by measuring physical quantities (noise, air humidity, ...) in order to produce a short unpredictable sequence of bits (e.g. 256 bits), which is expensive to generate and not necessarily uniform. This short "truly random" sequence gets usually expanded to any desired length – as long as it is polynomial with respect to the original lenght – through the use of a **pseudorandom generator (PRG)**. However, this process requires some strict computational assumptions, which we'll discuss later.

For now, we'll focus on understanding how to extract randomness from an unpredictable secure variable X. The first **extractor** that sparked the idea is Von Neumann's Extractor, which yields a fair random coin from an unpredictable unfair one. Let $B \in \{0, 1\}$ be the

random variable describing the unpredictable unfair coin, where $\Pr[B = 0] = p < \frac{1}{2}$. Let $Y \in \{0, 1\}$ be the random variable describing our new coin. The value of Y is determined by the following procedure. Sample two values b_1, b_2 from B at different times. If $b_1 = b_2$, Y assumes no value (marked as $Y =?$) and we repeat the sampling process. If not, $Y = 1$ if $b_1 = 0$ and $b_2 = 1$, otherwise $Y = 0$ if $b_1 = 1$ and $b_2 = 0$.

If the sampling process succeeds, i.e. Y assumes a value, we have that $\Pr[Y = 0] = p(1-p)$ and $\Pr[Y = 1] = (1 - p)p$, thus $\Pr[Y = 0] = \Pr[Y = 1]$. Moreover, we observe that the probability of $Y == ?$ being true for m consecutive tries is at most:

$$\Pr[Y =? \text{ for } m \text{ tries}] = (\Pr[Y == ?])^m = (1 - \Pr[Y = 0 \cup Y = 1])^m \leq (1 - 2p(1 - p))^m$$

As m grows to infinity, the latter probability goes to 0, making the even negligible. Implying that $\Pr[Y = 0]$ and $\Pr[Y = 1]$ tend to be $\frac{1}{2}$ due to them having the same probability and them being the only two outcomes. This makes Y a fair enough coin.

Our goal is to generalize this concept to any desider value, i.e. design an extractor Ext that uses a random variable X to output any desided uniform distribution $\text{Ext}(X)$. A good eye may recognize that this is clearly impossible to achieve unless the source is already truly unpredictable, which makes the extractor useless since we already have a truly random source. As for many computational and probabilistic concepts, we can only hope to achieve something that is good-enough for our purposes. This goodness is measured through a concept known as **min-entropy**, that being the largest value m having the property that each observation of X provides at least m bits of information.

> ### Definition 2.5: Min-entropy
>
> Given a random variable X, we define the min-entropy of X as:
> $$H_\infty(X) = -\log \max_x \Pr[X = x]$$

One way to justify the name of this operator is to compare it with the more standard definition of *entropy*, i.e. the expected value of $\log\left(\frac{1}{p_i}\right)$ over a distribution P:

$$H(P) = \sum_i p_i \log\left(\frac{1}{p_i}\right)$$

In the min-entropy, instead, we take take the minimum value of $\log\left(\frac{1}{p_i}\right)$, where:

$$\min_i \log\left(\frac{1}{p_i}\right) = \log\left(\min_i \frac{1}{p_i}\right) = -\log \max_i p_i$$

Consider a random variable $X \sim U_n$, where U_n is an uniform distribution over $\{0, 1\}^n$. In this case, we have that $\Pr[X = x] = 2^{-n}$ for all value x assumable by X. Hence, the min-entropy is:

$$H_\infty(X) = -\log \max_x \Pr[X = x] = -\log(2^{-n}) = n$$

This concludes that each observation of X provides at least n bits of information. Consider now a constant random variable X', i.e. $\Pr[X' = x^*] = 1$ for some fixed value x^* and $\Pr[X' = x] = 0$ for every $x^* \neq x$. It's easy to see that:

$$H_\infty(X') = -\log \max_{x'} \Pr[X' = x] = -\log 1 = 0$$

Concluding that each observation of X' provides at least 0 bits of information. This information-theoretic measure opens a new question regarding extractors: is there an extractor Ext^* that for any random variable X outputs an uniform distribution $Y = \text{Ext}(X)$ such that $H_\infty(X) \geq k$ for some value $k > 0$? Even under these constraints, the answer to this question is still negative. In particular, it's impossible to define such extractor even if we restrict our interest to extracting only one bit while revealing at least one bit less than the input length.

> **Proposition 2.3**
>
> There is no extractor Ext such that for every random variable X over $\{0,1\}^n$ with $H_\infty(X) \geq n-1$ it holds that $\text{Ext}(X)$ is a uniform distribution over $\{0,1\}$.

Proof. Let $\text{Ext} : \{0,1\}^n \to \{0,1\}$ be any extractor and let $b \in \{0,1\}$ be the output maximizing the cardinality of the preimage of the extractor, i.e. the set of inputs for which the extractor outputs b.

$$b = \arg\max_{b' \in \{0,1\}} \left| \text{Ext}^{-1}(b) \right|$$

By the pidgeonhole principle, we have that $\left| \text{Ext}^{-1}(b) \right| \geq \frac{|\{0,1\}^n|}{2} = 2^{n-1}$. Let X be a random variable uniform over $\text{Ext}^{-1}(b)$. Since X is uniform, we have that $H_\infty(X) \geq n-1$. However, we know that $\text{Ext}(X)$ isn't uniform due to the output being always b. This concludes that any extractor has always a bad input uniform random variable X that returns a non-uniform distribution $\text{Ext}(X)$. $\qquad\square$

Since we can't define an extractor that yields an uniform distribution, the best we can hope for is a distribution that is close enough to an uniform one. We use a standard measure for distribution similarity, called ε-**closeness**.

> **Definition 2.6: ε-closeness**
>
> Let X, X' be two random variables defined over the same set. We say that X and X' are ε-close, written as $X \sim_\varepsilon X'$ when their *statistical distance* $\text{SD}(X; X')$ is at most ε, where:
>
> $$\text{SD}(X; X') = \frac{1}{2} \sum_x \left| \Pr[X = x] - \Pr[X' = x] \right|$$

The concept of ε-closeness between two random variables is equivalent to saying that every *unbounded adversary* A, i.e. an entity with unlimited computational power that

wants to break our system, cannot distinguish whether a value x has been sampled from X or X'.

$$|\Pr[A(x) = 1 : x \in_R X] - \Pr[A(x) = 1 : x \in_R X']| \leq \varepsilon$$

Definition 2.7: Deterministic extractor

Let S be a random variable, referred to as *seed*. We say that $\text{Ext} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^\ell$ is a (k, ε)-extractor if for every random variable X with $H_\infty(X) \geq k$ it holds that $(S, \text{Ext}(S, X)) \sim_\varepsilon (S, U_\ell)$ when $S \sim U_d$.

We notice that the condition $(S, \text{Ext}(S, X)) \sim_\varepsilon (S, U_\ell)$ implies that the seed must be *public*. This requirement is forced in order to avoid trivial extractors such as $\text{Ext}(S, X) = S$. The idea is to use a source of randomness with high min-entropy in order to force that the output of a "good" hash function is statistically close to uniform – even if the input seed wasn't. This result is known as the **left-over hash lemma**.

The goodness of the hash functions is measured in terms of **low collision probability**, defined as the expected value that a random variable Y over a set $\mathcal{Y}$ and a copy Y' of Y, giving two i.i.d (identical and independend distributions) variables, return the same result:

$$\text{Col}(Y) = \Pr[Y = Y']$$

In particular, we observe that since Y and Y' are i.i.d. we get that:

$$\text{Col}(Y) = \Pr[Y = Y'] = \sum_{y \in \mathcal{Y}} \Pr[Y = y, Y' = y] = \sum_{y \in \mathcal{Y}} \Pr[Y = y]^2$$

Before proving the left-over hash lemma, we prove the following relationship commonly used in statistical analysis.

Proposition 2.4

Let Y be a random variable over a set $\mathcal{Y}$ and assume $\text{Col}(Y) = \frac{1}{|\mathcal{Y}|}(1 + 4\varepsilon^2)$ for some $\varepsilon \in \mathbb{R}_{>0}$. Then, it holds that $\text{SD}(Y, U) \leq \varepsilon$, where U is the uniform distribution over $\mathcal{Y}$.

Proof. We start by observing that:

$$\text{SD}(Y; U) = \frac{1}{2} \sum_{y \in \mathcal{Y}} |\Pr[Y = y] - \Pr[U = y]| = \frac{1}{2} \sum_{y \in \mathcal{Y}} \left| \Pr[Y = y] - \frac{1}{\mathcal{Y}} \right|$$

For each $y \in \mathcal{Y}$, fix $q_y = \Pr[Y = y] - \frac{1}{|Y|}$ and $s_y = \text{sign}(q_y)$. Let $\overrightarrow{q} = \begin{bmatrix} q_{y_1} & \cdots & q_{y_{|Y|}} \end{bmatrix}$ and $\overrightarrow{s} = \begin{bmatrix} s_{y_1} & \cdots & s_{y_{|Y|}} \end{bmatrix}$. We notice that:

$$\text{SD} = \frac{1}{2} \sum_{y \in \mathcal{Y}} \left| \Pr[Y = y] - \frac{1}{\mathcal{Y}} \right| = \frac{1}{2} \sum_{y \in \mathcal{Y}} q_y s_y = \frac{1}{2} \langle \overrightarrow{q}, \overrightarrow{s} \rangle$$

By the *Cauchy-Swartz inequality* (which we won't prove), we get that:

$$\text{SD}(Y; U) = \frac{1}{2} \langle \vec{q}, \vec{s} \rangle \leq \frac{1}{2} \sqrt{\langle \vec{q}, \vec{q} \rangle \langle \vec{s}, \vec{s} \rangle} = \frac{1}{2} \sqrt{\left(\sum_{y \in \mathcal{Y}} q_y^2 \right) |\mathcal{Y}|}$$

Claim: $\sum_{y \in \mathcal{Y}} q_y^2 \leq \dfrac{4\varepsilon^2}{|Y|}$

Proof of the claim. Through algebraic manipulation we get that:

$$\sum_{y \in \mathcal{Y}} q_y^2 = \sum_{y \in \mathcal{Y}} \left(\Pr[Y = y] - \frac{1}{\mathcal{Y}} \right)^2$$

$$= \sum_{y \in \mathcal{Y}} \left(\Pr[Y = y]^2 - \frac{2}{|\mathcal{Y}|} \Pr[Y = y] + \frac{1}{|\mathcal{Y}|^2} \right)$$

$$= \sum_{y \in \mathcal{Y}} \Pr[Y = y]^2 - \sum_{y \in \mathcal{Y}} \frac{2}{|\mathcal{Y}|} \Pr[Y = y] + \sum_{y \in \mathcal{Y}} \frac{1}{|\mathcal{Y}|^2}$$

$$= \sum_{y \in \mathcal{Y}} \Pr[Y = y]^2 - \frac{2}{|\mathcal{Y}|} + \frac{1}{|\mathcal{Y}|}$$

$$= \text{Col}(Y) - \frac{1}{\mathcal{Y}}$$

By hypothesis, we know that the collision probability of Y is bounded, concluding that:

$$\sum_{y \in \mathcal{Y}} q_y^2 = \text{Col}(Y) - \frac{1}{\mathcal{Y}} \leq \frac{1}{|\mathcal{Y}|}(1 + 4\varepsilon^2) - \frac{1}{\mathcal{Y}} = \frac{4\varepsilon^2}{|\mathcal{Y}|}$$

$\square$

Through the claim we directly conclude that:

$$\text{SD}(Y; U) = \frac{1}{2} \sqrt{\left(\sum_{y \in \mathcal{Y}} q_y^2 \right) |\mathcal{Y}|} \leq \frac{1}{2} \sqrt{\frac{4\varepsilon^2}{|\mathcal{Y}|} |Y|} = \varepsilon$$

$\square$

> **Lemma 2.2: Left-over hash lemma**
>
> Let $\mathcal{H} = \{h_S : \{0,1\}^n \to \{0,1\}^\ell\}_{S \in \{0,1\}^d}$ be a pairwise independent hash function. Let X be a random variable with $H_\infty(X) \geq k$. Then, for each $S \in \{0,1\}^d$ it holds that $\text{Ext}(S, X) = h_S(X)$ is a (k, ε)-extractor for $k \geq \ell + 2\log\left(\frac{1}{\varepsilon}\right) - 2$

Proof. Fix $S \in \{0, 1\}^d$ and two random variables X, X'. Let $Y = (S, \text{Ext}(S, X))$ and let Y' be a copy of Y defined as and $Y' = (S', \text{Ext}(S', X'))$. We observe that:

$$
\begin{aligned}
\text{Col}(Y) &= \Pr[Y = Y'] \\
&= \Pr[S = S', \text{Ext}(S, X) = \text{Ext}(S', X')] \\
&= \Pr[S = S', h_S(X) = h_{S'}(X')]
\end{aligned}
$$

Since S and S' are independend from X and X', we get that:

$$
\begin{aligned}
\text{Col}(Y) &= \Pr[S = S', h_S(X) = h_{S'}(X')] \\
&= \Pr[S = S', h_S(X) = h_S(X')] \\
&= \Pr[S = S'] \cdot \Pr[h_S(X) = h_S(X')] \\
&= 2^{-d} \cdot \Pr[h_S(X) = h_S(X')] \\
&= 2^{-d} \cdot (\Pr[X = X', h_S(X) = h_S(X')] + \Pr[X \neq X', h_S(X) = h_S(X')]) \\
&= 2^{-d} \cdot (\Pr[X = X'] + \Pr[X \neq X', h_S(X) = h_S(X')]) \\
&= 2^{-d} \cdot (\Pr[X = X'] + 2^{-\ell})
\end{aligned}
$$

By hypothesis, we know that $H_\infty(X) \geq k$. This implies that $\Pr[X = X'] \leq 2^{-k}$ since an adversary cannot guess whether a sample comes from X or X' with probability greater than 2^{-k}.

$$
\begin{aligned}
\text{Col}(Y) &= 2^{-d} \cdot (\Pr[X = X'] + 2^{-\ell}) \\
&= 2^{-d} \cdot (2^{-k} + 2^{-\ell}) \\
&= \frac{1}{2^{d+\ell}}(2^{\ell-k} + 1)
\end{aligned}
$$

Finally, by hypothesis we have conclude that:

$$
\text{Col}(Y) \leq \frac{1}{2^{d+\ell}}(2^{\ell-k} + 1) \leq \frac{1}{2^{d+\ell}}\left(2^{2-2\log\left(\frac{1}{\varepsilon}\right)} + 1\right) = \frac{4\varepsilon^2 + 1}{|\mathcal{Y}|}
$$

$\square$

2.4 Solved exercises

Problem 2.1

Given a prime $p \in \mathbb{P}$, let $\mathcal{M} = \mathcal{T} = \mathbb{Z}_p$ and $\mathcal{K} = \mathbb{Z}_p^2$. Prove that the hash family $\mathcal{H} = \{h_{(a,b)}\}_{(a,b)\in\mathbb{Z}_p^2}$, where $h_{(a,b)}(m) = am + b \pmod{p}$, cannot be a 2-time statistically secure MAC

Solution. Suppose that the values $h_{(a,b)}(m_1) = \tau_1$ and $h_{(a,b)}(m_2) = \tau_2$, where $m_1 \neq m_2$, are known to an adversary. Hence, the adversary knows that:

$$
\begin{cases} am_1 + b \equiv \tau_1 \pmod{p} \\ am_2 + b \equiv \tau_2 \pmod{p} \end{cases} \implies a(m_1 - m_2) \equiv \tau_1 - \tau_2 \pmod{p}
$$

Since $m_1 \neq m_2$, the value $m_1 - m_2$ is invertible in $\mathbb{Z}_p$, the value $Q \equiv (\tau_1 - \tau_2)(m_1 - m_2)^{-1}$ (mod p) (notice that $a \equiv Q$ (mod p)) can be easily computed. Once Q is known, b can be easily computed as $b \equiv \tau_1 - Qm_1$ (mod p). Therefore, we have that:

$$\Pr_{K \in_R \mathbb{Z}_p}[h_K(m) = \tau \mid h_K(m_1) = \tau_1, h_K(m_2) = \tau_2] = \begin{cases} 1 & \text{if } \tau \equiv Q(m - m_1) + \tau_1 \pmod p \\ 0 & \text{otherwise} \end{cases}$$

making $\mathcal{H}$ statistically insecure for every value $0 < \varepsilon < 1$.

Problem 2.2

Construct a 3-wise independent hash function family and prove it's correctness.

Solution. Given a prime $p \in \mathbb{P}$, let $\mathcal{M} = \mathcal{T} = \mathbb{Z}_p$ and $\mathcal{K} = \mathbb{Z}_p^3$. Consider the hash family $\mathcal{H} = \{h_{(a,b,c)}\}_{(a,b,c) \in \mathbb{Z}_p^3}$, where $h_{(a,b,c)}(m) = am^2 + bm + c$ (mod p).

Fix three values $x_1, x_2, x_3 \in \mathbb{Z}_p$ such that $x_i \neq x_j$ for all $i \neq j$. Fix $\tau_1, \tau_2, \tau_3 \in \mathbb{Z}_p$. We observe that:

$$\Pr_{(a,b,c) \in_R \mathbb{Z}_p^3}\begin{bmatrix} h_{(a,b,c)}(x_1) = \tau_1 \\ h_{(a,b,c)}(x_2) = \tau_2 \\ h_{(a,b,c)}(x_3) = \tau_3 \end{bmatrix} = \Pr_{(a,b,c) \in_R \mathbb{Z}_p^3}\begin{bmatrix} ax_1^2 + bx_1 + c = \tau_1 \\ ax_2^2 + bx_2 + c = \tau_2 \\ ax_3^2 + bx_3 + c = \tau_3 \end{bmatrix}$$

$$= \Pr_{(a,b,c) \in_R \mathbb{Z}_p^3}\left[\begin{pmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{pmatrix}\begin{pmatrix} a \\ c \\ b \end{pmatrix} = \begin{pmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{pmatrix} \right]$$

Since $x_i \neq x_j$ for all $i \neq j$, we observe that the matrix is invertible:

$$\det\begin{pmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{pmatrix} = x_1^2 \det\begin{pmatrix} x_2 & 1 \\ x_3 & 1 \end{pmatrix} - x_2^2 \det\begin{pmatrix} x_1 & 1 \\ x_3 & 1 \end{pmatrix} + x_3^2 \det\begin{pmatrix} x_1 & 1 \\ x_2 & 1 \end{pmatrix}$$

$$= x_1^2 x_2 - x_1^2 x_3 - x_2^2 x_1 + x_2^2 x_3 + x_3^2 x_1 - x_3^2 x_2$$

$$= (x_1 - x_2)(x_1 - x_3)(x_2 - x_3)$$

$$\neq 0$$

concluding that $\mathcal{H}$ is 3-wise independent:

$$\Pr_{(a,b,c) \in_R \mathbb{Z}_p^3}\begin{bmatrix} h_{(a,b,c)}(x_1) = \tau_1 \\ h_{(a,b,c)}(x_2) = \tau_2 \\ h_{(a,b,c)}(x_3) = \tau_3 \end{bmatrix} = \Pr_{(a,b,c) \in_R \mathbb{Z}_p^3}\left[\begin{pmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{pmatrix}\begin{pmatrix} a \\ c \\ b \end{pmatrix} = \begin{pmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{pmatrix} \right]$$

$$= \Pr_{(a,b,c) \in_R \mathbb{Z}_p^3}\left[\begin{pmatrix} a \\ c \\ b \end{pmatrix} = \begin{pmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{pmatrix}^{-1}\begin{pmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{pmatrix} \right]$$

$$= \frac{1}{|\mathbb{Z}_p^3|}$$

<div style="text-align: right; font-size: 4em; color: gray;">3</div>

Computational security

3.1 One-way functions and Impagliazzo's worlds

In the previous chapter we discussed how, without computational assumptions, symmetric encryption and random generation can be achieved with some strong limitations. For privacy and integrity, we saw how the message length and the key length must be at least equal in order to guarantee a 1-time security (no guarantees for t-time security with $t > 1$). For randomness, instead, we saw how we can't extract more than k bits when the min-entropy is k.

Our next objective is to overcome all these limitation (or at least reduce them). This can be achieved at the price of two very strong assumptions:

1. The adversary is *computationally bounded*, i.e. it has limited resources

2. There are *hard* problems

In other words, we'll prove results of the following form. "if problem X is hard against efficient solvers then cryptosystem Π is secure against efficient adversary". The direct consequence of these results is their contrapositive: if the system Π were ever to be proved insecure, there would be an efficient solution for problem X. Depending on the "level of hardness" of problem X, this would have groundbreaking consequences (e.g. X could be a problem that implies $\mathsf{P} = \mathsf{NP}$, factoring is easy, discrete-log is easy, ...).

Someone may ask "can't we just assume that $\mathsf{P} \neq \mathsf{NP}$ and prevent all of this?" The answer is *yes*, we could, but this assumption would be *too weak* for our necessities. In fact, to achieve good security we require something that isn't directly implied by $\mathsf{P} \neq \mathsf{NP}$: the existence of **one-way functions (OWF)**. In simple terms, a functions $f : \{0,1\}^n \to \{0,1\}^m$ is said to be *one-way* when it can be efficiently computed but hard to invert (meaning that f^{-1} cannot be efficiently computed).

It's easy to see that assuming the existence of OWFs implies that $\mathsf{P} \neq \mathsf{NP}$. This is due to the contrapositive statement: if $\mathsf{P} = \mathsf{NP}$ then OTWs cannot exist since for each function f we can efficiently verify if $f(x) = y$ in order to compute $f^{-1}(y) = x$. However, the

converse statement doesn't hold: assuming $P \neq NP$ could still imply that OWFs don't exist. In the article A Personal View of Average-Case Complexity presented at the 1995 Complexity Conference, Russell Impagliazzo describes **five possible worlds** that we may be living in and their implications to computer science:

- *Algorithmica*: there are no hard problems ($P = NP$) and OWFs cannot exist.

- *Heuristica*: there are hard problems ($P \neq NP$) but can they solved efficiently on average

- *Pessiland*: there are hard problems ($P \neq NP$) but OWFs don't exist

- *Minicrypt*: one-way functions exist but public-key cryptography is impossible

- *Cryptomania*: one-way functions exist and public-key cryptography is possible

Impagliazzo does not guess which world we live in, but describes *Pessiland* as the worst one out of all possible worlds: in this scenario, not only can we not solve hard problems on average but we apparantly do not get any cryptographic advantage from the hardness of these problems. Today, most computer scientists believe (and hope) that our world corresponds to either *Cryptomania* or *Minicrypt*.

All the results will be discussed with respect to a fixed standard computational model: the *Turing machine* (TM). From now on, *efficient computation* will refer to a computation made under a polynomial amount of steps with respect to the input size, i.e. if the input x has length n then poly(n)-time referres to at most n^c steps, for some $c \in \mathbb{N}$.

As mentioned before, the adversary we'll also assume that the adversary has limited resources. However, we'll be generous with them: the adversary is allowed to use a poly(n) amount of time and a poly(n) amount of *random bits*. This implies that their computational model is a *probabilistic polynomial time (PPT)* TM. To formally define the concept of OWFs, we'll use **negligible functions**, i.e. functions $\varepsilon : \mathbb{N} \to \mathbb{R}$ such that $\forall c \in \mathbb{N}$ there is a value $\lambda_0 \in \mathbb{N}$ for which $\forall \lambda > \lambda_0$ it holds that

$$\varepsilon(\lambda) < \frac{1}{\lambda^c}$$

> **Definition 3.1: One-way functions**
>
> We say that a function $f : \{0,1\}^n \to \{0,1\}^m$ is a one-way function if:
>
> - f is computable in poly(n)-time
>
> - for every PPT algorithm A there is a negligible function negl(n) such that:
>
> $$\Pr_{x \in_R U_n}[f(x') = y : y = f(x); x' \leftarrow A(y)] \leq \text{negl}(n)$$
>
> *Note*: $x \leftarrow A(y)$ is read as "x is the output of $A(y)$"

3.2 Pseudorandom generators

In the previous chapter we discussed how randomness is essential for good cryptography, even though true randomness cannot be achieved. To get a good enough approximation of true randomness, modern systems use **pseudo-randomness**, i.e. efficient deterministic algorithms that generate a sequence of bits that look unpredictable but aren't truly random. By definition, if enough time and resources are permitted an adversary is clearly able to learn how a pseudorandom algorithm work through bruteforce. Hence, we'll restrict that our adversary doesn't have such time and resources, i.e. their computation must also be efficient.

The idea behind pseudo-randomness is to generate a "non-random" probability distribution (this doesn't really mean anything, but you get the idea) that is **computationally indistinguishable** from a truly random probability distribution. The formal definition of computational indistinguishability refers to *probability ensembles*, i.e. infinite sequences of probability distributions $\mathcal{X} = \{X_n\}_{n \in \mathbb{N}}$, where each X_n is a distribution over $\{0,1\}^n$. This multi-length definition takes into account the fact that the distribution may change over the input length.

> ### Definition 3.2: Computational indistinguishability
>
> Let $\mathcal{X} = \{X_n\}_{n \in \mathbb{N}}$ and $\mathcal{Y} = \{Y_n\}_{n \in \mathbb{N}}$ be two probability ensambles. We say that $\mathcal{X}$ and $\mathcal{Y}$ are computationally indistinguishable, written as $\mathcal{X} \approx_c \mathcal{Y}$, if for every PPT algorithm D there is a negligible function $\mathrm{negl}(n)$ such that $\forall n \in \mathbb{N}$ it holds that:
>
> $$|\Pr[D(z) = 1 : z \in_R X_n] - \Pr[D(z) = 1 : z \in_R Y_n]| \leq \mathrm{negl}(n)$$

We observe that we already discussed a definition similar to the one above when we talked about ε-closeness. In fact, this can be seen as a practical computational counterpart of that definition (notice how the adversary is now restricted to PPT algorithms and the distributions vary over input lengths). Moreover, it's esy to see that the relation $\approx_c$ is *transitive*, i.e. if $\mathcal{X} \approx_c \mathcal{Y}$ and $\mathcal{Y} \approx_c \mathcal{Z}$ then $\mathcal{X} \approx_c \mathcal{Z}$, and closed under *reductions*, i.e. for any PPT function f if $\mathcal{X} \approx_c \mathcal{Y}$ then $f(\mathcal{X}) \approx_c f(\mathcal{Y})$.

> ### Definition 3.3: Pseudorandom generator (PRG)
>
> Let $G : \{0,1\}^n \to \{0,1\}^{n+\ell}$ be a function where $\ell \geq 1$. We say that G is a pseudorandom generator (PRG) with expansion factor ℓ if:
>
> - G can be computed in $\mathrm{poly}(n)$-time
> - $G(U_n) \approx_c U_{n+\ell}$.

The above definition requires a bit of discussion in order to be fully understood. First, we observe that G is <u>deterministic</u>, making $G(U_n) \approx_c U_{n+\ell}$ a very strong property. Then, we observe that the idea behind a PRG is to generate new bits that look like they are truly random, as long as the initial seed is already truly random (in practice, we use

a seed that is computationally indistinguishable from a truly random one). Hence, we can use a randomness extractor to get an initial seed and then pass it to a PRG in order to "expand the seed" up to any desired length: after constructing a simple PRG $G : \{0,1\}^n \to \{0,1\}^{n+1}$, it can be used to construct a new PRG $G : \{0,1\}^n \to \{0,1\}^{n+\ell}$ up to any polynomial value $\ell > 0$.

To prove this result, we use a technique known as **hybrid argument**. We define a sequence $H_0, \dots, H_\ell$ of distributions such that:

- $H_0 \approx_c G^\ell(U_n)$ and $H_\ell \approx_c U_{n+\ell}$

- $H_i \approx_c H_{i+1}$ for each $i \in [0, \ell-1]$

- Each H_i is an hybrid of truly random and pseudorandom bits, where the number of truly random bits increases as i grows, getting replacing the pseudorandom bits

Theorem 3.1: PRG stretching

Let $G : \{0,1\}^n \to \{0,1\}^{n+1}$ be a PRG. Then, for any $\ell = \text{poly}(n)$ there is a PRG $G^\ell : \{0,1\}^n \to \{0,1\}^{n+\ell}$.

Proof. For each $i \in [0, \ell]$, let $f_i(s) = b_1 \ || \ \dots \ || \ b_\ell \ || \ s_\ell$ – where $||$ is the string concatenation operator – be the function computed through the following algorithm (see Figure 3.1).

$f_i =$ "On input $s \in \{0,1\}^n$:

1. Set $s_i = s$

2. For each $j \leq i$ set b_j as a truly random bit, i.e. $b_j \in_R U_1$

3. For each $j > i$ set s_j and b_j as the values such that $G(s_{j-1}) = s_j \ || \ b_j$, where $s_j \in \{0,1\}^n$ and $b_j \in \{0,1\}$.

4. Return $b_1 \ || \ \dots \ || \ b_\ell \ || \ s_\ell$.

It's easy to see that each $i \in [0, \ell]$ the function f_i is PPT-computable since G is a PPT algorithm by hypothesis and $\ell = \text{poly}(n)$. Moreover, f_0 is deterministic since it doesn't generate random bits. Let $H_0, \dots, H_\ell$ be the distributions over the outputs of the functions $f_0, \dots, f_\ell$. We observe that $H_\ell = U_{n+\ell}$ since every bit is truly random in f_ℓ. Hence, by letting $G^\ell = f_0$ we get that $H_0 = G^\ell(U_n)$ and $H_\ell = U_{n+\ell}$.

Claim: For each $i \in [0, \ell-1]$ it holds that $H_i \approx_c H_{i+1}$.

Proof of the claim. Fix $i \in [0, \ell-1]$. We prove the claim by reducing the distinguishability of $G(U_n)$ from U_{n+1} to the distinguishability of H_i from H_{i+1}. By way of contradiction, suppose that $H_i \not\approx_c H_{i+1}$, meaning that there is a distinguisher D_i such that:

$$\left| \Pr[D_i(z) = 1 : z \in_R H_i] - \Pr[D_i(z) = 1 : z \in_R H_{i+1}] \right| > \frac{1}{n^c}$$

for some $c > 0$. Let D be the algorithm defined as follows:

$D =$ "On input $w \in \{0,1\}^{n+1}$:

1. Set s and b as the values such that $w = s \,||\, b$, where $s \in \{0,1\}^n$ and $b \in \{0,1\}$.

2. Compute $f_{i+1}(s) = b_1 \,||\, \ldots \,||\, b_\ell \,||\, s_\ell$.

3. Set $z' = b_1 \,||\, \ldots \,||\, b_{i-1} \,||\, b \,||\, b_{i+1} \,||\, \ldots \,||\, b_\ell \,||\, s_\ell$.

4. Return $D_i(z_w)$."

We observe that H_i and H_{i+1} differ from each other only by one bit: in f_i, the bit b_i is the last bit of $G(s_{i-1})$, while in f_{i+1} it is a truly random bit. Hence, when D is given an input $w \in G(U_n)$, the last bit of w is generated through G, making the string z' built by D an input sampled from H_i. Vice versa, when D is given an input $w \in_R U_{n+1}$, the last bit of w is truly random, making the string z' built by D an in input sampled from H_{i+1}. Therefore, we get that:

$$
\begin{aligned}
&|\Pr[D(w) = 1 : w \in_R G(U_n)] - \Pr[D(w) = 1 : w \in_R U_{n+1}]| \\
&= |\Pr[D_i(z') = 1 : z' \in_R H_i] - \Pr[D_i(z') = 1 : z' \in_R H_{i+1}]| \\
&> \frac{1}{n^c}
\end{aligned}
$$

concluding that D is a distinguisher for G, raising a contradiction over G being a PRG. $\square$

By transitivy of $\approx_c$ and the above claim, we conclude that $G^\ell(U_n) = H_0 \approx_c \ldots \approx_c H_\ell = U_{n+\ell}$, meaning that G^ℓ is a PRG. $\square$
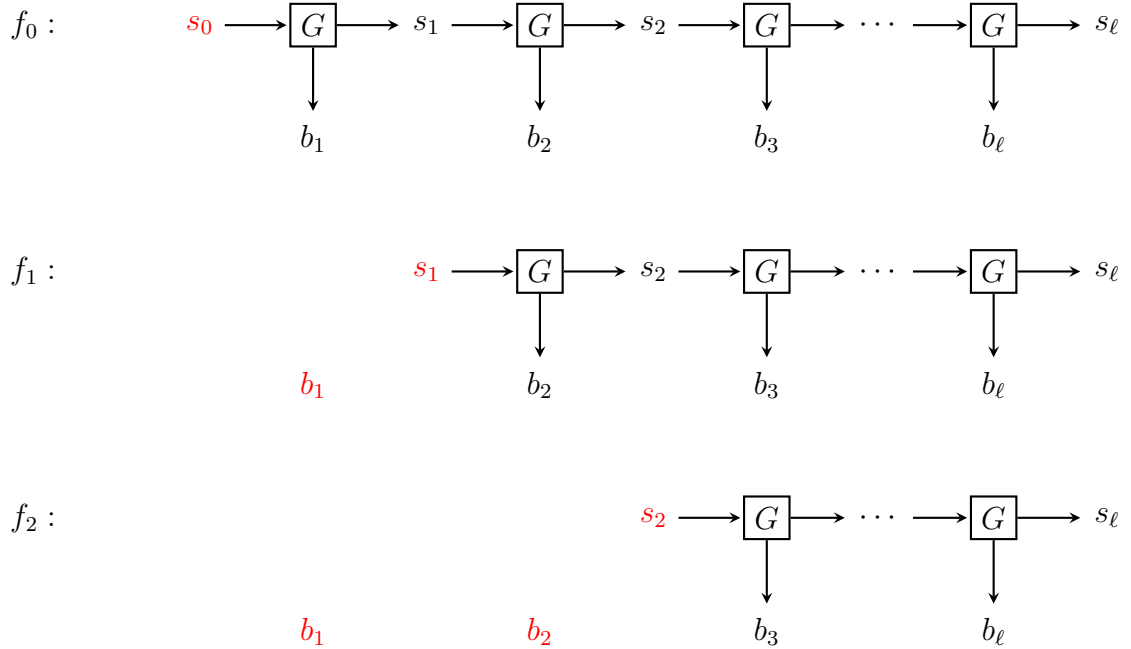

Figure 3.1: The idea behind the construction of the functions $f_0, \ldots, f_\ell$. Red strings represent truly random bits.

3.3 Hard-core predicates

In the previous chapter we saw how a PRG with stretch 1 is – in theory – sufficient to construct a PRG with stretch $\text{poly}(n)$. The next natural objective is to build such PRG with stretch 1. In theoretical terms, we'll see that such PRG can be constructed from any OWF. However, it can be proven that the existence of OWF and PRG is equivalent, i.e. that there exists an OWF if and only if there is a PRG. This theoretical result makes building PRG as hard as building a OWF, which we aren't even sure it exists. To bypass (only partially) this problem, in practical applications we build good-enough PRGs using heuristic constructions.

> **Theorem 3.2: From PRGs to OWFs**
>
> Let $G : \{0,1\}^n \rightarrow \{0,1\}^{2n}$ be a PRG. Then, there is a OWF $f : \{0,1\}^{2n} \rightarrow \{0,1\}^{2n}$.

Proof. Let f be the function that runs G on the first half of the input:

$$f(x_1 \,||\, \ldots \,||\, x_{2n}) = G(x_1 \,||\, \ldots \,||\, x_n)$$

By way of contradiction, suppose that f is not a OWF, meaning that there is an adversary A such that:

$$\Pr_{x \in_R U_n}[f(x') = y : y = f(x); x' \leftarrow A(y)] \geq \frac{1}{n^c}$$

for some $c > 0$. We build an adversary D defined as:

$D =$ "On input $z \in \{0,1\}^{2n}$:

1. Compute $A(z) = b_1 \,||\, \ldots \,||\, b_{2n}$

2. If $G(b_1 \,||\, \ldots \,||\, b_n) = z$ accept, otherwise reject."

If $z \in_R U_{2n}$, the adversary D accepts with probability at most 2^{-n} since the first half of the output $A(z)$ is also uniform at random. If $z \in_R G(U_n)$, instead, the adversary D accepts with probability at least n^{-c} since it first inverts the output and then recomputes it to distinguish it. Hence, we conclude that:

$$|\Pr[D(z) = 1 : z \in_R G(U_n)] - \Pr[D(z) = 1 : z \in_R U_{2n}]| \geq \frac{1}{n^c} - \frac{1}{2^n} \geq \frac{1}{n^c}$$

contradicting the assumption of G being a PRG. $\qquad\square$

To prove that the implication also holds from right to left, thus that OWF imply PRGs, we need a fundamental theoretical concept called **hard-code bits**. Given a function f, these bits contain information about an input x that is hard to obtain from the output $f(x)$, meaning that any adversary A cannot do better than guessing with 50% chance – up to some negligible error – if the bit comes from x or not. Hard-core bits are given by a function h – called *predicate* – that extract an hard-core bit from any input x.

> ### Definition 3.4: Hard-core predicate
>
> Let $f : \{0,1\}^n \to \{0,1\}^m$ be a PPT-computable function. We say that a function $h : \{0,1\}^n \to \{0,1\}$ is a hard-core predicate for f if for every PPT adversary A it holds that:
> $$\Pr[A(y) = h(x) : x \in_R U_n, y = f(x)] \leq \frac{1}{2} + \mathrm{negl}(n)$$

From the very definition that we just gave, it's easy to get an equivalent formulation: h is hard-core for f if and only if $(f(x), h(x)) \approx_c (f(x), U_1)$ with $x \in_R U_n$ (proof is left as an exercise).

We observe that both definitions we gave don't require that f is a OWF. However, hard-core predicates are – intuitively – what allows a function to be one-way: if we can establish an hard-core bit for every input x, any adversary will have a hard time inverting $f(x)$ due to this very bit. From these observation, we ask three questions:

1. Is there a universal hard-core predicate that is hard-core for every OWF?

2. Is every function with an hard-core predicate also one-way?

3. Does every one-way function have an hard-core predicate?

For the first question, the answer is *no*. Suppose that such h^* exists. Fix a OWF f and let $\widehat{f}$ be defined $\widehat{f}(x) = h^*(x) \;||\; f(x)$. It's easy to see that the adversary A^* that always returns the first bit of the output $\widehat{f}(x)$ is such that:

$$\Pr[A^*(y) = h^*(x) : x \in_R U_n, y = \widehat{f}(x)] = 1$$

concluding that h^* is not an hard-core predicate for $\widehat{f}$. By way of contradiction, suppose that $\widehat{f}$ is not a OWF, i.e. there is an adversary $\widehat{A}$ such that:

$$\Pr_{x \in_R U_n}[\widehat{f}(x') = y : y = \widehat{f}(x); x' \leftarrow \widehat{A}(y)] \geq \frac{1}{n^c}$$

for some $c > 0$. We build an adversary A such that $A(y) = \widehat{A}(b \;||\; y)$, where $b \in_R U_1$. By construction, we have that:

$$
\begin{aligned}
&\Pr_{x \in_R U_n}[f(x') = y : y = f(x); x' \leftarrow A(y)] \\
&\geq \Pr_{x \in_R U_n}[b = h^*(x), f(x') = y : y = f(x); x' \leftarrow \widehat{A}(b \;||\; y); b \in_R U_1] \\
&= \frac{1}{2} \Pr_{x \in_R U_n}[f(x') = y : y = f(x); x' \leftarrow \widehat{A}(h^*(x) \;||\; y)] \\
&= \frac{1}{2} \Pr_{x \in_R U_n}[\widehat{f}(x') = h^*(x') \;||\; y : y = f(x); x' \leftarrow \widehat{A}(h^*(x) \;||\; y)] \\
&= \frac{1}{2n^c}
\end{aligned}
$$

which contradicts the assumption of f being OWF. Hence, $\widehat{f}$ must be an OWF without h^* as an hard-predicate.

For the second question, the answer is also *no*. Let f be the function such that drops the last bit of the input, i.e. $f(x_1 \mathbin{||} \ldots \mathbin{||} x_n) = x_1 \mathbin{||} \ldots \mathbin{||} x_{n-1}$. Let h be the function that returns only the last bit of the input, i.e. $h(x_1 \mathbin{||} \ldots \mathbin{||} x_n) = x_n$. By construction, h is trivially an hard-core predicate for f since the last bit is independent from the others when $x \in_R U_n$. However, f is not a OWF since an adversary that picks a random bit $b \in_R U_1$ has 50% chance of guessing $b = x_n$, inverting the output.

What about the third question? In this case, the answer is *yes, kinda*. Goldreich and Levin showed that every OWF can be trivially modified to obtain a new OWF that has a specific hard-core predicate. For any OWF $f : \{0,1\}^n \to \{0,1\}^m$, the **Goldreich-Levin alteration** of f is the function $g : \{0,1\}^{2n} \to \{0,1\}^{m+n}$ defined as $g(x,r) = f(x) \mathbin{||} r$, where r is supposed to be taken as $r \in_R U_n$. We omit the proof due to it being too convoluted.

> **Theorem 3.3: The Goldreich-Levin theorem**
>
> Let f be an OWF and let g be obtained through the Goldreich-Levin construction. Then, g is an OWF function with an hard-core predicate $h(x,r) = \bigoplus_{i=1}^{n} x_i r_i$.

Proof. Omitted. $\qquad\square$

Surprisingly, OWFs can be combined with an hard-core predicate in order to generate a PRG. The easiest case of such property is given by **One-way Permutations (OWPs)**, that being bijective OWFs.

> **Proposition 3.1: From OWP to PRG**
>
> Let f be a OWP with an hard-core predicate h. Then, $G(s) = f(s) \mathbin{||} h(s)$ is a PRG with expansion factor 1.

Proof. Consider the first n bits of the output $G(s)$, i.e. the bits $f(s)$. When s is taken uniformly at random, the output $f(s)$ will also be uniform at random due to f being a permutation. For the last $n+1$ bit of the output, instead, we know that $h(s)$ is by definition computationally hard to distinguish from a bit $b \in_R U_1$. This concludes that:

$$G(U_n) \equiv (f(U_n), h(U_n)) \approx_c (U_n, U_1) \equiv U_{n+1}$$

$\square$

The above proposition can be strenghtened by generalizing to every OWF instead of OWPs (the idea is to mix multiple outputs of f with their hard-core bits).

> **Theorem 3.4: From OWF to PRG**
>
> For every OWF f there is a PRG G with expansion factor 1.

Proof. Omitted. $\qquad\square$

3.4 Solved exercises

> **Problem 3.1**
>
> For $m > n$, let $G : \{0,1\}^n \to \{0,1\}^m$ be a secure PRG. Consider G' defined as $G'(s) = G(s) \oplus (0^{m-n} \,||\, s)$. Prove or disprove that G' is a secure PRG.

Solution:

G' is not necessarily a secure PRG. Let $G(s) = G^*(s_{1:n-1}) \,||\, s_n$, where $G^* : \{0,1\}^n \to \{0,1\}^{m-1}$ is a PRG, $s_{1:n-1}$ denotes the first $n-1$ bits of s and s_n is the last bit of s. It can be proven that this chosen G is indeed a PRG, making it a valid choice. Now, we observe that:

$$
\begin{aligned}
G'(s) &= G(s) \oplus (0^{m-n} \,||\, s) \\
&= (G^*(s_{1:n-1})_{1:m-n} \oplus 0^{m-n}) \,||\, (G^*(s_{1:n-1})_{m-n+1:m-1} \oplus s_{1:n-1}) \,||\, (s_n \oplus s_n) \\
&= G^*(s_{1:n-1})_{1:m-n} \,||\, (G^*(s_{1:n-1})_{m-n+1:m-1} \oplus s_{1:n-1}) \,||\, 0
\end{aligned}
$$

In other words, the last bit of $G'(s)$ will always be 0. Consider now the adversary A that checks the input $z \in \{0,1\}^m$ and returns 1 if $z_m = 0$ and 0 otherwise. Then:

$$
|\Pr[A(z) = 1 : z \in_R G'(U_n)] - \Pr[A(z) = 1 : z \in_R U_m]| = \left| 1 - \frac{1}{2} \right| = \frac{1}{2}
$$

concluding that A is an adversary that distinguishes $G'(U_n)$ from U_m with non-negligible probability.

4

Symmetric-key encryption schemes

4.1   Games as security proofs

In Chapter 2 we gave the initial definitions of secure cryptographic schemes, without caring about the computation being achieved efficiently. In Chapter 3, instead, we discussed the basics of secure efficient computation. Now, we're ready to merge the two ideas.

We start by discussing secure SKEs. Using the concept of pseudorandom generator (PRG) that we introduced in the previous chapter, we can define a *PRG One Time Pad (PRG-OTP)*. Given $\mathcal{K} = \{0,1\}^\lambda$ and $\mathcal{M} = \mathcal{C} = \{0,1\}^{\lambda+\ell}$, we define:

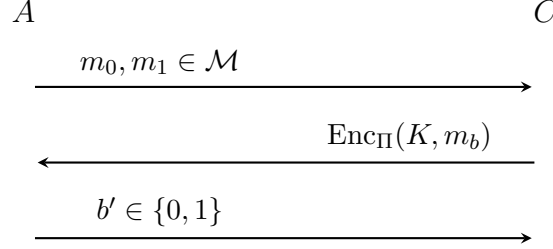$$\text{Enc}(K,m) = G(K) \oplus m \qquad \text{Dec}(K,c) = G(K) \oplus c$$

where $G : \{0,1\}^\lambda \rightarrow \{0,1\}^{\lambda+\ell}$ is a PRG and $\ell = \text{poly}(\lambda)$. The use of a PRG allows us to "bypass" Shannon's key-length constraint since the "pseudorandom key" generated through $G(K)$ is still at least as long as the message. However, this implies that our SKE doesn't have real perfect secrecy, otherwise the theorem would be false. This idea moves us into defining a new concept of **computationally secure SKE**.

In this and many other cases, security definitions will be given in terms of **games** between two entities: a challenger and an adversary. The two players play the game sending messages to each other. The challger's objective is proving that the scheme is secure by answering the questions posed the adversary, while the latter's objective is to break the scheme. The challenger wins if it is capable of responding to any possible series of questions without making the adversary able to break the scheme. If the challenger has a strategy to win the game, we consider the scheme to be secure. We assume that the two players are allowed only efficient computation.

Consider the following example. Let $\text{Game}_{\Pi,A}^{\text{1-time}}(\lambda,b)$ be the 2-player game defined as follows:

1. The adversary A sends two messages $m_0, m_1 \in \mathcal{M}$ to the challenger C

2. C generates a key $K \in \mathcal{K}$ and sends $\text{Enc}_\Pi(K, m_b) = c$ back to A

3. A analyzes c and tries to guess if it was obtained through m_0 or m_1 by sending a bit $b' \in \{0, 1\}$

4. If $b' \neq b$, the challenger wins. Otherwise, the adversary wins.

$$A \qquad\qquad\qquad\qquad\qquad\qquad C$$

$$\xrightarrow{\quad m_0, m_1 \in \mathcal{M} \quad}$$

$$\xleftarrow{\quad \text{Enc}_\Pi(K, m_b) \quad}$$

$$\xrightarrow{\quad b' \in \{0, 1\} \quad}$$

Figure 4.1: Graphical representation of $\text{Game}_{\Pi,A}^{\text{1-time}}(\lambda, b)$

By the very definition of the game, it's easy to see that if, for every pair of messages, the adversary is not capable of distinguishing which message got encrypted by the challenger, i.e. if they are playing $\text{Game}_{\Pi,A}^{\text{1-time}}(\lambda, 0)$ or $\text{Game}_{\Pi,A}^{\text{1-time}}(\lambda, 1)$, then the scheme can be considered **1-time computationally secure SKE**.

> **Definition 4.1: 1-time computational security**
>
> Let $\Pi = (\text{Enc}, \text{Dec})$ be a SKE. We say that Π is 1-time computationally secure if:
>
> $$\text{Game}_{\Pi,A}^{\text{1-time}}(\lambda, 0) \approx_c \text{Game}_{\Pi,A}^{\text{1-time}}(\lambda, 1)$$
>
> for all PPT adversaries A.

To see how this game can be used for specific cryptosystems, we prove that PRG-OTP has 1-time computationally security.

> **Proposition 4.1**
>
> PRG-OTP has 1-time computational security.

Proof. Let $\ell = \text{poly}(n)$ be the expansion factor of the PRG G used inside PRG-OTP. We use an argument similar to the hybrid argument. Let $\text{Hyb}(\lambda, b)$ be the 2-player game defined as follows:

1. The adversary A sends two messages $m_0, m_1 \in \{0, 1\}^\lambda$ to the challenger C

2. C picks $z \in_R U_{\lambda+\ell}$ and sends $z \oplus m_b = c$ back to A

3. A analyzes c and tries to guess if it was obtained through m_0 or m_1 by sending a bit $b' \in \{0, 1\}$

4. If $b' \neq b$, the challenger wins. Otherwise, the adversary wins.

We observe that the difference between $\mathrm{Game}_{\Pi,A}^{\text{1-time}}(\lambda, b)$ and $\mathrm{Hyb}(\lambda, b)$ is given by $G(K)$ being replaced by a uniform at random string $z \in_R U_{\lambda+\ell}$. Since $G(U_\lambda) \approx_c U_{\lambda+\ell}$ by choice of G, the hybrid game and the original game are expected to also be indistinguishable from each other.

Claim: For each $b \in \{0, 1\}$ it holds that $\mathrm{Game}_{\Pi,A}^{\text{1-time}}(\lambda, b) \approx_c \mathrm{Hyb}(\lambda, b)$

Proof of the claim. By way of contradiction, assume that there is an adversary A such that:

$$\left| \Pr\left[A(b') = 1 : b' \leftarrow \mathrm{Game}_{\Pi,A}^{\text{1-time}}(\lambda, b) \right] - \Pr\left[A(b') = 1 : b' \leftarrow \mathrm{Hyb}(\lambda, b) \right] \right| \geq \frac{1}{\lambda^c}$$

for some $c > 0$. We build an adversary D defined as follows:

$D = $ "On input $z \in \{0, 1\}^{\lambda+\ell}$.

1. D challenges A to play $\mathrm{Hyb}(\lambda, b)$ (thus D is the challenger and A is the adversary). D will use z to compute $m_b \oplus z = c$ instead of choosing a random string.

2. D accepts if A wins, otherwise D rejects."

Since A can distinguish the two games, it is also able to distinguish if the string c passed by D was computed with z taken from $G(U_\lambda)$ or $U_{\lambda+\ell}$. This implies that:

$$\Pr[D(z) = 1 : z \in_R G(U_\lambda)] = \Pr\left[A(b') = 1 : b' \leftarrow \mathrm{Game}_{\Pi,A}^{\text{1-time}}(\lambda, b) \right]$$

and:

$$\Pr[D(z) = 1 : z \in_R U_{\lambda+\ell}] = \Pr\left[A(b') = 1 : b' \leftarrow \mathrm{Hyb}(\lambda, b) \right]$$

concluding that:

$$\begin{aligned}
&\left| \Pr[D(z) = 1 : z \in_R G(U_\lambda)] - \Pr[D(z) = 1 : z \in_R U_{\lambda+\ell}] \right| \\
&= \left| \Pr\left[A(b') = 1 : b' \leftarrow \mathrm{Game}_{\Pi,A}^{\text{1-time}}(\lambda, b) \right] - \Pr\left[A(b') = 1 : b' \leftarrow \mathrm{Hyb}(\lambda, b) \right] \right| \\
&\geq \frac{1}{\lambda^c}
\end{aligned}$$

and thus contradicting the fact that G is a PRG. $\square$

By definition, it's easy to see that $\mathrm{Hyb}(\lambda, 0) \equiv \mathrm{Hyb}(\lambda, 1)$ since the distribution of c is uniform and independent of b. Therefore, we conclude that:

$$\mathrm{Game}_{\Pi,A}^{\text{1-time}}(\lambda, 0) \approx_c \mathrm{Hyb}(\lambda, 0) \equiv \mathrm{Hyb}(\lambda, 1) \approx_c \mathrm{Game}_{\Pi,A}^{\text{1-time}}(\lambda, 1)$$

$\square$

4.2   CPA security and pseudorandom functions

In the previous section we proved that PRG-OTP is 1-time computationally secure, meaning that it is secure against attackers that know only one message. However, this SKE's weakness is *ciphertext attacks*. Suppose that an attacker knows a ciphertext-message pair (m_1, c_1). If the challenger sends another ciphertext c_2 obtained from a message m_2, the attacker is able to retrieve the contents of m_2 through the SKE itself:

$$c_1 \oplus c_2 \oplus m_1 = (G(K) \oplus m_1) \oplus (G(K) \oplus m_2) \oplus m_1 = m_2$$

The generalization of this type of attack is known as **Known-plaintext Attack (KPA)** and the security counterpart is known as *t-time computational security*. We observe that KPAs require that the attacker only knows the pairs $(m_1, c_1), \ldots, (m_t, c_t)$ but is not able to choose them. When the attacker is able to choose the pairs (e.g. they can choose messages and get their ciphertexts) we talk about **Chosen-plaintext Attack (CPA)**.

In the CPA game the adversary is given access to an encryption **oracle**, i.e. a sub-procedures with unlimited power that can be queried in order to get an answer. Each query costs $\Theta(1)$ computational steps. We observe that the oracle queries made by the attacker can be seen as "questions already answered". For instance, suppose that the attacker queries an oracle O capable of returning the ciphertext of a message, i.e. solves the problem $\text{Enc}(K, \cdot)$. If the attacker queries $m_1, \ldots, m_t$, the oracle will answer with $c_1, \ldots, c_t$. This is equivalent to saying that the attacker has chosen the pairs $(m_1, c_1), \ldots, (m_t, c_t)$. The concept of oracle is just a computational medium to express this idea.

Let $\text{Game}_{\Pi,A}^{\text{CPA}}(\lambda, b)$ be the 2-player game defined as follows:

1. The challenger C generates a key $K \in \mathcal{K}$.

2. The adversary A queries $m_1, \ldots, m_t \in \{0, 1\}^\lambda$, where $t = \text{poly}(\lambda)$, to an oracle for $\text{Enc}_\Pi(K, \cdot)$. The oracle answers with $c_1, \ldots, c_t$.

3. The adversary sends to messages $m_0^*, m_1^* \in \{0, 1\}^\lambda$ to the challenger C

4. C sends $\text{Enc}_\Pi(K, m_b^*) = c^*$ back to A

5. A queries $m_1', \ldots, m_s' \in \{0, 1\}^\lambda$, where $s = \text{poly}(\lambda)$, to an oracle for $\text{Enc}_\Pi(K, \cdot)$. The oracle answers with $c_1', \ldots, c_t'$.

6. A tries to guess if it was obtained through m_0 or m_1 by sending a bit $b' \in \{0, 1\}$

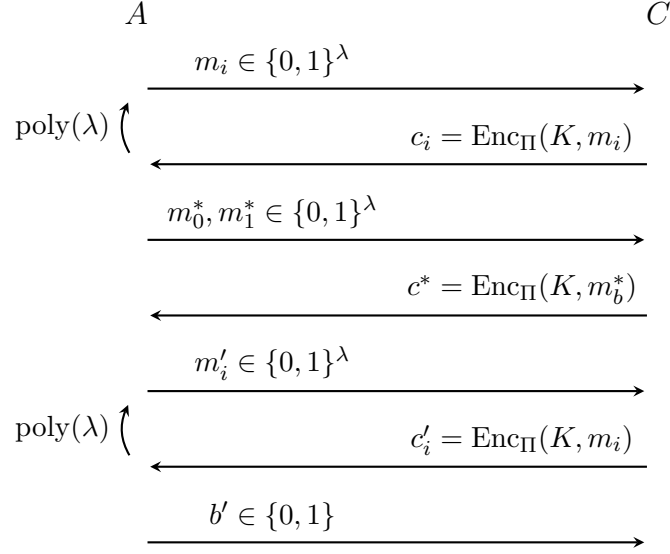7. If $b' \neq b$, the challenger wins. Otherwise, the adversary wins.

Figure 4.2: Graphical representation of $\mathrm{Game}_{\Pi,A}^{\mathrm{CPA}}(\lambda, b)$. Queries to the oracle can be substituted with queries to C, as if the two players already exchanged some pairs.

> ### Definition 4.2: CPA-security
>
> Let $\Pi = (\mathrm{Enc}, \mathrm{Dec})$ be a SKE. We say that Π is CPA-secure if:
>
> $$\mathrm{Game}_{\Pi,A}^{\mathrm{CPA}}(\lambda, 0) \approx_c \mathrm{Game}_{\Pi,A}^{\mathrm{CPA}}(\lambda, 1)$$
>
> for all PPT adversaries A.

It's easy to see that the PRG-OTP is not CPA-secure since the output of the encryption scheme is *deterministic* (meaning that it will always return the same output when given the same input). Consider the attacker that queries two messages $m_0, m_1 \in \{0, 1\}^n$. After obtaining the ciphertexts c_0, c_1 from C, the attacker sends the final queries $m_0^* = m_0$ and $m_1^* = m_1$, obtaining the final ciphertext c^*. Since the scheme is deterministic, we know that either $c^* = c_0$ or $c^* = c_1$ must hold (depending on the value b used by C). Thus, the attacker returns 0 if $c^* = c_{0,,}$ otherwise they returns 1. This allows the attacker to distinguish the two games with probability 1. Clearly, this idea is not restricted to the PRG-OTP: it holds for any deterministic SKE!

> ### Proposition 4.2
>
> No deterministic SKE can be CPA-secure.

In order to make a CPA-secure SKE, we need to substitute pseudorandom generators with **pseudorandom functions**, i.e. functions that are indistinguishable from a randomly selected one (thus they make their internal workings incomprehensible).

Let $\mathcal{R}(s \to t)$ denote the uniform distribution over the set of all functions from strings of s bits to strings of t bits. We define the following 2-player game $\mathrm{Game}_{\mathcal{F},A}^{\mathrm{PRF}}(\lambda, b)$:

1. The challenger C generates a key $K \in \{0,1\}^\lambda$ and sets $H = F_K$ if $b = 0$, otherwise $H \in_R \mathcal{R}(n \to n)$.

2. The adversary A queries $x_1, \ldots, x_t \in \{0,1\}^n$ where $t = \text{poly}(n)$, to an oracle for $H(\cdot)$. The oracle answers with $y_1, \ldots, y_t$.

3. A tries to guess b by sending a bit $b' \in \{0,1\}$

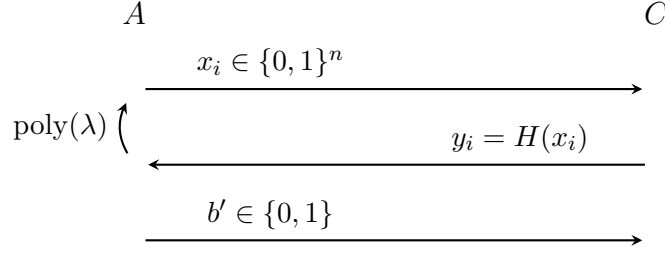4. If $b \neq b'$, the challenger wins. Otherwise, the adversary wins.

$$A \hspace{8cm} C$$

$$\xrightarrow{\quad x_i \in \{0,1\}^n \quad}$$

$$\text{poly}(\lambda) \Big\uparrow \quad \xleftarrow{\quad y_i = H(x_i) \quad}$$

$$\xrightarrow{\quad b' \in \{0,1\} \quad}$$

Figure 4.3: Graphical representation of $\text{Game}_{\mathcal{F},A}^{\text{PRF}}(\lambda, b)$. Queries to the oracle can be substituted with queries to C, as if the two players already exchanged some pairs.

> **Definition 4.3: Pseudorandom functions**
>
> Let $\mathcal{F} = \{F_K : \{0,1\}^n \to \{0,1\}^m\}_{K \in \{0,1\}^\lambda}$ be a family of poly-time functions. We say that $\mathcal{F}$ is a pseudorandom function family (PRF) if:
>
> $$\text{Game}_{\mathcal{F},A}^{\text{PRF}}(\lambda, 0) \approx_c \text{Game}_{\mathcal{F},A}^{\text{PRF}}(\lambda, 1)$$
>
> for all PPT adversaries A.

We observe that the above definition requires that $\mathcal{F}$ is a family of efficient <u>deterministic</u> functions: randomness is only used to choose a key that identifies which function of the family must be used. The uniform distribution $\mathcal{R}(n \to n)$, instead, also contains functions that aren't poly-time computable. The idea behind the PRF game is to establish that each function of the family is computationally indistinguishable from a randomly chosen function.

In practical applications, many common encryption schemes (e.g. DES, 3DES, AES, ...) are based on on *pseudorandom permutations (PRP)*, i.e. PRFs that are invertible if they key is known. For now, we'll give the idea behind how PRFs can be used in theory to produce CPA-secure SKEs.

First of all, we observe that we cannot simply swap PRGs with PRFs in order to get a good scheme since each function is deterministic. The trick here is to use a random initial value called **nonce** (short for "number used once"). As their name suggests, these values are used exactly once by the cryptosystem, granting that the same input will always give a different ciphertext.

Given a PRF $\mathcal{F} = \{F_K : \{0,1\}^n \to \{0,1\}^n\}_{K \in \{0,1\}^\lambda}$, let the *PRF One Time Pad (PRF-OTP)* be the scheme defined by the following operations:

1. $\text{Enc}(K, m) = (r, F_K(r) \oplus m)$, where $r \in \{0, 1\}^n$ is a nonce

2. $\text{Dec}(K, (r, c)) = F_K(r) \oplus c$

We observe that the encryption operation must also return the used nonce in order for the decryption operation to work. However, this is not an issue for CPA-security: since the nonce will never be used again by the scheme, the adversary gains no information on the key by knowing it.

Proposition 4.3

PRF-OTP is CPA-secure.

Proof. Omitted (an hybrid argument suffices). □

4.2.1 The GGM Tree

After discussing how PRFs can be used to achieve CPA-security, we prove that PRGs can be used to construct PRFs. In particular, we'll give an explicit construction known as the **Goldreich-Goldwasser-Micali Tree (GGM Tree)**.

Definition 4.4: GGM Tree

Given a PRG $G : \{0, 1\}^\lambda \to \{0, 1\}^{2\lambda}$, let $G_0, G_1 : \{0, 1\}^\lambda \to \{0, 1\}^\lambda$ be two functions that respectively return the first and last λ bits of the output of G, meaning that $G(k) = G_0(k) \,||\, G_1(k)$ for all $k \in \{0, 1\}^\lambda$. The GGM Tree function family is defined as $\mathcal{F} = \{F_K : \{0, 1\}^n \to \{0, 1\}^\lambda\}_{K \in \{0,1\}^\lambda}$, where:

$$F_K(x_1 \,||\, \ldots \,||\, x_n) = G_{x_n}(G_{x_{n-1}}(\ldots G_{x_2}(G_{x_1}(K))))$$
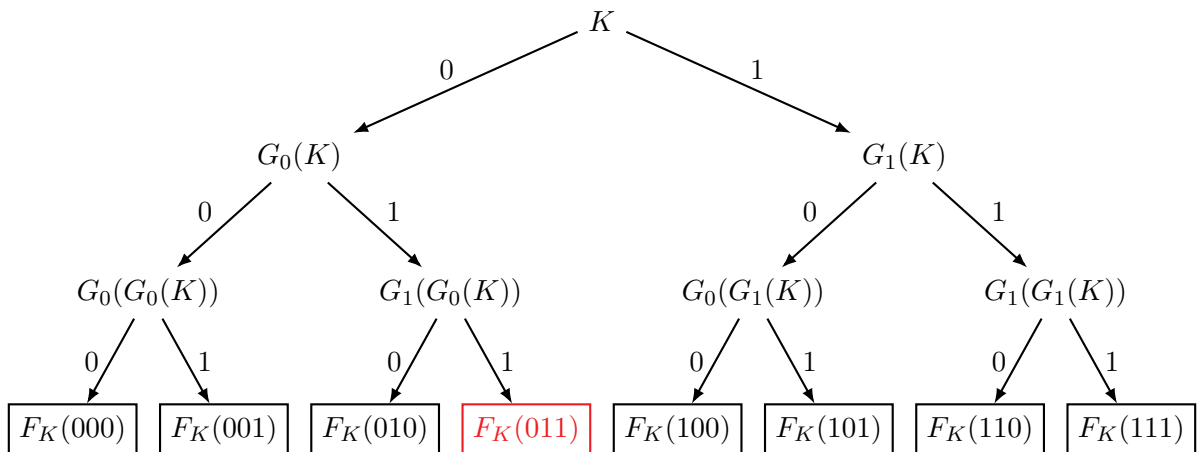

Figure 4.4: The GGM Tree for inputs of length $n = 3$. The path leading to the red rectangle represent the computation $F_K(011) = G_1(G_1(G_0(K)))$.

Lemma 4.1

Let $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ be a PRG. Given $K_1, \ldots, K_t \in_R U_\lambda$, with $t = \text{poly}(\lambda)$ and $K_i \neq K_j$ for all $i \neq j$, it holds that:

$$(G(K_1), \ldots, G(K_t)) \approx_c (U_{2\lambda}, \ldots, U_{2\lambda})$$

Proof. We define t hybrid distributions $H_0, \ldots, H_t$ such that for each $i \in [t]$ it holds that:

$$H_i = (G(K_1), \ldots, G(K_{t-i}), \underbrace{U_{2\lambda}, \ldots, U_{2\lambda}}_{i \text{ times}})$$

We observe that $H_0 \equiv (G(K_1), \ldots, G(K_t))$ and $H_t \equiv (U_{2\lambda}, \ldots, U_{2\lambda})$.

Claim: For each $i \in [0, t-1]$ it holds that $H_i \approx_c H_{i+1}$.

Proof of the claim. Fix $i \in [0, t-1]$. We prove the claim by reducing the distinguishability of $G(K_{t-i})$ from $U_{2\lambda}$ to the distinguishability of H_i from H_{i+1}. By way of contradiction, suppose that $H_i \not\approx_c H_{i+1}$, meaning that there is a distinguisher D_i such that:

$$|\Pr[D_i(z) = 1 : z \in_R H_i] - \Pr[D_i(z) = 1 : z \in_R H_{i+1}]| > \frac{1}{n^c}$$

for some $c > 0$. Let D be the algorithm defined as follows:

$D = $ "On input $z \in \{0,1\}^{2\lambda}$:

1. Extract $t - i$ keys $K_1, \ldots, K_{t-i} \in_R U_\lambda$

2. Extract $i - 1$ strings $w_1, \ldots, w_{i-1} \in_R U_{2\lambda}$

3. Return $D_i(G(K_1) \mid\mid \ldots \mid\mid G(K_{t-i}) \mid\mid z \mid\mid w_1 \mid\mid \ldots \mid\mid w_{i-1})$"

We observe that H_i and H_{i+1} differ only on the i-th string of $2n$ bits: the former contains an output of G while the latter contains a truly random strng. This implies that:

$$\begin{aligned}
&|\Pr[D(w) = 1 : w \in_R G(K_{t-i})] - \Pr[D(w) = 1 : w \in_R U_{2\lambda}]| \\
&= |\Pr[D_i(z') = 1 : z' \in_R H_i] - \Pr[D_i(z') = 1 : z' \in_R H_{i+1}]| \\
&> \frac{1}{n^c}
\end{aligned}$$

concluding that D is a distinguisher for G, raising a contradiction over G being a PRG. $\quad\square$

By transitivity of $\approx_c$, the claim concludes that:

$$(G(K_1), \ldots, G(K_t)) \equiv H_0 \approx_c \ldots \approx_c H_t \equiv (U_{2\lambda}, \ldots, U_{2\lambda})$$

$\square$

Theorem 4.1: The GGM theorem

GGM Tree is a PRF when constructed using a PRG

Proof. Let $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ be a PRG and let $G_0, G_1 : \{0,1\}^\lambda \to \{0,1\}^\lambda$ be the two functions such that $G(x) = G_0(x) \mathbin{||} G_1(x)$. Let also $\mathcal{F} = \{F_K : \{0,1\}^n \to \{0,1\}^\lambda\}_{K \in \{0,1\}^\lambda}$ be the GGM function family.

Fix a key $K \in \{0,1\}^\lambda$. We proceed by induction on the length n of the input. When $n = 1$ it holds that $F_K(x) = G_x(K)$. Since $U_{2\lambda} \approx_c G(K) \equiv (G_0(K), G_1(K))$, we know that $G_0(K) \approx_c U_\lambda$ and $G_1(K) \approx_c U_\lambda$. This concludes that, independently from the value of x, the output $F_K(x)$ is undistinguishable from a string taken from U_λ, making it undistinguishable from a random function taken from $\mathcal{R}(1 \to \lambda)$.

We now assume that the inductive hypothesis holds for $n - 1$, i.e. that the GGM family $\mathcal{F} = \{F_K : \{0,1\}^{n-1} \to \{0,1\}^\lambda\}_{K \in \{0,1\}^\lambda}$ is a PRF. First, we observe that for each key $K \in \{0,1\}^\lambda$, for each $x \in \{0,1\}^n$ and for each $y \in \{0,1\}^{n-1}$ it holds that $F_K(x \mathbin{||} y) = G_x(F'_K(y))$.

We define the following three hybrid distributions:

- Hyb_0 is the distribution over the function $F_K(x \mathbin{||} y) = G_x(F'_K(y))$

- Hyb_1 is the distribution over the function $H(x \mathbin{||} y) = G_x(R'(y))$ where $R' \in_R \mathcal{R}(n-1 \to \lambda)$.

- Hyb_2 is the distribution over the function $R \in_R \mathcal{R}(n \to \lambda)$.

To prove that $\mathcal{F}$ is also a PRF, we'll show that $\mathrm{Hyb}_0 \approx_c \mathrm{Hyb}_1 \approx_c \mathrm{Hyb}_2$.

Claim 1: $\mathrm{Hyb}_0 \approx_c \mathrm{Hyb}_1$

Proof of Claim 1. We reduce the distinguishability of $\mathrm{Game}_{\mathcal{F}',A}^{\mathrm{PRF}}(\lambda, 0)$ from $\mathrm{Game}_{\mathcal{F}',A}^{\mathrm{PRF}}(\lambda, 1)$ to the distinguishability of Hyb_0 from Hyb_1. By way of contradiction, suppose that $\mathrm{Hyb}_0 \not\approx_c \mathrm{Hyb}_1$, meaning that there is a PPT adversary D_{01} such that:

$$|\Pr[D_{01}(z) = 1 : z \in_R \mathrm{Hyb}_0] - \Pr[D_{01}(z) = 1 : z \in_R \mathrm{Hyb}_1]| > \frac{1}{n^c}$$

for some $c > 0$. We define the attack strategy A_{01} for $\mathrm{Game}_{\mathcal{F}',A}^{\mathrm{PRF}}(\lambda, b)$ as follows. The idea is for A_{01} to play two games simultaneously.

1. The challenger C generates a key $K \in \{0,1\}^\lambda$ and sets $H = F'_K$ if $b = 0$, otherwise $H \in_R \mathcal{R}(n-1 \to \lambda)$.

2. A_{01} challenges D_{01} to distinguish Hyb_0 from Hyb_1.

3. For each query $x \mathbin{||} y$ – with $x \in \{0,1\}$ and $y \in \{0,1\}^n$ – made by D_{01}, the adversary A_{01} forwards y to the challenger C.

4. For each answer z returned by C, the adversary A_{01} forwards $G_x(z)$ to D_{01}.

5. When D_{01} gives the final bit b', A_{01} forwards the same bit to C.

In the above strategy, A_{01} plays two games simultaneously, using the challenger as a way to compute outputs of either F'_K or a random function R. After obtaining the outputs, A_{01} passes them through G_x before forwarding them to D_{01}. If the challenger used F'_K, the values forwarded from A_{01} will be outputs of F_K. Otherwise, if the challenger used R',

the values forwarded from A_{01} will be outputs of R. Since D_{01} can distinguish between F_K and R, A_{01} will be able to distinguish F'_K from R'. This contradicts the fact that $\mathcal{F}'$ is a PRF, concluding that $\mathrm{Hyb}_0 \approx_c \mathrm{Hyb}_1$ must be true. $\qquad\square$

Claim 2: $\mathrm{Hyb}_1 \approx \mathrm{Hyb}_2$

Proof of Claim 2. We reduce the distinguishability of $(G(K_1), \ldots, G(K_t))$ from $(U_{2\lambda}, \ldots, U_{2\lambda})$ to the distinguishability of Hyb_1 from Hyb_2. By way of contradiction, suppose that there is an algorithm D_{12} that distinguishes Hyb_1 from Hyb_2 For each string $w \in \{0,1\}^{2\lambda}$, let $w^0, w^1 \in \{0,1\}^\lambda$ denote the first and last λ bits of w, i.e. $w = w^0 \ || \ w^1$. Let A_{12} be the algorithm defined as follows:

$A_{12} = $ "On input $z = z_1 \ || \ \ldots \ || \ z_t$ with $z_i \in \{0,1\}^{2\lambda}$:

1. A_{12} challenges D_{12} to distinguish Hyb_1 from Hyb_2, allowing D_{12} exactly t queries.

2. For each query $x \ || \ y$ – with $x \in \{0,1\}$ and $y \in \{0,1\}^n$ – made by D_{12}, the adversary A_{12} answers with $z^x_{i(y)}$, where $i(y) \in [1,t]$ is either the index that was used by A_{12} if D_{12} already queried y or the next unused index.

3. When D_{01} gives the final bit b', A_{01} returns b'

When $z_1, \ldots, z_t \in U_{2\lambda}$, all values returned by A_{12} to D_{12} will be truly random, making D_{12} assume that the values are taken from the distribution Hyb_2. When $z_1, \ldots, z_t \in G(U_\lambda)$, instead, all values will be outputs of G, making D_{12} assume that the values are taken from the distribution Hyb_1. This concludes that A_{12} is a distinguisher for $(G(K_1), \ldots, G(K_t))$ and $(U_{2\lambda}, \ldots, U_{2\lambda})$, contradicting Lemma 4.1. $\qquad\square$
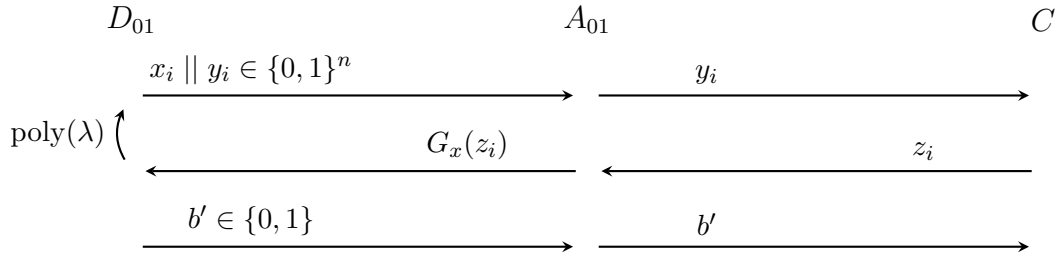
$\hfill\square$

Figure 4.5: Graphical representation of the double-game attack used to prove Claim 1 of the GGM theorem.

$$D_{12} \qquad\qquad\qquad A_{12} \qquad\qquad\qquad C$$

$$z_i \,||\, \ldots \,||\, z_t \in \{0,1\}^{2\lambda t}$$

$$x_j \,||\, y_j \in \{0,1\}^n$$

$$t \left\{\; z_{i(y_j)}^{x_j} \right.$$
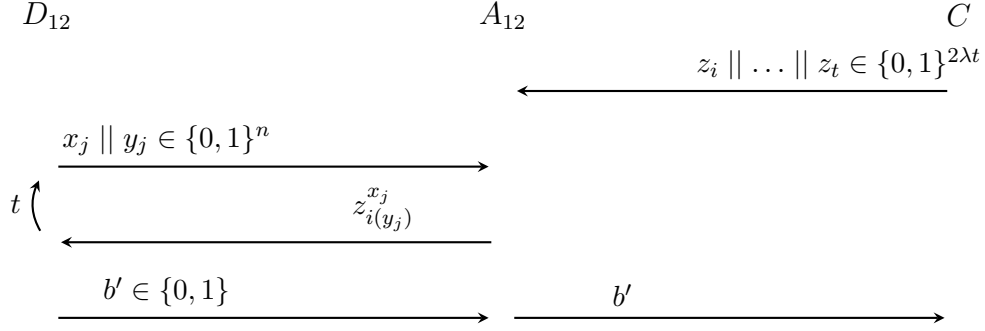
$$b' \in \{0,1\} \qquad\qquad\qquad b'$$

Figure 4.6: Graphical representation of the double-game attack used to prove Claim 2 of the GGM theorem.

4.3 Encryption modes for SKEs

After proving that OWFs can be used to create PRFs through PRGs, we're going to show that PRFs are enough to achieve practical symmetric-key encryption. As we'll see, for some practical applications we strictly require that the pseudorandom functions are invertible, i.e. that they are PRPs. Practical functions with this property are commonly called **blockchiphers** (the concept of blockchipers was used way before the introduction of the concept of PRPs). From a theoretical prospective, we'll discuss how PRFs can be used to derive PRPs. From a practical prospective, instead, we'll discuss some blockchipers used in the real world.

First of all, we observe that real-world encryption systems must work with messages of *different lengths.* Formally, we say that these SKEs have **Variable Input Length (VIL)**, meaning that for each message m it holds that $|m| = \text{poly}(\lambda)$, where λ is the key length. In order to achieve security for VIL SKEs, we require a "standardized way" to encrypt messages of variable length. To solve this issue, each message m is split into a variable number of **blocks** of an equal fixed length n (commonly is a power of 2, such as 256). In other words, each message m is considered as $m = (m_1, \ldots, m_d)$ where $d \in \mathbb{N}$ and $m_i \in \{0,1\}^n$.

The simplest encryption type is the **Electronic Code Book (ECB)** mode, where the scheme simply applies the selected PRF to every block of the message.

> **Definition 4.5: Electronic Code Book (ECB) mode**
>
> Let $\mathcal{F} = \{F_K : \{0,1\}^n \to \{0,1\}^n\}_{K \in \{0,1\}^\lambda}$ be a PRF. Let $K \in \{0,1\}^\lambda$ be a chosen key. On input $m = (m_1, \ldots, m_d)$ the ECB encryption mode returns the ciphertext $c = (c_1, \ldots, c_d)$ constructed as:
>
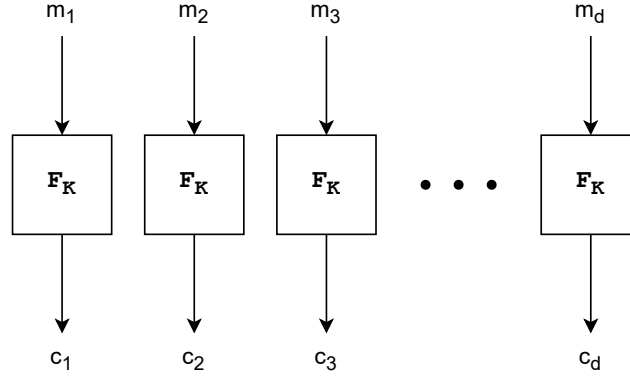> - For all $i \in [d]$ it holds that $c_i = F_K(m_i)$

Figure 4.7: Graphical representation of ECB mode.

Clearly, the ECB mode cannot be CPA-secure since it is deterministic. As discussed in the PRF-OTP example, a nonce value is sufficient to make the ecryption non-deterministic. In encryption modes, this nonce value is usually referred to as **Initialization Vector (IV)**. In order to make decryption harder to bruteforce, common encryption modes are based on *sequential encryption*: some value produced during the encryption of block m_i is used for the encryption of block m_{i+1}. This allows us to restrict our interests to the secretness of the key and the IV. Moreover, it also makes decryption unparallelizable, further improving the security of the SKE. The first type of encryption mode based on this idea is the **Cipher Block Chaining (CBC)**.

Definition 4.6: Cipher Block Chaining (CBC) mode

Let $\mathcal{F} = \{F_K : \{0,1\}^n \to \{0,1\}^n\}_{K \in \{0,1\}^\lambda}$ be a PRF. Let $K \in \{0,1\}^\lambda$ be a chosen key. On input $m = (m_1, \ldots, m_d)$ the CBC encryption mode returns the ciphertext $c = (c_1, \ldots, c_d)$ constructed as:

- $c_0 \in_R U_n$ is a nonce value (IV)
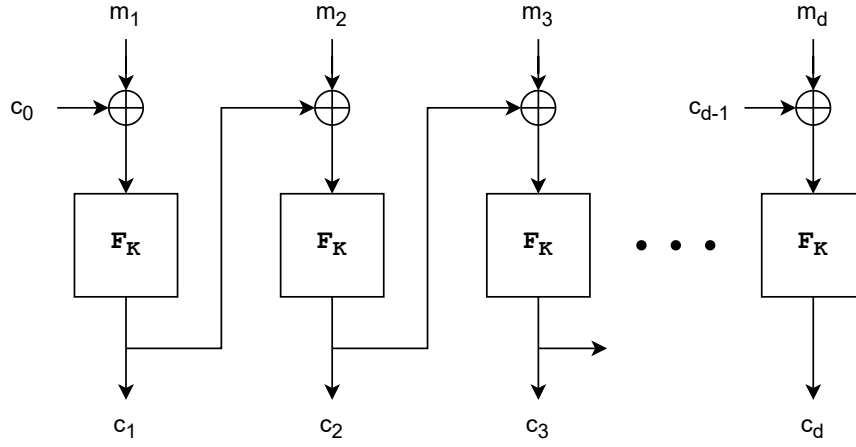- For all $i \in [d]$ it holds that $c_i = F_K(c_{i-1} \oplus m_i)$

Figure 4.8: Graphical representation of CBC mode.

Other types of encryption mode are pretty much based on ideas similar to CBC. The only real difference lies in the order of operations. For instance, the **Cipher Feedback (CFB)** mode XORes each block after applying the PRF to the nonce value, instead of before. We observe that this was the idea behind our PRF-OTP example, the only difference lies in the chaining of encryptions.

Definition 4.7: Cipher Feedback (CFB) mode

Let $\mathcal{F} = \{F_K : \{0,1\}^n \to \{0,1\}^n\}_{K \in \{0,1\}^\lambda}$ be a PRF. Let $K \in \{0,1\}^\lambda$ be a chosen key. On input $m = (m_1, \ldots, m_d)$ the CFB encryption mode returns the ciphertext $c = (c_1, \ldots, c_d)$ constructed as:

- $c_0 \in_R U_n$ is a nonce value (IV)

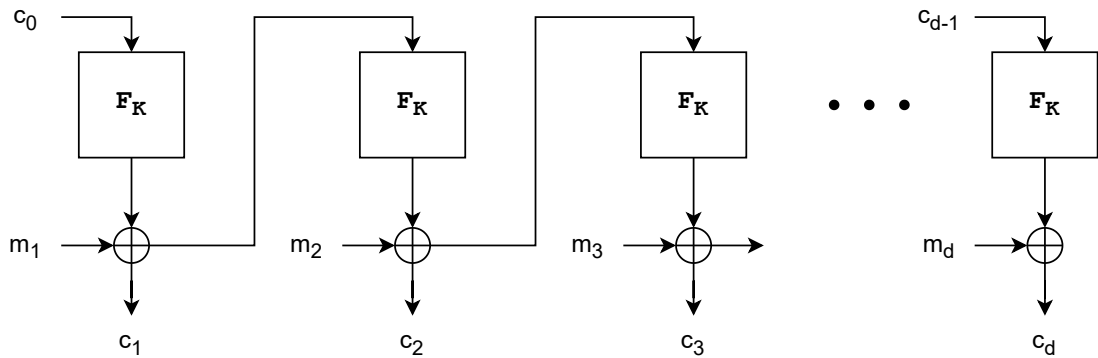- For all $i \in [d]$ it holds that $c_i = F_K(c_{i-1}) \oplus m_i$

Figure 4.9: Graphical representation of CFB mode.

Another encryption mode extremely similar to CFB is the **Output Feedback (OFB)** mode, where the output of the PRF is directly forwarded instead of being XORed first.

> ### Definition 4.8: Output Feedback (OFB) mode
>
> Let $\mathcal{F} = \{F_K : \{0,1\}^n \to \{0,1\}^n\}_{K \in \{0,1\}^\lambda}$ be a PRF. Let $K \in \{0,1\}^\lambda$ be a chosen key. On input $m = (m_1, \ldots, m_d)$ the OFB encryption mode returns the ciphertext $c = (c_1, \ldots, c_d)$ constructed as:
>
> - $q_0 \in_R U_n$ is a nonce value (IV)
>
> - For all $i \in [d]$ it holds that $q_i = F_K(q_{i-1})$
>
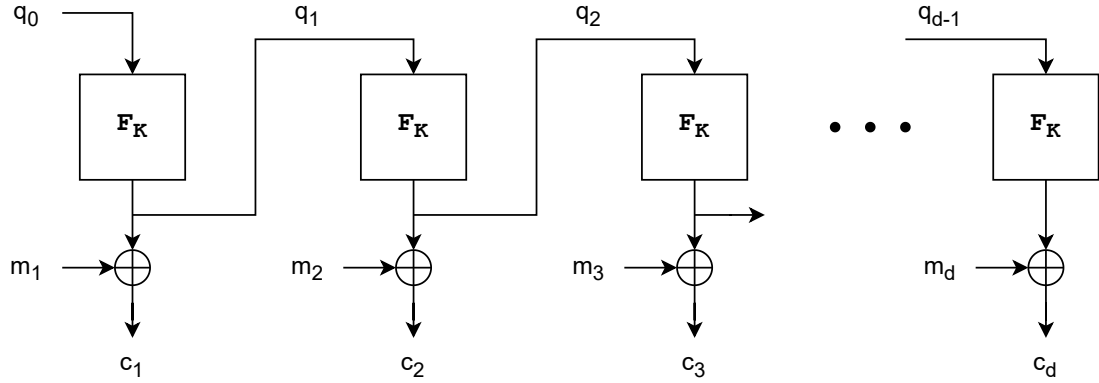> - For all $i \in [d]$ it holds that $c_i = q_i \oplus m_i$

Figure 4.10: Graphical representation of OFB mode.

The last standard encryption mode we discuss is **Counter (CTR)** mode. Instead of using the previous (partial) output as the next input for the PRF, this mode gradually increases the initial nonce value and uses it for the next encryptions.

> ### Definition 4.9: Counter (CTR) mode
>
> Let $\mathcal{F} = \{F_K : \{0,1\}^n \to \{0,1\}^n\}_{K \in \{0,1\}^\lambda}$ be a PRF. Let $K \in \{0,1\}^\lambda$ be a chosen key. On input $m = (m_1, \ldots, m_d)$ the CTR encryption mode returns the ciphertext $c = (c_0, c_1, \ldots, c_d)$ constructed as:
>
> - $r \in_R U_n$ is a nonce value (IV)
>
> - $c_0 = r$
>
> - For all $i \in [d]$ it holds that $c_i = F_K(r + i - 1) \oplus m_i$
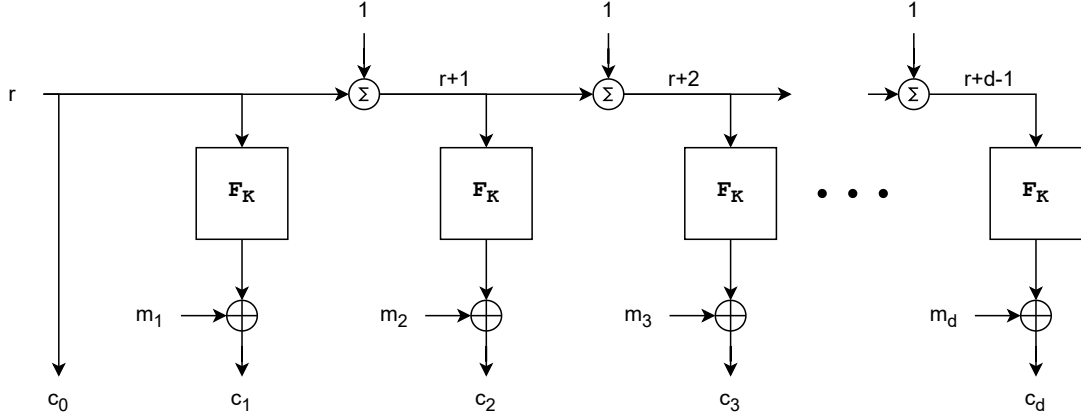>
> *Note*: the sum operation is in mod 2.

Figure 4.11: Graphical representation of CTR mode.

CBC, CFB, OFB and CTR can all be proven to be CPA-secure for VIL. Instead of proving the security of all of them, we'll give an idea by proving the result only for CTR (which is actually harder to prove).

Theorem 4.2: CPA-security through PRFs

CBC, CFB, OFB and CTR are CPA-secure for VIL when constructed using a PRF

Proof. We prove the theorem only for CTR. Let $G(\lambda, b) := \text{Game}_{\Pi,A}^{\text{CPA}}(\lambda, b)$, where Π is a CTR mode scheme constructed through a PRF $\mathcal{F}$.

Recall that for $G(\lambda, b)$ the challenge is given by two messages m_0^*, m_1^* such that $m_b^* = (m_{b,1}^*, \ldots, m_{b,d^*}^*)$ and an answer $c^* = (c_0^*, c_1^*, \ldots, c_{d^*}^*)$ such that $c_0^* = r^* \in_R U_n$ and $c_i^* = F_K(r^* + i - 1) \oplus m_{b,i}^*$.

We define an hybrid distribution for both values of b. Let $H(\lambda, b)$ be the distribution obtained by replacing $F_K(\cdot) \in \mathcal{F}$ with $R(\cdot) \in \mathcal{R}(n \to n)$ in the game $G(\lambda, b)$ (in other words, we're using a random function R for the encryption instead of F_K).

Claim 1: $G(\lambda, b) \approx_c H(\lambda, b)$ for $b \in \{0, 1\}$

Proof of Claim 1. We reduce the distringuishability of F_K from R to the distinguishability of $G(\lambda, b)$ from $H(\lambda, b)$. Fix $b \in \{0, 1\}$. By way of contradiction, suppose that $G(\lambda, b) \not\approx_c H(\lambda, b)$, i.e. there is an adversary A that distringuished $G(\lambda, b)$ from $H(\lambda, b)$ with probability at least n^{-c} for some $c > 0$.

Let A' be the adversary for $\mathcal{F}$ defined as follows:

1. The adversary A' starts both with the challenger C and the other adversary A

2. For $i \in [t]$ iterations, with $t = \text{poly}(\lambda)$, A sends a message $m^{(i)} = (m_1^{(i)}, \ldots, m_{d_i}^{(i)})$ to A'

3. Upon receiving $m^{(i)}$, A' sends $r^{(i)}, r^{(i)}+1, \ldots, r^{(i)}+d-1$ to the challenger C, which answers with $z_1^{(i)}, \ldots, z_d^{(i)}$

4. A' sends $c^{(i)} = (r^{(i)}, c_1^{(i)}, \ldots, c_{d_i}^{(i)})$ back to A, where $c_j^{(i)} = z_j^{(i)} \oplus m_j^{(i)}$

5. After i iterations, the above istructions are repeated again with the challenge messages m_0^*, m_1^*, getting the answer c^*

6. Once A receives c^* it returns a final answer $b' \in \{0, 1\}$ and A' forwards b' to C

7. If $b' = b$, A' wins the PRF-game with C, otherwise it loses.

The idea behind the above simultaneous game is to use the challenger C to compute the outputs of either a pseudorandom function or a truly random one (depends on the value of b) and use those outputs to play the game with A.

If $b = 0$, C will use a pseudorandom function to compute the outputs, thus A will recognize that we're playing $G(\lambda, b)$, outputting 0. If $b = 1$, C will use a truly random function to compute the outputs, thus A will recognize that we're playing $H(\lambda, b)$, outputting 1. Therefore, A' distinguishes F_K from R with non-negligible probability, contradicting the fact that $\mathcal{F}$ is a PRF. $\qquad\square$

Now, show that both $H(\lambda, 0)$ and $H(\lambda, 1)$ are indistinguishable from $U_{n(d^*+1)}$, concluding the proof of the theorem through transitivity of $\approx_c$.

Claim 2: $H(\lambda, b) \approx_c U_{n(d^*+1)}$ for $b \in \{0, 1\}$

Proof of Claim 2. Consider the values $R(r^*), R(r^*+1), \ldots, R(r^*+d^*-1)$ computed during the challenge query of the game $H(\lambda, b)$. Similarly, consider the values $R(r^{(i)}), R(r^{(i)}+1), \ldots, R(r^{(i)}+d_i-1)$ computed during the encryption queries.

Let B_i be the binary "bad event" such that $B_i = 1$ if $\exists j \in [0, d_i - 1], j^* \in [0, d^* - 1]$ such that $r^{(i)} + j = r^* + j^*$, otherwise $B_i = 0$. More generally, let $B = B_1 \cup \ldots cup B_t$

Consider the conditioning over the event $B = 0$. When $B = 0$, the two sequences never overlap over each $i \in [t]$, which means that the challenge ciphertext is distributed as:

- $c_0^* = r^*$

- $c_{j^*}^* = m_{b,j^*}^* \oplus u_{j^*}$ with $u_{j^*} \in_R U_n$ for $j^* \in [0, d^* - 1]$

which is indistinguishable from $U_{n(d^*+1)}$. As a consequence of this, we get that:

$$\mathrm{SD}(H(\lambda, b); U_{n(d+1)}) \leq \Pr[B = 1]$$

We're left with computing $\Pr[B = 1]$. Fix $i \in [t]$. Without loss of generality, assume that $d_i = d^* = t = q$. Since the event $B_i = 1$ holds when $r^*, r^*+1, \ldots, r^*+1-1$ overlaps with $r^{(i)}, r^{(i)}+1, \ldots, r^{(i)}+q-1$, this can happen only when:

$$r^* - q + 1 \leq r^{(i)} \leq r^* + q - 1$$

Therefore, we have that:

$$\Pr[B_i = 1] \le \frac{1 + (r^* + q - 1) - (r^* - q - 1)}{2^n} = \frac{2q - 1}{2^n}$$

Finally, through the union bound we conclude that:

$$\Pr[B = 1] \le \sum_{i=1}^{q} \Pr[B_i = 1] = \frac{q(2q - 1)}{2^n}$$

thus $H(\lambda, b) \approx_c U_{n(d^*+1)}$. $\qquad\square$

The two claims conclude that $G(\lambda, 0) \approx_c H(\lambda, 0) \approx_c U_{(}n(d^* + 1)) \approx H(\lambda, 1) \approx_c G(\lambda, 1)$.
$\qquad\square$

4.4 Computationally secure MACs

After showing that the common encryption schemes are CPA-secure, we focus on secure message authentication. Conside any tag function $\mathrm{Tag} : \mathcal{K} \times \mathcal{M} \to \mathcal{T}$. When can this function be considered secure under a computational setting? The idea is to make it hard to forge a pair (m^*, τ^*) such that $\mathrm{Tag}(K, m^*) = \tau^*$ when K is unknown. This concept is formally known as **Unforgeability under Chosen-Message Attacks (UFCMA)**.

In $\mathrm{Game}_{\Pi,A}^{\mathrm{UFCMA}}(\lambda)$, the adversary is given access to an oracle for the Tag function and it is allowed to make $t = \mathrm{poly}(\lambda)$ tag queries to the oracle. After completing the tag queries, the adversary returns a pair (m^*, τ^*) to the challenger (with $m^* \ne m_i$ for all $i \in [t]$). The challenger computes the tag of m^* and compares it with τ^*. If $\mathrm{Tag}(K, m^*) = \tau^*$ is true, the adversary wins. Otherwise, the challenger wins.
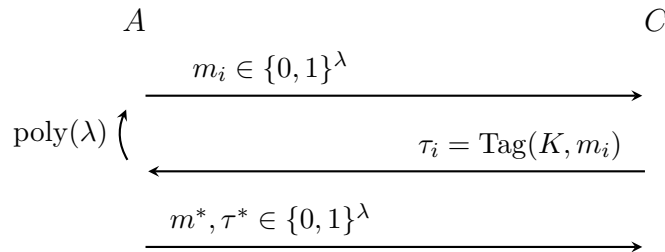
Figure 4.12: Graphical representation of $\mathrm{Game}_{\Pi,A}^{\mathrm{UFCMA}}(\lambda)$

Definition 4.10: UFCMA-security

Let $\Pi = (\mathrm{Tag})$ be a MAC. We say that Π is UFCMA-secure if:

$$\Pr[\mathrm{Game}_{\Pi,A}^{\mathrm{UFCMA}}(\lambda) = 1] \le \mathrm{negl}(\lambda)$$

for all PPT adversaries A.

Pseudorandom functions are intuitively an easy way to get an UFCMA-secure MAC: since the function is hard to invert, the adversary won't be able to infer informations on the key through message-tag pairs. However, this works only for FIL (Fixed Input Length) schemes.

> **Theorem 4.3: UFCMA-security through PRFs**
>
> If $\mathcal{F} = \{F_K : \{0,1\}^n \to \{0,1\}^\lambda\}_{K \in \{0,1\}^\lambda}$ is a PRF family, $\text{Tag}(K,m) = F_K(m)$ is UFCMA-secure for FIL.

Proof. Let $G(\lambda) := \text{Game}_{\Pi,A}^{\text{UFCMA}}(\lambda)$. Let $H(\lambda)$ be the hybrid distribution obtained by replacing $F_K(\cdot) \in \mathcal{F}$ with $R(\cdot) \in \mathcal{R}(n \to n)$ in the game $G(\lambda)$ (in other words, we're using a random function R for the tag instead of F_K).

Proceeding in the usual way with a simultaneous game, we can reduce the distringuishability of F_K from R to the distinguishability of $G(\lambda)$ from $H(\lambda)$ (omitted, similar to Claim 1 of Theorem 4.2). This concludes that $G(\lambda) \approx_c H(\lambda)$.

Finally, since $H(\lambda)$ uses a truly random function R, the probability of an adversary being able to guess the correct tag for a message is at most 2^{-n}, concluding that Π is UFCMA-secure. $\qquad\square$

4.4.1 Domain expansion for UFCMA-secure MACs

We proved that PRFs can be used to construct a UFCMA-secure MAC for FIL inputs. But what about VIL tags? What about inputs of longer FIL? It's easy to see that simple tricks don't work. For instance, consider the tag function that concatenes the tags of the blocks, i.e. $\text{Tag}(K,m) = \tau$ such that $m = (m_1, \ldots, m_d)$ and $\tau = (\tau_1, \ldots, \tau_d)$. The UFCMA game requires that message in the challenge query is different from the messages in the tag queries. However, a simple trick suffices to "break" this constraint:

1. The adversary queries $m_1 \| m_2$ and gets $\tau_1 \| \tau_2$

2. The adversary then queries $m_3 \| m_4$ and gets $\tau_3 \| \tau_4$

3. The adversary challenges with the pair $(m_1 \| m_3, \tau_1 \| \tau_3)$ and wins with probability 1

Consider now the tag function that XORes the blocks and returns a single tag, i.e. $\text{Tag}(K, \bigoplus_{i=1}^d m_i) = \tau$. It's easy to see that this construction still doesn't work:

1. The adversary queries $m_1 \| m_1$ and gets τ_1

2. The adversary challenges with the pair $(m_2 \| m_2, \tau_1)$ and wins with probability 1

It seems that our construction cannot be easily changed to allow VIL or longer FIL inputs. However, we still have a trick up our sleeve: we can hash an input of length nd and reduce it to length n, preserving the UFCMA security of the construction. In other words, we have that $\text{Tag}(K,m) = F_K(h_s(m))$ with $h_s \in \mathcal{H}$, for an hash family $\mathcal{H} = \{h_s : \{0,1\}^{nd} \to \{0,1\}^n\}_{s \in \{0,1\}^\lambda}$.

Does this idea work with every type of hash family? Inuitively, no. The hash family should be such that every hash function hardly collides on different inouts, meanin that it should be hard to have two messages m, m' with $m \neq m'$ such that $h_s(m) = h_s(m')$. Depending on the degree of hardness, these hash families are called ϵ-**almost universal**.

Definition 4.11: Universal hash families

Let $\mathcal{H} = \{h_s : \{0,1\}^{nd} \to \{0,1\}^n\}_{s \in \{0,1\}^\lambda}$ be a family of hash functions. We say that $\mathcal{H}$ is ε-almost universal if $\forall m, m' \in \mathcal{M}$ with $m \neq m'$ it holds that:

$$\Pr_{s \in_R \{0,1\}^\lambda}[h_s(m) = h_s(m')] \leq \varepsilon$$

When $\varepsilon = \mathrm{negl}(\lambda)$, the family is said to be *almost universal (AU)*. When $\varepsilon = 2^{-n}$, the family is said to be *perfectly universal (PU)*.

As the name suggests, PU hash families are considered *perfect* in a sense that it is practically impossible to get a collision. Nonetheless, for our purposes AU hash families are enough. In fact, it can be proven that composing them with a PRF yields another PRF.

Theorem 4.4: Composition of PRFs with AU hash families

Given a PRF family $\mathcal{F} = \{F_k : \{0,1\}^n \to \{0,1\}^n\}$ and an AU hash family $\mathcal{H} = \{h_s : \{0,1\}^{nd} \to \{0,1\}^n\}_{s \in \{0,1\}^\lambda}$, the family $\mathcal{F}(\mathcal{H})$ defined as:

$$\mathcal{F}(\mathcal{H}) = \{F_K \circ h_s : \{0,1\}^{nd} \to \{0,1\}^n\}_{(K,s) \in \{0,1\}^{2\lambda}}$$

is a PRF family.

Proof. Consider the two games $G(\lambda, b) := \mathrm{Game}_{\mathcal{F}(\mathcal{H}),A}^{\mathrm{PRF}}(\lambda, b)$ with $b \in \{0,1\}$. We recall that $G(\lambda, 0)$ is the PRF game where we use a function $F_K(h_s(\cdot)) \in \mathcal{F}(\mathcal{H})$, while $G(\lambda, 1)$ is the PRF game where we use a truly random function $R \in \mathcal{R}(nd \to n)$.

Ler $H(\lambda)$ be the hybrid distribution that plays the PRF game with the function $R'(h_s(\cdot))$, where $R' \in \mathcal{R}(n \to n)$. It can be proven that $G(\lambda, 0) \approx_c H(\lambda)$ in the standard way by reducing the distringuishability of F_K from R' to the distringuishability of $G(\lambda, 0)$ from $H(\lambda)$.

Therefore, it remains to show that $H(\lambda) \approx G(\lambda, 1)$. Let $m_1, \ldots, m_t$ be the encryption queries made by the adversary, where $t = \mathrm{poly}(\lambda)$. Let B be the bad event such that $B = 1$ if $\exists i, j \in [t]$ with $i \neq j$ for which $h_s(m_i) = h_j(m_j)$. By conditioning on $B = 0$, no collisions occurr, thus:

$$\mathrm{SD}(H(\lambda); G(\lambda, 1)) \leq \Pr[B = 1]$$

Through the union bound, we have that:

$$
\begin{aligned}
\Pr[B = 1] &= \Pr_{s \in_R \{0,1\}^R}[\exists i, j \in [t] \ i \neq j \ \text{s.t.} \ h_s(m_i) = h_s(m_j)] \\
&\leq \sum_{i=1}^{t} \sum_{\substack{j=0: \\ j \neq i}}^{t} \Pr_{s \in \{0,1\}^\lambda}[h_s(m_i) = h_s(m_j)] \\
&= \binom{t}{2} \varepsilon \\
&= \text{negl}(\lambda)
\end{aligned}
$$

Therefore, $H(\lambda) \approx_c G(\lambda, 1)$. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\square$

Surprisingly, it is very easy to construct universal hash families, in particular through *Galois fields*. We give two examples:

1. Let $\mathbb{F} = \mathrm{GF}(2^n)$. Let $m \in \mathbb{F}^d$ be the input and let $s \in \mathbb{F}^d$ be the hash seed, where $m_i, s_i \in \mathbb{F}$. We define each $h_s(\cdot)$ as the scalar product between the input and the hash seed:

$$
h_s(m) = \langle s, m \rangle = \sum_{i=1}^{d} s_i m_i
$$

Then, given $m, m' \in \mathbb{F}^d$ with $m \neq m'$, we have that:

$$
\begin{aligned}
h_s(m) = h_s(m') &\iff \langle s, m \rangle = \langle s, m' \rangle \\
&\iff \sum_{i=1}^{d} s_i m_i = \sum_{i=1}^{d} s_i m_i' \\
&\iff \sum_{i=1}^{d} s_i(m_i - m_i') = 0 \\
&\iff \sum_{i=1}^{d} s_i \delta_i = 0
\end{aligned}
$$

where $\delta_i = m_i - m_i'$ for each $i \in [d]$. Since $m \neq m'$, there is at least one index $j \in [d]$ such that $m_j \neq m_j'$, therefore $\delta_j \neq 0$:

$$
\begin{aligned}
h_s(m) = h_s(m') &\iff \sum_{i=1}^{d} s_i \delta_i = 0 \\
&\iff s_j = -\frac{1}{\delta_j} \sum_{\substack{i=1: \\ j \neq i}}^{d} s_i \delta_i
\end{aligned}
$$

which is true with probability at most 2^{-d}. This concludes that $\mathcal{H}$ is a perfectly universal hash family.

2. Let $\mathbb{F} = \mathrm{GF}(2^n)$. Let $m \in \mathbb{F}^d$ be the input and let $s \in \mathbb{F}$ be the hash seed, where $m_i, s_i \in \mathbb{F}$. We define each $h_s(\cdot)$ as the d-degree polynomial $Q_m(s) \in \mathbb{F}[s]_d$ with coefficients $m_1, \ldots, m_d$ and variable s:

$$h_s(m) = Q_m(s) = \sum_{i=1}^{d} m_i s^{i-1}$$

Then, given $m, m' \in \mathbb{F}^d$ with $m \neq m'$, we have that:

$$h_s(m) = h_s(m') \iff Q_m(s) = Q_{m'}(s)$$
$$\iff \sum_{i=1}^{d} m_i s^{i-1} = \sum_{i=1}^{d} m_i' s^{i-1}$$
$$\iff \sum_{i=1}^{d} (m_i - m_i') s^{i-1} = 0$$

which is true with probability $(d-1)2^{-n}$. This concludes that $\mathcal{H}$ is an almost universal hash family.