ECOLE
POLYTECHNIQUE
DE BRUXELLES

Introduction to Cryptography

École Polytechnique de Bruxelles

Professeur : *Gilles* VAN ASSCHE

Sami ABDUL SATER

Année académique : 2021-2022

Contents

Chapter 1

Historical ciphers and general principles

Cryptology is a term merging two similar fields of stuy : cryptograph and cryptanalysis.

- **Cryptography** : the study of secret writing with the goal of **hiding a message**.
- **Cryptanalysis** : breaking cryptosystems.

1.1 Cryptography

In cryptography, to hide a message, there are two things that interest us : hide our content (**Confidentiality**) and authenticating a message (**Authentication**).

1.1.1 Confidentiality

For a generic cryptosystem that ensures confidentiality of a message, we talk about two operations :

- **Encryption** of a plain-text message to get a ciphertext
- **Decryption** of a ciphertext to retrieve a message

We always have to picture two persons communicating with each other, and eventually a third-party intervenant trying to have access to the conversation. Hence, encryption and decryption are meaningless without talking about the intervenants, that we choose to name Ali and Bachar [1].

Ali and Bachar's communicating schema is the following : the first encrypts a message, sends it to the seconds that knows how to decrypt it to find the original content of the message. As they must be the only ones able to encrypt and decrypt the same way, they must have some kind of **key**.

This key generation/sharing/storage is the source of the division of cryptograph in two kinds : a symmetric way or an asymmetric way.

- In **Symmetric crypto**, also called **secret-key** crypto, both parts have an encryption and a decryption method, and they **share the same key that is secret**, kepts out of the sight of any outsider. We also assume that the encryption and decryption algorithms are **publicly known**.

[1] Instead of Alice and Bob, let's change continent a bit.

- In **Asymmetric crypto** (since 1976), the two possess both a private and a public key. They share their public key, but never their private key !

So in general, we will talk about encryption as a mechanism that takes a message m, encrypts it with a key k_E to get a ciphertext c, and sends it. As for decryption, it takes a ciphertext c, decyrpts it under a key k_D to obtain m.

- Symmetric : $k_E = k_D$

- Asymmetric : k_E is public, k_D is private.

1.1.2 Authentication

As for authentication, we **are not trying to hide anything**. The message is sent in full plain-text from Ali to Bachar. Our goal here is to **check** the source of our message, assure its authenticity.

Similarly to encryption/decryption, we here have two mechanisms with keys :

- Authentication : Ali generates a tag under a key k_A, and sends the couple (m, tag) to Bachar

$$m \Rightarrow (m, \text{tag})$$

- Verification : Bachar receives (m, tag), and under key k_V, identifies the source.

$$(m, \text{tag}) \Rightarrow \{m, \perp\}$$

In symmetric crypto, $k_A = k_V$ and is **secret**. In this case, the tag is more commonly called **Message Authentication Code** (MAC).

In asymmetric crypto, k_A is private and k_V is public. In this case, the tag is called "**signature**". To do so, with the message, Ali sends a tag.

Confidentiality

Encryption

- plaintext $\Rightarrow$ ciphertext
- Under key $k_E \in K$

Decryption

- ciphertext $\Rightarrow$ plaintext
- Under key $k_D \in K$

Symmetric cryptography: $k_E = k_D$ is the secret key.

Asymmetric cryptography: k_E is public and k_D is private.

Authenticity

Authentication

- message $\Rightarrow$ (message, **tag**)
- Under key $k_A \in K$

Verification

- (message, **tag**) $\Rightarrow$ {message, $\perp$}
- Under key $k_V \in K$

Symmetric cryptography: $k_A = k_V$ is the secret key.
The tag is called a *message authentication code* (MAC).

Asymmetric cryptography: k_A is private and k_V is public.
The tag is called a *signature*.

1.2 Cryptanalysis

Cryptanalysis is the field that studies algorithms and ways of breaking a cryptosystem. This means, recovering the message, or recovering the key. There are several ways to do this, going from ”little average mathematician boi that exploits the inner structure of the scheme” to the ”chad asking you your password with a gun pointing to the head”. All the methods, from the first to the last, are part of **cryptanalysis**. But in this course, we focus on what we call **mathematical cryptanalysis**. We will also place ourselves in the Kerckhoff's principles.

Kerckhoff's principles for cryptographic systems

The security of a cryptosystem must only rely on the **secrecy of its key**. We hence assume, when evaluating the security of a system, that everything is known : length of the messages, encryption and decryption scheme.

1.2.1 Mathematical cryptanalysis

Some definitions

- Key space : set of all possible keys
- Brute-force attack : attack that tries all the keys of the key space.

This branch studies brute-force attacks and analytical attacks. Analytical attacks can be of several types : exploiting some statistical patterns, length extension attack, ...

1.2.2 Key length

This is an informative section on key length, just to develop an intuition on the impact of the length of a key in a cryptographic system.

First, it is important to mention that the key length in a **symmetric** crypto system is relevant only if the brute-force attack is the best-known attack.

Excluding this case, then the guaranteed security of a cryptosystem according in function with the key length is very different depending on the kind of crypto : a 80-bit key in symmetric crypto can ensure the same security as a 1024-bits key asymmetric scheme (such as RSA).

1.2.3 Time to break

Here is an indicator of the meaning of the ”time-to-break” (TTB) of cryptosystems in function of the key length for a **symmetric scheme**.

Key length	Security estimation
56–64 bits	short term: a few hours or days
112–128 bits	long term: several decades in the absence of quantum computers
256 bits	long term: several decades, even with quantum computers that run the currently known quantum computing algorithms

1.3 Some ciphers

1.3.1 Shift encryption scheme

Chose a key k between 0 and 26, message m also between 0 and 26. The shift encryption scheme is all about XORing the message with the key :

$$\begin{aligned} E_k(m) &= m + k \mod 26 = c \\ D_k(c) &= c - k \mod 26 = m \end{aligned}$$

It is really dumb : as the key space is very limited (26), it is quickly subject to brute-force methods.

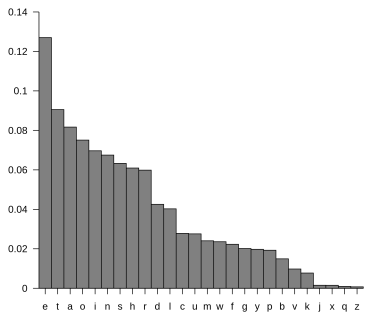
1.3.2 Mono-alphabetic substitution

The mono-alphabetic cipher consists in replacing each letter of the message by a corresponding letter in a mixed alphabet chosen randomly. So we define a substitution table, and we apply our mapping. Let's break it down a little bit.

- It is a symmetric scheme.
- The key space is $s = 26! > 4 \cdot 10^{26}$: a brute-force attack would take some time.
- It is breakable using a statistical approach.

Historical ciphers

Mono-alphabetic substitution



Frequency of invidual letters in English

9 / 57

Historical ciphers

Mono-alphabetic substitution

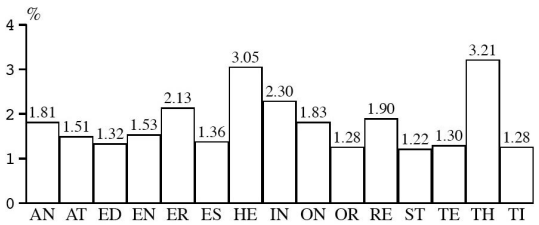


Figure 7.6: *Frequency of 15 common digrams in English text.*

10 / 57

1.3.3 Poly-alphabetic substitution

Instead of encrypting a entire message with the same mapping, we here divide the message into t blocks.

$$x = x_1 \| x_2 \| \dots \| x_t$$

Then, we define a mapping for each block. This will be encoded in the key k of the scheme. Indeed, each $k \in K$ will define a **set of permutations**

$$k \Rightarrow (p_1, p_2, \dots, p_t) \, .$$

Hence, $E_k(x)$ will be given by

$$E_k(x) = p_1(x_1)\|p_2(x_2)\|\dots\|p_t(x_t)\|$$

As for the decryption key k', it needs to define the set of the t corresponding inverse permutations :

$$k' \Rightarrow (p_1^{-1}, p_2^{-1}, \dots, p_t^{-1}) .$$

1.3.4 Vigenère cipher

- Message m of length $|m|$, chosen in $(\mathbb{Z}_{26})^\star$
- Key space $K \subset (\mathbb{Z}_{26})^t$. Size : logically 26^t
- Key k taken randomly in K, so

$$k = (k_0, k_1, \dots, k_{t-1}) \in K$$

Then, the encryption of m will result in the concatenation of the XORing of each bit m_i with a part of the key that. Remember that the key has only t parts, so we will repeat the same parts if the message is very long ! The same with a modular difference for the decryption

$$\begin{aligned} E_k(m) \equiv E_k(m_0\|m_1\|\dots\|m_{|m|-1}) &= \|_{0\le i\le |m|-1} (m_i + k_i \mod t) = c \\ D_k(c) \equiv E_k(c_0\|c_1\|\dots\|c_{|c|-1}) &= \|_{0\le i\le |c|-1} (c_i - k_i \mod t) = m \end{aligned}$$

In practice, the key can actually be a t-long string. During the process, each character is converted to a number.

Historical ciphers

Vigenère cipher – example

plaintext: `rendezvousahuitheure`
key: `hello` (7 4 11 11 14)

17 04 13 03 04 25 21 14 20 18 00 07 20 08 19 07 04 20 17 04
07 04 11 11 14 07 04 11 11 14 07 04 11 11 14 07 04 11 11 14
⇓
24 08 24 14 18 06 25 25 05 06 07 11 05 19 07 14 08 05 02 18

ciphertext: YIYOSGZZFGHLFTHOIFCS

Cryptanalysis of Vigenère cipher

We stick to Kerckhoff's principles : so we know how every message is encrypted, and the only thing that is kept secret is the key k. We don't even know its length t. And as it has been seen before, some bits of m are encoded with the same key chunk because of the modulo in the XOR ($k_{i \mod t}$). So we can group the bits of m that are encoded with the same chunks.

$$m_i, m_{i+t}, m_{i+2t} \longleftarrow k_{i \mod t}$$

For each chunk, we see that we actually have a simple shift encryption scheme that can be easily broken. So, if we know the length t of the key, we can break Vigenère cipher easily.

How to find the length of the key ? We can try brute-force. It will work. But there is a better method, using the **index of coincidence**.

Now that we have introduced the fundamental ciphers, we can move on some more cryptanalysis definitions : how do we quantify the security of cryptosystems ?

1.4 Perfect secrecy (PS – unconditional security)

A first definition that comes into the hand when talking about security of cryptosystems is **perfect secrecy**. It is an **ideal property** that a cryptosystem can achieve. In English, it states that it must not leak any information, even to an adversary with unlimited computational power. In mathematical terms (not real mathematics, but mmmhh), it states the following

Perfect secrecy

An encryption scheme satisfies perfect secrecy an adversary can not distinguish two random encryptions :

- For any two messages $m_1, m_2 \in M$
- For every ciphertext $c \in C$
- Choosing a key $k \in K$

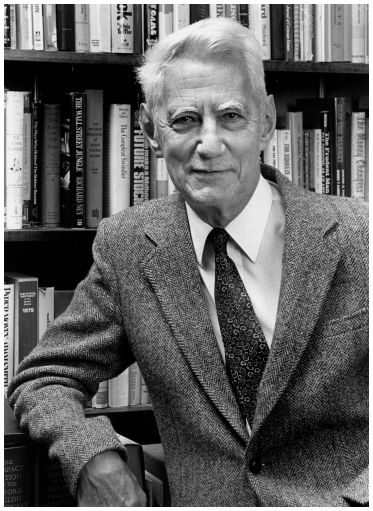
$$\Pr[\text{Enc}_k(m_1) = c] = \Pr[\text{Enc}_k(m_2) = c]$$

1.4.1 Perfect secrecy and length of keys

Claude Shannon showed that for a system to achieve perfect secrecy, the length of the key must be at least the length of the message. Note that it is possible to have a scheme with a key much longer than the message but for it to be not secure at all... It is an implication : it must, but it is not sufficient.

Perfect secrecy vs computational security

Perfect secrecy implies long keys



Claude Shannon (1916-2001)

Claude Shannon showed that, for perfect secrecy, the entropy of the key is at least the entropy of the plaintext, i.e.,

$$\text{perfect secrecy} \Rightarrow H(K) \geq H(M).$$

So the secret key must be at least as long as the plaintext and it may not be reused!

22 / 57

Note that in practical, this is a property that annoys us a lot because for long messages, we must have a longer key. We will see later some *lighter* security definitions.

1.4.2 One-time pad (OTP)

The one-time pad encryption scheme is quite easy : it is close to the Vigenère cipher, except for the fact that **the key is as long as the message**. It can thus ensure PS. But, does it really ?

We are given a message space

$$M = (\mathbb{Z}_2)^t$$

and $C = K = M$. This means we play with binary strings. The key will be written as

$$k = (k_0, k_1, \ldots, k_{t-1}) \ ,$$

and the messages, ciphertexts, can also be similarly written. The scheme is the following :

$$\begin{array}{lll} E_k(m) \equiv E_k(m_0, m_1, \ldots, m_{t-1}) & = (m_i + k_i)_{0 \leq i \leq t-1} & = c \\ D_k(c) \equiv D_k(c_0, c_1, \ldots, c_{t-1}) & = (c_i - k_i)_{0 \leq i \leq t-1} & = m \end{array}$$

It looks indeed similar to the Vigenère cipher. But here, let's compute the probability of knowing a plaintext – ciphertext pair.

$$\begin{array}{lll} \Pr[E_k(m) = c] = \Pr[m \oplus k = c] & & \text{Bit-wise operator} \\ = \Pr[k = m \oplus c] & & \text{Because } m \oplus m = 0 \\ = 2^{-t} & & \text{Because } k \text{ is chosen randomly in } (\mathbb{Z}_2)^t \end{array}$$

We thus prove that the OTP achieves perfect secrecy. It may look like Vigenère's, but in security ways, it is way different. OTP is the ideal scheme, Vigenère's is easily broken as we saw.

Why "one-time" pad ? Because if we use twice the same key for a different message, there is information that is leaked from the system. In particular, we can find a link between the ciphertexts and the messages. Indeed :

If	$c_1 = m_1 \oplus k$	
and	$c_2 = m_2 \oplus k$	
Then	$c_1 \oplus c_2 = m_1 \oplus m_2$	Because $k \oplus k = 0$

1.5 Computational security

As we already saw, perfect secrecy requires the key being at least as long as the message. This annoys us very much. Some systems can be very secure **without achieveing perfect secrecy**. This allows us to leak some information. This introduces us the notion of **computational security**.

Perfect secrecy vs computational security

Computational security

Computational security

A scheme is (t, ϵ)-secure if any adversary running for time at most t, succeeds in breaking the scheme with probability at most ϵ.

Security strength

We say that a scheme is s-bit secure if, for all t, the scheme is $(t, \epsilon(t))$-secure and $\log_2 t - \log_2 \epsilon(t) \geq s$.

Example: exhaustive key search

After t attempts, the probabilty of finding the correct key is $\epsilon(t) = \frac{t}{|K|}$ with $|K|$ the size of the key space. If there are no faster other attacks than this, then the scheme is $s = \log_2 |K|$-bit secure.

28 / 57

We really need to interpret this with the example : if the key space of a system is of size 2^{128} and that for the best attack, the probability of finding k after t attempts is $\varepsilon(t) = t/2^{128}$, then the scheme is $s = 128$-bit secure.

1.6 Security principles

There is a big gap between cryptography and practical cryptosystems. In practical, none of the cryptosystems will see offer perfect secrecy. The only ways to gain confidence in the security of a scheme is by *peer-reviewing*. Particularly, the **algorithm must be public**, first for people to help proving its security/breaking it, but most importantly to stick to Kerckhoff's principles.

1.6.1 Goals of an adversary

Security definitions | For encryption schemes

To what does an encryption scheme must resist?

An adversary can have the following **goals**:

- recovery of the (secret or private) key
- recovery of even some partial information about the plaintext
- a property that distinguishes the scheme from ideal

The adversary is allowed to get (**data model**):

- ciphertexts only
- known plaintexts (and corresponding ciphertexts)
- chosen plaintexts (and corresponding ciphertexts)
- chosen plaintexts and ciphertexts

Attacks continue to work when going down in the two lists above. The best for the attacker is to be able to recover the key with ciphertexts only. A designer will be happy if no cryptanalyst was able to show a disitinguisher even with chosen plaintexts and ciphertexts.

34 / 57

1.6.2 Ability to query the scheme

Offline and online complexities

Computational security

A scheme is (t, d, ϵ)-secure if any adversary running for time at most t and having access to d data, succeeds in breaking the scheme with probability at most ϵ.

- t: offline complexity
- d: online complexity

Security strength

We say that a scheme is s-bit secure if, for all (t, d), the scheme is $(t, d, \epsilon(t, d))$-secure and $\log_2(t + d) - \log_2 \epsilon(t, d) \geq s$.

1.6.3 Taxonomy of attacks

Taxonomy of attacks

To describe an attack, one should specify:

- the goal;
- the data model and the online complexity (d);
- the offline complexity (t) and success probability (ϵ).

Example: exhaustive key search

The exhaustive key search is a **key recovery attack** that requires $d = 1$ **pair*** **of known plaintext / ciphertext** and takes **offline complexity** $t = |K|$ for a **success probability** $\epsilon = 1$.

* Depending on the relative size of the plaintext and the key, this may require more than one pair to avoid multiple key candidates.

1.6.4 Formal definition of an encryption scheme

Security definitions | For encryption schemes

Formal definition of an encryption scheme

An encryption scheme is a triple of algorithms $\mathcal{E} = (\text{Gen}, \text{Enc}, \text{Dec})$ and a plaintext space M.

- **Gen** is a probabilistic algorithm that outputs a secret (or private) key k_D from the key space K. In asymmetric cryptography, it publishes the corresponding public key k_E.
- **Enc** takes as input a secret/public key k_E and message $m \in M$, and outputs ciphertext $c = \text{Enc}_{k_E}(m)$. The range of Enc is the ciphertext space C.
- **Dec** is a deterministic algorithm that takes as input a secret/private key k_D and ciphertext $c \in C$ and output a plaintext $m' = \text{Dec}_{k_D}(c)$.

37 / 57

1.6.5 Semantic security

An encryption scheme is **semantically secure** if : when the adversary can compute with the ciphertext something about the plaintext in a polynomial time, it can also do it without the ciphertext (in a polynomial time).

This immediately implies that **deterministic encryption** is not semantically secure. To make it secure, we must go **probabilistic**. Using a nonce for the key generation.

Nonce

A *nonce* is a number used once. Most likely randomly generated.

1.6.6 INDististinguishability games

Let's play a bit, and introduce the following game. It is called the "IND-game", and we apply it on an encryption scheme $\mathcal{E} = (\text{Gen}, \text{Enc}, \text{Dec})$. If the adversary can not win it with other than with a negligeable advantage against a **random guess**, we are good.

Basically, it ensures that an adversary can not know between two plaintexts, even if they are chosen, which of them was encrypted, if it is given a ciphertext. This game can be twisted and allow the adversary to query the encryption function giving us the IND-CPA game (includes steps 2 and 5)

IND(-CPA/CCA) game

1. Challenger generates a key $k \longleftarrow \text{Gen}()$.
2. [CPA/CCA only] Adversary queries Enc_k with plaintexts of his choice
   [CCA only] and Dec_k with ciphertexts of his choice
3. Adversary chooses two same-length plaintexts $m_1, m_2 \in M \wedge |m_1| = |m_2|$ and gives them to the Challenger.
4. Challenger randomly encrypts m_1 or m_2, gives the ciphertext $c_x = E_k(m_x)$ to the adversary.
5. [CPA/CCA only] Adversary queries Enc_k with plaintexts of his choice
   [CCA only] and Dec_k with ciphertexts of his choice
6. Adv. guesses x' was encrypted
7. Adv. wins if $x = x'$

We note the "Advantage" of a system the probability that this winning situation is far from 1/2, that corresponds to a totally random guess. Formally :

$$\text{Advantage} \equiv \varepsilon = \left| \Pr[\text{win}] - \frac{1}{2} \right| .$$

Note

For symmetric crypto, the IND game only addresses the security of a single encryption. Indeed, it needs the challenger to encrypt, as there is no public key [2]. The game can only be played **online**.

1.6.7 Extending security strength definition

Hence, we can extend the security strength definitions for indististinguishability.
A system id $(t, d, \varepsilon)-$IND-CPA (resp. IND-CCA) secure if any adversary running at most t attempts and having access to d data succeeds in winning the respective game with Advantage at most ε.

Here, we mentioned a limit quantity d of data that we authorize the adversary to query with. But what attempts are we counting ? If we are in asymmetric crypto, then the encryption key is completely public so anyway the adversary can compute any encryption he wants. The decryption key is however private, so d in asymmetric crypto refers to the number of times an adversary can query the decryption scheme, and is hence logic to be found only in the IND-CCA game. Similarly for the symmetric case : here d refers to both encryption and decryption querying because both rely on a secret key. This gives us the table on the slide below :

Security definitions | For encryption schemes

IND-CPA or IND-CCA security strength

Security strength for IND-CPA (resp. IND-CCA)

A scheme is (t, d, ϵ)-IND-CPA (resp. IND-CCA) secure if any adversary running for time at most t and having access to d data, succeeds in winning the IND-CPA (resp. IND-CCA) game with advantage at most ϵ.

What is counted towards the online complexity d?

	Symmetric	Asymmetric
IND-CPA	Enc_k	-
IND-CCA	Enc_k + Dec_k	Dec_k

43 / 57

1.6.8 INDististinguishability using a diversifier (nonce)

In symmetric crypto, encryption is diversified at each use : encryption takes an extra parameter d randomly-chosen, completely public. The decryption also takes d as input.

Thus, the symmetric-key encryption scheme must be modified :

Security definitions | For encryption schemes

Symmetric encryption with diversification

A symmetric-key encryption scheme with diversification is a triple of algorithms $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, a diversifier space D and a plaintext space M.

- Gen is a probabilistic algorithm that outputs a secret key k from the key space K.
- Enc takes as input a secret key k, a diversifier $d \in D$ and message $m \in M$, and outputs ciphertext $c = \mathsf{Enc}_k(d, m)$. The range of Enc is the ciphertext space C.
- Dec takes as input a secret key k, a diversifier $d \in D$ and ciphertext $c \in C$ and output a plaintext $m' = \mathsf{Dec}_k(d, c)$.

Correctness: $\forall k \in K, d \in D, m \in M$, we must have $\mathsf{Dec}_k(d, \mathsf{Enc}_k(d, m)) = m$.

45 / 57

Here is the IND-CPA game with a diversifier :

IND-CPDA game (chosen plaintext and diversifier)

1. Challenger generates a key $k \longleftarrow \text{Gen}()$.
2. Adversary queries Enc_k with (d, m) of his choice.
3. Adversary chooses d and m_1, m_2 and gives them to the Challenger.
4. Challenger randomly encrypts m_1 or m_2, gives the ciphertext $c_x = E_k(d, m_x)$ to the adversary.
5. Adversary queries Enc_k with (d, m) of his choice
6. Adv. guesses x' was encrypted
7. Adv. wins if $x = x'$

But **attention** : the adversary **must** respect that d is a **nonce** !! The nonces used at steps 2, 3, and 5 **must all be different**.

1.7 Authentication schemes

What about authentication schemes ? Recall that our goal here is not to hide a message : the message is sent in plain text. Here, we want to provide a signature of our message, and to recognize someone's signature. So the adversary has the following goals :

- Recover the (secret or private) key
- Forgery : be able to send a message that does not come from an authentic party and is still verified
- Find a property that distinguishes the scheme from ideal.

And he is allowed to get :

- Known messages and corresponding tags
- Chosen messages and corresponding tags

1.7.1 Types of forgeries

- Universal forgery : the attack must be able to work for any message, possibly chosen by a challenger
- Selective forgery : the adversary chooses the message beforehand
- Existantial forgery : the message content is **irrelevant**, the adversary can choose it adaptatively just to make the attack work. We guess that this is not really senseful and a weak attack.

1.7.2 Taxonomy of authentication scheme attacks

Security definitions | For authentication schemes

Taxonomy of attacks

As for encryption, to describe an attack, one should specify:

- the goal;
- the data model and the online complexity (d);
- the offline complexity (t) and success probability (ϵ).

Example: random tag guessing

The random tag guessing is a **universal forgery attack** that submits d random (message, tag) pairs. It requires **no known message** and takes **online complexity** d for a **success probability** $\frac{d}{2^n}$, assuming that the tag length is n bits. (Here t is negligible.)

Example: exhaustive key search

The exhaustive key search is a **key recovery attack** that requires $d = 1$ **known** (**message**, **tag**) **pair** and takes **offline complexity** $t = |K|$ for a **success probability** $\epsilon = 1$.

49 / 57

1.7.3 Formal definition of an authentication scheme

Security definitions | For authentication schemes

Formal definition of an authentication scheme

An authentication scheme is a triple of algorithms $\mathcal{T} = (\mathsf{Gen}, \mathsf{Tag}, \mathsf{Ver})$ and a message space M.

- Gen is a probabilistic algorithm that outputs a secret (or private) key k_A from the key space K. In asymmetric cryptography, it publishes the corresponding public key k_V.
- Tag takes as input a secret/private key k_A and message $m \in M$, and outputs tag $t = \mathsf{Tag}_{k_A}(m)$. The range of Tag is the tag space T.
- Ver is a deterministic algorithm that takes as input a secret/public key k_V, a message $m \in M$ and a tag $t \in T$ and output either m (if the tag is valid) or $\perp$ (otherwise).

50 / 57

1.7.4 Some games for unforgeability : EU-CMA

Security definitions | For authentication schemes

EU-CMA: Existential unforgeability, chosen messages

A scheme $\mathcal{T} = (\mathsf{Gen}, \mathsf{Tag}, \mathsf{Ver})$ is **EU-CMA-secure** if no adversary can win the following game for more than a negligeable probability.

- Challenger generates a key (pair) $k \leftarrow \mathsf{Gen}()$
- Adversary queries Tag_k with messages of his choice
- Adversary produces a $(\text{message}, \text{tag})$ pair, with a message not yet queried
- Adversary wins if $\mathsf{Ver}_k(\text{message}, \text{tag}) \neq \perp$
  (Advantage: $\epsilon = \Pr[\text{win}]$.)

Chapter 2

Secret-key techniques

We now study the case of symmetric crypto. In symmetric crypto, Ali and Bachar have one key each, and this key is private. This is very important to hold into account. In asymmetric crypto, each one has a public key, a private key, must know the public key of his collegues, etc. But here, none of that : an encryption scheme works under a private key. This is why this is the simplest case to study and the first one we study.

We are thus interested here in building a secure encryption scheme, that gets as input a message in plaintext and outputs a ciphertext, as well as a decryption algorithm that allows us to retrieve the message. The definitions of IND-CPA was made very clear : if the system is secure, an adversary that does not know the key has better chance of going random instead of trying to guess !

There are many functions needed to build to *in fine* build a secure symmetric encryption scheme. There is the mechanism of the key, and some more functions such as block ciphers, permutations, etc. All these are useless separately, but at the end they build an encryption scheme. In this course, we will cover :

- Keystream generators and stream ciphers
- Block ciphers
- Permutations

2.1 Basic schema

The basic schema of the communication is the following : Ali wants to send a message to Bachar. Then :

- One key is generated and given to Ali, and to Bachar, under a **secure channel**.
- Ali encrypts the message with the key k,
- transmits it to Bachar in a public channel, in clear : any outsider Omar[1] can observe the thing sent.
- Bachar receives the ciphertext, and decrypts it with the same key k.

[1] Instead of Oscar.

2.2 Stream ciphers vs block ciphers

There are two ways of doing symmetric crypto : with stream ciphers of with block ciphers.

- Stream ciphers process a message bit by bit, XORing every bit with a key stream s_i built from the original key by a keystream generator.
- Block ciphers process a message by separating it into blocks, and processing block by block. Within a block, each bit value has an impact on the output for the block.

2.3 Keystream generators and stream ciphers

Stream ciphers are simple : they need a key as long as the plain text and they XOR it with the message. For this, we take a private key k that is not necessarly very long, and we extend it, building a key stream $s = (s_i)$. This key stream generation process is the main focus for stream ciphers.

The encryption and decryption algorithm are as follows, with the generated key stream $s = (s_i)$. We notice that the two algorithms are the same. It indeed works.

Keystream generators and stream ciphers

What is a stream cipher?

Keystream generator: $G : K \times \mathbb{N} \to \mathbb{Z}_2^\infty$

- inputs a secret key $k \in K$ and a diversifier $d \in \mathbb{N}$
- outputs a long (potentially infinite) keystream $(s_i) \in \mathbb{Z}_2^\infty$

To encrypt plaintext $(m_0, \ldots, m_{|m|-1})$:

$$c_i = m_i + s_i \quad (\text{in } \mathbb{Z}_2)$$

To decrypt ciphertext $(c_0, \ldots, c_{|c|-1})$:

$$m_i = c_i + s_i \quad (\text{in } \mathbb{Z}_2)$$

The diversifier d is public and must be a **nonce**, i.e., unique per encryption for a given key.

6 / 69

2.4 Block ciphers

A block cipher is basically a mapping from :

- Input : secret key $k \in \mathbb{Z}_2^m$, and **input block** $x \in \mathbb{Z}_2^n$
- Output block $y = E_k(x) \in \mathbb{Z}_2^n$.

And we ensure that for each key k, the function has an inverse : $x = E_k^{-1}(y)$.

Note that as it is, we can not use it as encryption scheme, it is completely deterministic so it loses the IND-CPA game.

2.4.1 DES

DES is a block cipher : it takes a block as input, a key, and generates an output block. For a classical DES algorithm, the input block has 64 bits and the key is 56-bits long. Actually, 64 bits come as input for the key, but every $7th$ bit is used as a parity bit, not a key bit. DES is a 56-bit cipher.

$$x \in \mathbb{Z}_2^{64} \qquad k \in \mathbb{Z}_2^{56}$$

With the input block x and the key k, DES works by computing 16 rounds of its underlying network. For each round, it needs a key, that is derived from the original key. Those are subkeys K_i.

$$k \Rightarrow K_1, K_2, \ldots K_{16} \in \mathbb{Z}_2^{48}$$

The section below explains how the subkeys (also called *round keys* are generated).

Key schedule : round keys generation

Input : the 56-bits key. It is all based on permutations, and more precisely on two permutations : PC-1 and PC-2. Here is how 16 48-bits keys are generated.

1. The initial key goes under a bit transposition, through the table of permutation of PC-1 (see figure below). So at the exit of PC-1, we still have a 56-bit key. We separate it into two 28-bits blocks : C_0 and D_0.

$$\text{PC1} \ : \ \mathbb{Z}_2^{56} \rightarrow \mathbb{Z}_2^{56}$$

2. For i going from 1 to 16 :

   - We build C_i from C_{i-1}, D_i from D_{i-1}, thanks to a permutation LS_i
   - LS_i is a circular shift to the left : by one when $i = 1, 2, 9, 16$, by two positions otherwise
   - K_i is obtained by the concatenation $C_i \| D_i$ but again under a permutation. This time, it is PC-2 that handles it, and ignore some bits to make it fit in 48 bits :

$$\text{PC2} \ : \ \mathbb{Z}_2^{56} \rightarrow \mathbb{Z}_2^{48}$$

Block ciphers DES

DES: Tables for PC1 and PC2

PC1						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
above for C_i; below for D_i						
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

PC2					
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

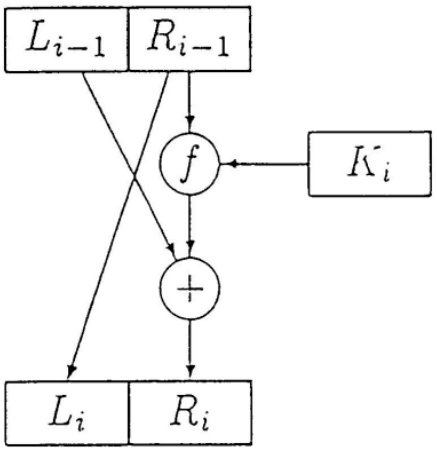
Table 7.4: DES key schedule bit selections (PC1 and PC2).

20 / 69

DES's f function

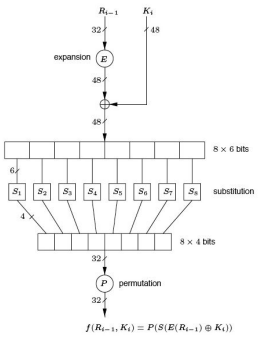
At each round, the current state is processed, divided into two blocks, L_{i-1} and R_{i-1}, using K_i and a function called f. DES's f function works as follows : it takes a 32-bit input, which is the half-message block, and the subkey.

$$f : \mathbb{Z}_2^{32} \times \mathbb{Z}_2^{48} \to \mathbb{Z}_2^{32}$$



Block ciphers DES

Inside DES' f function



$f : \mathbb{Z}_2^{32} \times \mathbb{Z}_2^{48} \to \mathbb{Z}_2^{32}$

1. Expand R through E from 32 to $48 = 16 + 2 \times 16$ bits
2. Compute $E(R) \oplus K_i$
3. Cut in 6-bit blocs B_j
4. $C_j = S_j(B_j)$, 8 different S-Boxes $S_j : \mathbb{Z}_2^6 \to \mathbb{Z}_2^4$
5. Bit transposition $P : \mathbb{Z}_2^{32} \to \mathbb{Z}_2^{32}$

Figure 7.10: DES inner function f.

16 / 69

The $S-$boxes are what make DES non linear, crucial for cryptanalysis.

Non-ideal properties

- There are some weak keys (4) for which DES is an involution
- There are some weak pairs (6) of keys for which DES is an involution
- Complementarity property : $E_k(x) = y \iff E_{\bar{k}}(\bar{x}) = \bar{y}$ reduces the Exhaustive search by 2 (one bit).

Flaws of DES

- Size of the key space : 56 bits is very small... Exhaustive search can be done in 2^{55} operations, which is nowadays feasible.
- Susceptible to mathematic cryptanalysis : differential, linear.

Triple-DES

An improvement is then to take larger keys, for example three keys and building Triple-DES with 168 bits ($= 3 \times 56$), with two keys, 112 bits. Note that 3DES is retro-compatible with single DES, we just have to huse thrice the same key.

$$3\text{DES}_{k_1\|k_2\|k_3} = \text{DES}_{k_3} \circ \text{DES}_{k_2}^{-1} \circ \text{DES}_{k_1}$$

$$3\text{DES}_{k_1\|k_2} = \text{DES}_{k_1} \circ \text{DES}_{k_2}^{-1} \circ \text{DES}_{k_1}$$

Why not simple Double-DES ? Because with two rounds of DES, one can make two times a brute force, one for each system, and breaking the system, in a time of 2^{57} and using memory 2^{56} : storing every attempt on the first system (intercepting information before it comes to the second DES layer), then trying to decrypt the second DES with the saved attempts.

2.4.2 Rijndael and AES

With such flaws for DES, it stinks. Rijndael comes to help. We talk of Rijndael as AES, you should know it, and if you don't, Google the origins. Anyway, AES takes as input a 128-bits block, and there are multiple cases for the key, and for each case ther is a number of rounds of the cipher :

- AES-128 : 128-bits key : 10 rounds
- AES-192 : 192-bits key : 12 rounds
- AES-256 : 256-bits key : 14 rounds

Representation of x in the Galois Field

x is represented as a 4×4 array in the Galoid Field. It is called at any point the **state** of the algorithm. The key, $k \in \mathbb{Z}_2^{128}$.

Round of AES

It works by **layers**. Each round contains 3 layers :

- Key addition layer : a 128-bit round key, or subkey, which has been derived from the main key in the key schedule, is XORed to the state
- Byte Substitution layer (S-Box) Each element of the state is nonlinearly transformed using lookup tables with special mathematical properties. This introduces *confusion* to the data, i.e., it assures that changes in individual state bits propagate quickly across the data path.
- Diffusion layer : it consists of two sublayers, both of which perform linear operations, in order to provide diffusion over all state bits :
  1. *ShiftRows* : permute the data on a **byte** level
  2. *MixColumns* : matrix multiplication which combines, mixes, blocks of 4 bytes $(s_1, ..., s_4)$. Here, if one of the values change, the hole state changes.

Block ciphers | Rijndael and AES

Rijndael data path

Input $x \in \mathbb{Z}_2^{128}$ and $k \in \mathbb{Z}_2^{128\text{ (or 192 or 256)}}$

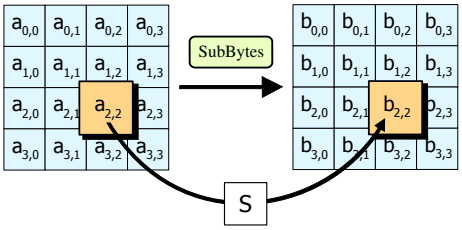
- Key schedule: $(K_i) \leftarrow \texttt{KeyExpansion}(k)$
- state $\leftarrow x$ (but represented as $\mathrm{GF}(2^8)^{4\times 4}$)
- $\texttt{AddRoundKey}(\text{state}, K_0)$
- For each round $i = 1$ to 9 (or 11 or 13):
  - $\texttt{SubBytes}(\text{state})$
  - $\texttt{ShiftRows}(\text{state})$
  - $\texttt{MixColumns}(\text{state})$
  - $\texttt{AddRoundKey}(\text{state}, K_i)$
- And for the last round:
  - $\texttt{SubBytes}(\text{state})$
  - $\texttt{ShiftRows}(\text{state})$
  - $\texttt{AddRoundKey}(\text{state}, K_{10\text{ (or 12 or 14)}})$

Output $y \leftarrow$ state back in $\mathbb{Z}_2^{128}$

32 / 69

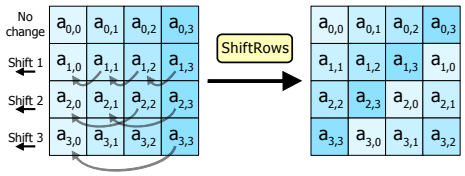
Block ciphers | Rijndael and AES

Rijndael – SubBytes



Block ciphers | Rijndael and AES

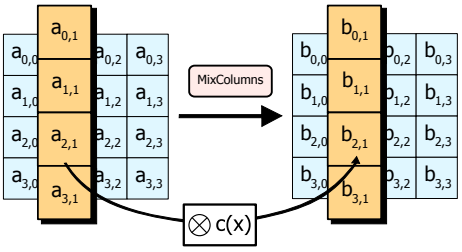
Rijndael – ShiftRows



34 / 69

Block ciphers | Rijndael and AES

Rijndael – MixColumns



37 / 69

36 / 69

Block ciphers | Rijndael and AES

Rijndael – MixColumns

Each column undergoes the following matrix multiplication:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

The inverse of MixColumns uses the inverse matrix:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

38 / 69

Quick chat about Galois Field

In AES the finite field contains 256 elements and is denoted as $GF(2^8)$. This field is chosen because then each element of the field can be represented, in binary, by one byte. This is why in the S-box and MixColumns transforms, AES treats every byte of the internal data path as an element of the field $GF(2^8)$ and manipulates is by performing arithmetics in this finite field.

Inverting AES

As AES is not based on a Feistel network, all layers must be inverted for decryption. We begin by inverting last round, round n_r, then n_{r-1} until round 1.

- We first go through the same key expansion algorithm to get the subkeys K'_i from k if needed (remember, we are dealing with symmetric crypto, encryption and decryption are with the same key!)
- With the ciphertext y as input, we compute XOR with the last key chunk $K_{10,12\text{ or}14}$.
- Then, for each round, we compute the inverse of each step.
  - SubBytes becomes InvSubBytes
  - InvShiftRows
  - InvMixColumns
  - AddRoundKey stays the same (XOR)

2.4.3 Modes of operation

Okay we've seen the famous block ciphers DES, AES, but what to do with them ? As we have already said, it can not constitute an encryption scheme that is IND-CPA, because of their deterministic kind. We must build another architecture using those.

Electronic codebook (ECB)

A first example of architecture using block ciphers is the **electronic codebook** (ECB). It takes as input a message p, a secret key, and a number n which corresponds to the desired block length. So, p is first padded to reach a length that is a multiple of the block size n, and then each block is processed the same way using the block cipher encryption with the key. It is represented at the left figure below.

$$c_i = E_k(p_i)$$

The problem here is of course that every similar block of plaintext is encoded the same way.

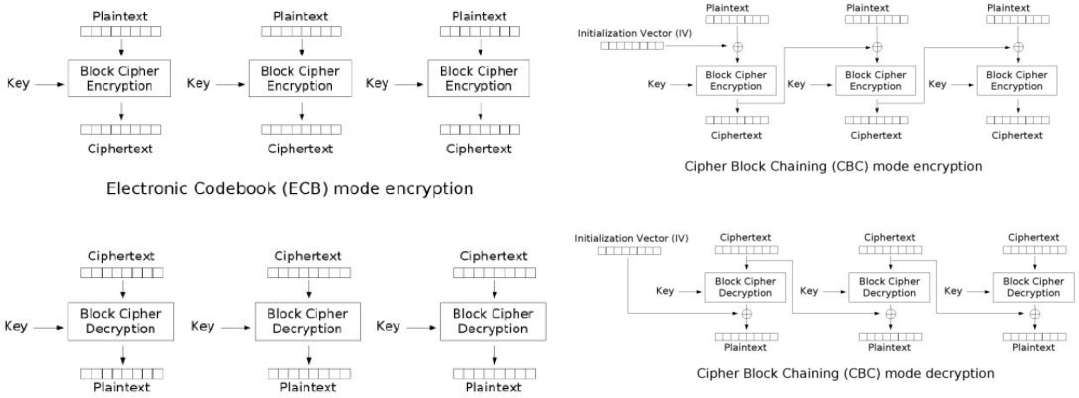
Ciphertext block chaining (CBC)

We take the same scheme as the ECB but at each round, the plaintext block is XORed with a vector before output. The first vector is an initialization vector (IV), and the vector to XOR with p_i is c_{i-1}, nothing but the ciphertext of previous encryption. Note that we here also add **diversification** : the initialization vector is **random**, and it must be a **nonce**. Then :

$$\begin{aligned} c_0 &= E_k(p_0 \oplus \text{IV}) \\ c_i &= E_k(p_i \oplus p_{i-1}) \end{aligned}$$

And to decrypt :

$$p_i = E_k^{-1}(c_i) \oplus c_{i-1}$$



Limits of CBC

What are the limits of the CBC ? Here, we present the probability that for encryption with a same key, $c_i = c'_j$ for different ciphertext blocks.

$$\begin{aligned} c_i &= c'_j \\ E_k(p_i \oplus c_{i-1}) &= E_k(p'_j \oplus c'_{j-1}) && \text{By def.} \\ p_i \oplus c_{i-1} &= p_j \oplus c'_{j-1} && \text{Because } E \text{ invertible} \\ p_i \oplus p'_j &= c_{i-1} \oplus c'_{j-1} \end{aligned}$$

Then, like we saw for the OTP, this means information is revealed on the plaintext ! But how likely is it that $c_i = c'_j$? This is related to the birthday paradox. Indeed, here, the ciphertexts of different blocks and different encryptions (same key, but other diversifier) are supposed to be completely random in respect with each other. Then, the probability of having such a *collision* is left to randomness. This is why it is related to the birthday paradox.

Birthday paradox

With a block size of n, the probability to get a collision after L attempts is

$$\Pr[\text{coll.}] = \frac{L^2}{2^{n+1}}$$

which means that it tends to $1/2$ if

$$L \sim 2^{n/2}$$

.

- For DES, the block size is 64 ($n = 64$), so the number of blocks resulting from the message must be of 2^{32} in order to have a problem. This is a large number, but not really.
- For AES, the block size is 128, which gives us 2^{64} blocks before having a big probability of collision.

The birthday paradox is hence more a problem for DES.

2.4.4 Counter mode

The Counter mode (CTR) is also an encryption mode that uses block cipher as stream cipher.

- Inputs :
  - Secret key $k \in \mathbb{Z}_2^m$
  - Plaintext $p \in \mathbb{Z}_2^*$
  - Diversifier $d \in \mathbb{N}$
- First, we put a target, the desired block size : n.
- Then, we cut our message $p \longrightarrow (p_1, p_2, \ldots, p_x)$. Each block is of n bits, except p_x that does not even require padding.
- We can now begin. The key stream is computed as followed, for i going from 1 to x :
  $$k_i = E_k(d \| i)$$
  The last key chunk, p_x is truncated to the length of p_x
- Encryption of block i :
  $$c_i = k_i \oplus p_i$$
- Output : $c \in \mathbb{Z}_2^*$

To decrypt, we can use the **same keystream** (remember that the nonce is public!) :

$$p_i = k_i \oplus c_i$$

2.4.5 Build MAC from block cipher : CBC-MAC

Aaaah, a bit of authentication ! Yes, block ciphers can also build MACs. Using the CBC architecture, but changing our goals, we can have :

- Inputs :
  - Secret key $k \in \mathbb{Z}_2^m$
  - Message $m \in \mathbb{Z}_2^*$
- Prepend m with its length : $m \Rightarrow |m| \ \|m$
- Pad this with 1 and 0s, cut into blocks $(m_1, m_2, \ldots, m_x)$
- Define $z_0 = 0^n$ and compute
  $$z_i = E_k(m_i \oplus z_{i-1})$$
- Output : MAC is z_x, the last chunk.

As we can see, we only keep the **last chunk** as output, in contrary with classical CBC. Verification goes by doing the same operation at home, with our key k which is the same because of symmetric crypto, and see if we get the same MAC.

2.4.6 Authenticated encryption

As already said, we've seen primitives. Then we can combine them to create a scheme that is secure and provides authentication ! For example, we can encrypt with one key, and compute a MAC with another key (using CBC-MAC for example). Or there are also some modes that provide encryption and authentication with the same key.

2.5 Permutations

Permutations are also primitives, and they are very important. A **permutation is basically** a bijective mapping in $\mathbb{Z}_2^b$:

$$f : \mathbb{Z}_2^b \to \mathbb{Z}_2^b$$

A As it is injective, is has an inverse.

2.5.1 The sponge construction

With the help of permutations, we can build the following architecture, that is called a **sponge function**.

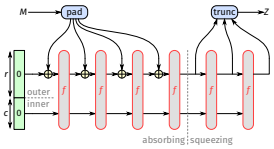
Two parameters, they are natural numbers :

- r : bits of rate
- c : bits of capacity (security parameter)

and of course the permutation f. Together, they form a **sponge function** :

Permutations

The sponge construction



A sponge function implements a mapping from $\mathbb{Z}_2^*$ to $\mathbb{Z}_2^\infty$ (truncated at an arbitrary length)

- $s \leftarrow 0^b$
- $M \| 10^*1$ is cut into r-bit blocks
- For each M_i do (absorbing phase)
  - $s \leftarrow s \oplus (M_i \| 0^c)$
  - $s \leftarrow f(s)$
- As long as output is needed do (squeezing phase)
  - Output the first r bits of s
  - $s \leftarrow f(s)$

55 / 69

The output can be of any length that we want, we just need to add more rounds of f, this is why we see the ∞ symbol.

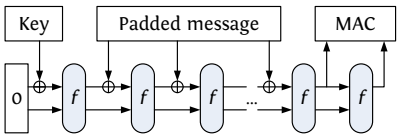
A keyed-sponge structure can be obtained if the message m is prefixed with the secret key K.

Keyed sponge for authentication

Again, here, the goal is not to encrypt the text, but to build a MAC from the key and the message. When Bachar receives the message with the MAC, he has the secret key and can thus compute the same operation at home to verify that no Omar intercepted the message and changed it authenticity.

Permutations

Keyed sponge for authentication



$\text{sponge}_K(M)$ gives a MAC on M under key K

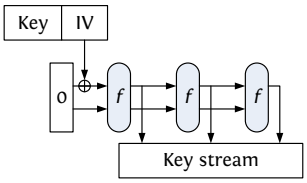
57 / 69

Keyed sponge as a stream cipher

Used here to build an encryption scheme : the sponge function acts as a key stream generator, and the encryption goes as it always do for stream ciphers.

Permutations

Keyed sponge as a stream cipher



$s_i \leftarrow \text{sponge}_K(d)$ produces keystream from key K and diversifier d

$$c_i = m_i + s_i \quad (\text{in } \mathbb{Z}_2)$$

Note: IV = d

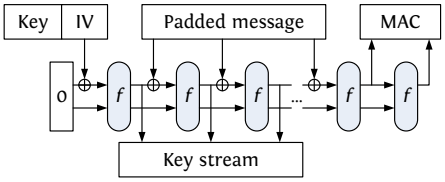
58 / 69

Authenticated encryption : spongeWrap

Ewa c'est quoi la prochaine étape, spongeBob ?

Permutations

Authenticated encryption: spongeWrap



Variant of the sponge construction:

- Keystream: $s_i \leftarrow \text{sponge}_K(d \| \text{previous plaintext blocks})$
- MAC: $\text{sponge}_K(d \| \text{plaintext})$

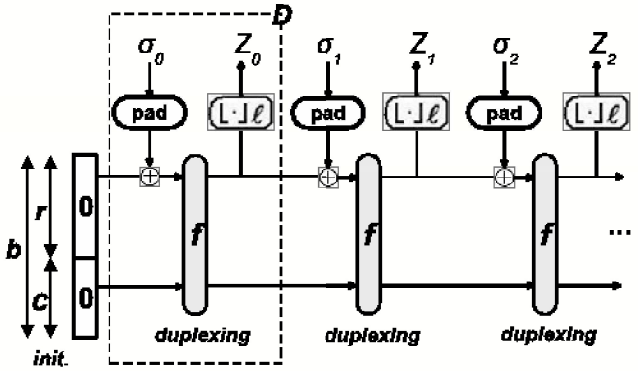
59 / 69

2.5.2 The duplex construction

and sponge constructions in a coherent and rather simple and compact way. Original results are presented in Section 4. Security of authenticated encryption schemes defined for block ciphers and schemes based on the duplex construction is reviewed in Section 4.1. Section 4.2 shows comparison of some key wrapping schemes. The assessment proves that cryptographic schemes based on the duplex construction can be used for protection of the classified information up to the "TOP SECRET" level and unclassified information of different sensitivity levels. Section 4.3 briefly presents the design and evaluation of a cheap but very fast pseudo-random sequences generator based on the duplex construction and a slow random bit generator developed by Military Communication Institute. Finally, Section 5 concludes the paper.

2 The Duplex Construction

The duplex construction (Fig. 1), like the sponge construction [**4**], [**8**], uses a fixed length transformation or permutation f operating on a fixed number b of bits, a padding rule "pad" to build a cryptographic scheme. The duplex construction operates on a state of $b = r + c$ bits. The value b is called the width, the value r is called the bitrate and the value c the capacity. Different values for a bitrate and a capacity give the trade-off between speed and security. The higher bitrate gives the faster cryptographic function that is less secure. It is important that the last c bits of the b-bit state are never directly affected by the input blocks and are never output during the output producing. The capacity c, the most important security parameter, determines the attainable security level of the constructions, as proven in chapter 3. The duplex construction results in an object that accepts calls that take an input string and return an output string that depends on all inputs received so far. An instance of the duplex construction is called a *duplex object* and is denoted by D.



Rysunek 1. The duplex construction

A duplex object D has a state of b bits. Upon initialization all the bits of the state are set to zero. From then on one can send to it D.duplexing (σ, ℓ) calls, with σ as an input string and the requested number of bits. The maximum number of bits ℓ one

2.6 Pseudo-random functions

Chapter 3

Public-key cryptography

3.1 Going public

3.1.1 Symmetric crypto vs public-key crypto

Symmetric cryptography is the oldest kind of cryptography. Indeed, the idea of having a public key is quite disturbing at first glance. Just a quick reminder, here is how we settle a supposed secure way of communication between Ali and Bachar using public-key crypto.

1. Ali and Bachar both generate a key-pair : they keep one private for each, and they publish the other, making it public, so anyone has access to Ali and Bachar's public key.
2. If Bachar wants to send a message to Ali, he **encrypts with Ali's public key** and sends him
3. Ali receives messages that are encrypted using his public key. He decrypts them using his private key.

That's the kind of communication we have for public-key crypto. Symmetric crypto was based on the idea of having **only one key**, kept secret, between A and B.

3.1.2 Why going public ?

There are many problems to symmetric crypto :

- Key distribution problem : the key must be established between Ali and Bachar. How do they do ? The communication link between them is not secure.
- Number of keys : if there are n users, there are $n(n-1)/2$ keys. That's a lot ! 4 million keys for 2000 people.
- No protection against cheating : as Ali has the same secret key as Bachar, he can create a message from scratch and claim that Bachar created it.

In addition to all this, let me talk a little bit, because you guys talk too much,  . Symmetric crypto algorithms are meant to avoid any compact mathematical description. That is easy shit. Asymmetric crypto is **built on one-way number-theoric functions so there is no bullshitting in their security**, you get me ?

3.1.3 Actually going public

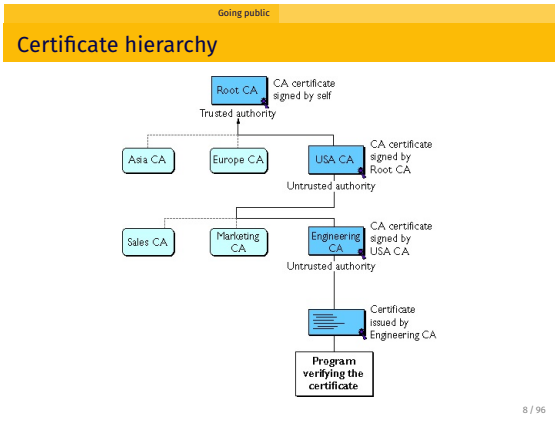
When going public, we are immediately subject to a particular kind of attack : the **Man-In-the-Middle attack** (MIM[1]). In public-key crypto, a crucial part is the exchange of keys : Ali gives his public key to Bachar, so that Bachar can send him messages, and inversely. But there can be an outsider Omar that acts as follows :

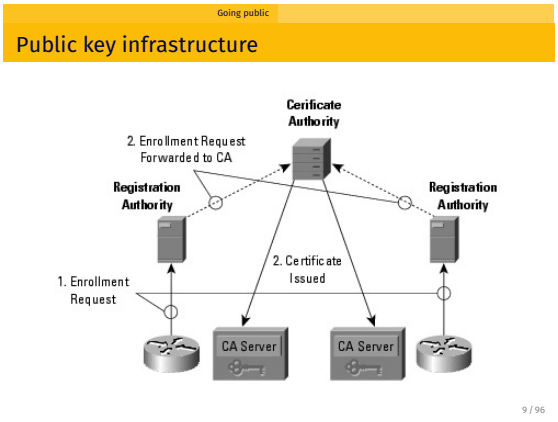
- He captures Ali's public key k_A before Bachar gets it
- He sends to Bachar a false public key, being his own, k_O
- Bachar sends his public key to Ali but it gets intercepted by Omar, that gives to Ali his k_O instead.
- Ali thinks he is receiving messages from Bachar, and inversely, but it is actually the no-life Omar that is intercepting everything and manipulating the conversation.

To avoid this, we make use of **certificates**, objects that bind a public key to an identity. A public key certificate is an object that contains :

- The public key
- Information allowing to identify its owner, such as the name, the ID, IP, etc.
- A digital signature on the key and the information : this is signed by a **certification authority**.

Thanks to this, at the end, Omar can not generate a fake certificate using his key and Ali's information because the signature will be different than Ali's public key certificate, and Bachar will notice that. The certification authority must of course be trustworthy. We can also build hierarchies between CAs as seen below.





Let us recall that we solve only a part of the problem, because Omar can still intercept the communication between Ali and the CA ! Hence, we must have a **secure authenticated channel with the CA**.

[1]Pas MYM, mécréant.

3.1.4 Hybric encryption

Note that in practice, we use **hybrid encryption** to improve both efficiency and bandwith (avoid transmitting a lot of data).

Going public

In practice: hybrid encryption

To improve both efficiency and bandwidth we can combine asymmetric encryption with symmetric encryption.

Alice computes the encryption c of the message m intended for Bob in the following way:

1. Alice chooses randomly the symmetric key k;
2. Alice computes $(c_k, c_m) = \left(\mathsf{Enc}_{\text{Bob's public key}}(k), \mathsf{Enc}_k(m)\right)$.

Bob decrypts as follows:

1. Bob recovers $k = \mathsf{Dec}_{\text{Bob's private key}}(c_k)$
2. Bob recovers $m = \mathsf{Dec}_k(c_m)$

12 / 96

There are many ways of doing public-key crypto, based on different mathematical problems. There is **large integer factorization** ($\rightarrow$ RSA), **discrete logarithm problem** ($\rightarrow$ ElGamal DSA, and DH), and **elliptic curves** ($\rightarrow$ ECDSA)

3.2 RSA : a factorization problem

RSA is mainly about key generation. It is an interessant way of generating keys but is subject to a lot of attacks.

3.2.1 Key generation

The key generation relies on the fact that factorizing large integers is complicated. Hence, it is based on two prime numbers p and q, kept private but **their product n is kept public**, as part of the public key.

The other part of the public key is an exponent that will come in play when encrypting. We note it e. It is public, and satisfies

$$3 \leq e \leq (p-1)(q-1) - 3 \qquad \gcd(e, (p-1)(q-1)) = 1\,.$$

Often, one chooses e and then generates the primes.

The private key is the inverse of e modulo $(p-1)(q-1)$.

Rivest-Shamir-Adleman

RSA key generation

The user generates a public-private key pair as follows:

- Privately generate two large distinct primes p and q
- Choose a public exponent $3 \leq e \leq (p-1)(q-1)-3$
  - it must satisfy $\gcd(e,(p-1)(q-1)) = 1$
  - often, one chooses $e \in \{3, 17, 2^{16}+1\}$ then generates the primes
- Compute the private exponent $d = e^{-1} \bmod (p-1)(q-1)$
- Compute the public modulus $n = pq$ (and discard p and q)

The public key is (n, e) and the private key is (n, d).

26 / 96

3.2.2 RSA textbook encryption

Here is an encryption scheme that stinks a lot : we elevate the message to the power of e to get the ciphertext. To decrypt the ciphertext, we elevate it to the power of d, as e is the inverse of d.

3.2.3 RSA textbook signature

It is basically similar, but here we are **signing** a message with the private key, so $s = m^d \bmod n$. Again, to check the signature, we can elevate s to the power of e to verify if we indeed get the message back (valid signature) or not.

Rivest-Shamir-Adleman

RSA "textbook encryption" (don't use!)

From plaintext $m \in \mathbb{Z}_n$ to ciphertext $c \in \mathbb{Z}_n$ and back:

- **Encryption:** $c = m^e \bmod n$
- **Decryption:** $m = c^d \bmod n$

Why is it correct?

$$\begin{aligned} c^d &\equiv (m^e)^d \equiv m^{ed} \\ &\equiv m^{ed \bmod \Phi(n)} && \text{by Euler's theorem} \\ &\equiv m^{ed \bmod (p-1)(q-1)} && \text{since } p \text{ and } q \text{ are distinct primes} \\ &\equiv m^1 && \text{by definition of } e \text{ and } d \\ &\equiv m \pmod{n} \end{aligned}$$

27 / 96

Rivest-Shamir-Adleman

RSA "textbook signature" (don't use!)

From message $m \in \mathbb{Z}_n$ to signature $s \in \mathbb{Z}_n$ and back:

- **Signature:** Send (m, s), with $s = m^d \bmod n$
- **Verification:** Check $m \stackrel{?}{=} s^e \bmod n$

28 / 96

3.2.4 Attacks on RSA

RSA is subject to many attacks, one of the easiest ones being the **short message attack**, when we know that e is small. We can send short messages so the modular reduction by n does nothing in the encryption. So we can find back m by taking the e-th root of $c = m^e$. One just has to try the different cs.

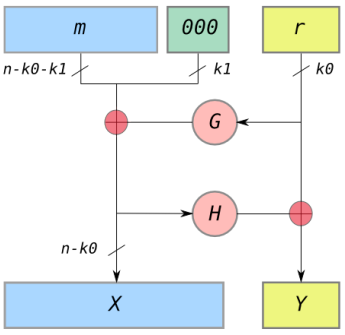
3.2.5 Powering RSA

RSA-OAEP : optimal asymmetric encryption padding

Padding is really interesting. We here add padding to the original message, and randomness to the pre-processing of the message before encryption. Encryption is done as always, exponentiating by e, moduling by n, but the input is a new one : the message after pre-processing.

Rivest-Shamir-Adleman | RSA for confidentiality

RSA-OAEP: Optimal asymmetric encryption padding



[Mihir Bellare and Phillip Rogaway, Eurocrypt 1994]

Encryption: m ... then $c = (X\|Y)^e \bmod n$
Decryption: $(X\|Y) = c^d \bmod n$, then ... m

RSA-OAEP

- makes the scheme probabilistic;
- prevents that an adversary recovers any portion of the plaintext without being able to invert RSA;
- is shown to be IND-CCA secure, assuming that G and H behave as random oracles.

32 / 96

This helps with the short message attack, makes the scheme probabilistic, and introduces diffusion. In addition, it is shown to be IND-CCA secure. Wow !

RSA-KEM : Key Encapsulation Method

Here, we present a method to do **hybrid encryption** using RSA. It is not an encryption scheme that we present ! Remember that in hybrid encryption, Ali somehow chooses the symmetric key and sends it to Bachar in an encrypted way. Here we will have something different :

- Ali chooses a random string m of size n, where n is part of his RSA public key.
- He hashes m : gives $k = \text{hash}(m)$ which will be the secret key
- Ali encrypts $c = m^e \mod n$, and sends c to Bachar

To decrypt (actually, *decapsulate*), Bachar does the following :

- Recovers $m = c^d \mod n$
- Computes $k = \text{hash}(m)$

We thus see that the key k is encapsulated in m, himself encrypted by RSA textbook encryption.

3.2.6 Computational analysis of RSA

Encryption in RSA consists in two operations : exponentiating for encrypting, exponentiating for decrypting. If we choose a small e, it will be fast for encryption. However, there might still have some problems. Below, we list how to improve the computations for RSA :

- Encryption – computing a^e : fast encryption using **square and multiply** (short public exponents)
- Decryption – computing c^d : fast decryption using **Chinese Reminder Theorem** (see next 2 pages)

7.5.2 Fast Decryption with the Chinese Remainder Theorem

We cannot choose a short private key without compromising the security for RSA. If we were to select keys d as short as we did in the case of encryption in the section above, an attacker could simply brute-force all possible numbers up to a given bit length, i.e., 50 bit. But even if the numbers are larger, say 128 bit, there are key recovery attacks. In fact, it can be shown that the private key must have a length of at least $0.3t$ bit, where t is the bit length of the modulus n. In practice, e is often chosen short and d has full bit length. What one does instead is to apply a method which is based on the Chinese Remainder Theorem (CRT). We do not introduce the CRT itself here but merely how it applies to accelerate RSA decryption and signature generation.

Our goal is to perform the exponentiation $x^d \bmod n$ efficiently. First we note that the party who possesses the private key also knows the primes p and q. The basic idea of the CRT is that rather than doing arithmetic with one "long" modulus n, we do two individual exponentiations modulo the two "short" primes p and q. This is a type of transformation arithmetic. Like any transform, there are three steps: transforming into the CRT domain, computation in the CRT domain, and inverse transformation of the result. Those three steps are explained below.

Transformation of the Input into the CRT Domain

We simply reduce the base element x modulo the two factors p and q of the modulus n, and obtain what is called the modular representation of x.

$$x_p \equiv x \bmod p$$
$$x_q \equiv x \bmod q$$

Exponentiation in the CRT Domain

With the reduced versions of x we perform the following two exponentiations:

$$y_p = x_p^{d_p} \bmod p$$
$$y_q = x_q^{d_q} \bmod q$$

where the two new exponents are given by:

$$d_p \equiv d \bmod (p-1)$$
$$d_q \equiv d \bmod (q-1)$$

Note that both exponents in the transform domain, d_p and d_q, are bounded by p and q, respectively. The same holds for the transformed results y_p and y_q. Since the two

primes are in practice chosen to have roughly the same bit length, the two exponents as well as y_p and y_q have about half the bit length of n.

Inverse Transformation into the Problem Domain

The remaining step is now to assemble the final result y from its modular representation (y_p, y_q). This follows from the CRT and can be done as:

$$y \equiv [q\, c_p]\, y_p + [p\, c_q]\, y_q \bmod n \tag{7.7}$$

where the coefficients c_p and c_q are computed as:

$$c_p \equiv q^{-1} \bmod p, \qquad c_q \equiv p^{-1} \bmod q$$

Since the primes change very infrequently for a given RSA implementation, the two expressions in brackets in Eq. (7.7) can be precomputed. After the precomputations, the entire reverse transformation is achieved with merely two modular multiplications and one modular addition.

Before we consider the complexity of RSA with CRT, let's have a look at an example.

Example 7.6. Let the RSA parameters be given by:

$$\begin{aligned} p &= 11 & e &= 7 \\ q &= 13 & d &\equiv e^{-1} \equiv 103 \bmod 120 \\ n &= p \cdot q = 143 \end{aligned}$$

We now compute an RSA decryption for the ciphertext $y = 15$ using the CRT, i.e., the value $y^d = 15^{103} \bmod 143$. In the first step, we compute the modular representation of y:

$$\begin{aligned} y_p &\equiv 15 \equiv 4 \bmod 11 \\ y_p &\equiv 15 \equiv 2 \bmod 13 \end{aligned}$$

In the second step, we perform the exponentiation in the transform domain with the short exponents. These are:

$$\begin{aligned} d_p &\equiv 103 \equiv 3 \bmod 10 \\ d_q &\equiv 103 \equiv 7 \bmod 12 \end{aligned}$$

Here are the exponentiations:

$$\begin{aligned} x_p &\equiv y_p^{d_p} = 4^3 = 64 \equiv 9 \bmod 11 \\ x_q &\equiv y_q^{d_q} = 2^7 = 128 \equiv 11 \bmod 13 \end{aligned}$$

In the last step, we have to compute x from its modular representation (x_p, x_q). For this, we need the coefficients:

3.3 Discrete logarithm problem in $\mathbb{Z}_p^*$

The discrete logarithm problem is also a problem hard to solve, like the factorization of large integers. Here, we focus on the definition of a **group**, that we dont recall in this document. To enter this section, we need ourselves a group :

- A set of items : it will be $\mathbb{Z}_p^*$. It is a set of strings where each character can take values from 0 to $p-1$.
- A composition relation : we choose modular addition
- Identity : 0
- Each item has an inverse

Discrete logarithm problem in $\mathbb{Z}_p^*$

What is the discrete logarithm problem?

Domain parameters:

- Let p be a large prime.
- Let g be a generator of $\mathbb{Z}_p^*$, i.e., $\{g^i \bmod p \ : \ i \in \mathbb{N}\} = \mathbb{Z}_p^*$.

Discrete logarithm problem (DLP)

Given $A = g^a \bmod p$, find a.

There are currently no known polynomial time algorithm to solve this problem.

For 128-bit security, NIST recommends p to be at least 3072-bit long.
[NIST SP 800-57, see also https://www.keylength.com/]

41 / 96

3.3.1 Key generation

Discrete logarithm problem in $\mathbb{Z}_p^*$

Key generation

Domain parameters (public, common to all users):

- Let p be a large prime.
- Let g be a generator of $\mathbb{Z}_p^*$, i.e., $\{g^i \bmod p \ : \ i \in \mathbb{N}\} = \mathbb{Z}_p^*$.

The user generates a public-private key pair as follows:

- Privately choose a random integer $a \in [1, p-2]$
- Compute $A = g^a \bmod p$

The public key is A and the private key is a.

42 / 96

Discrete logarithm problem in $\mathbb{Z}_p^*$

Key generation (group notation)

Domain parameters (public, common to all users):

- Let G be a group.
- Let $g \in G$ be a generator of G of order $q = |G|$.

The user generates a public-private key pair as follows:

- Privately choose a random integer $a \in [1, q-1]$
- Compute $A = [a]g$

The public key is A and the private key is a.

43 / 96

3.3.2 ElGamal encryption

This is an encryption scheme, as stated in the name. We come armed with the key generated like at previous section. We are now ready to fight. We have a message $m \in \mathbb{Z}_p^*$ that we want to send to Ali. We need his public key, he has generated it before, it is $A = g^a \mod p$.

- **Encryption** : he uses the generated key as a multiplicative mask with an exponent. Typically :

$$c = mA^k \mod p$$

  where k is a random integer $\in [1, p-2]$ [2]. k is not shared. However, Ali sends $K = g^k \mod p$ alongside c.

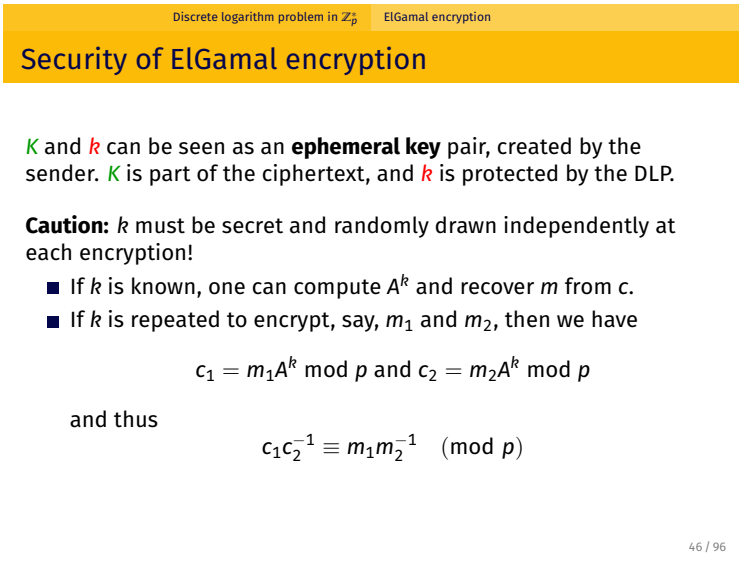
- **Decryption** : Ali receives c and K, and computes

$$m = cK^{-a} \mod p$$

Just a quick reminder here :

- Are public : K, g, A, p
- Are private : a, k

k needs to be ephemeral



Now, let's have a talk about the security of ElGamal encryption scheme.

[2] Note here the bounds : for exponents, it is often $[1, p-2]$

3.3.3 The Diffie-Hellman problem

The Diffie-Hellman problem

Domain parameters:

- Let p be a large prime.
- Let g be a generator of $\mathbb{Z}_p^*$, i.e., $\{g^i \bmod p \ : \ i \in \mathbb{N}\} = \mathbb{Z}_p^*$.

Diffie-Hellman problem

Given $X = g^x \bmod p$ and $Y = g^y \bmod p$, find $X^y = Y^x = g^{xy} \bmod p$.

It is an easier problem than DLP (breaking DHP does not give you the exponents). However, there are currently no known polynomial time algorithm to solve the DHP either.

The DHP does not give x and y, so it does not solve the DLP. We can see two things here :

- Retrieving the secret key a or k from A or K is DLP
- Breaking ElGamal requires to compute g^{ak}, so this is DHP.