A Practical Lecture on IT Security

Jean-François COUCHOT
couchot[arobase]femto-st[point]fr

April 3, 2018

Contents

Chapter 1

Introduction to Security

DEFINITION 1.1. Security of systems concerns the whole set of technical, organisational, juridic and human based ways that are used to keep, to establish to guarantee that approaches aiming at collecting, saving, treating and publishing data are safe.

Particularly, it is interesting to preserve the following properties:

- the *integrity* of information: data are not modified;
- the *confidentiality* of information: it is available only to authorized individuals;
- the *availability*: the information must be available when it is needed.

Other aspects are often considered as objectives of security like:

- the *authenticity*: authentication is the process of actually providing the identity of the information origin;
- the *tracking*: all the access attempt are logged.

This chapter is inspired from [AJOP15, Ghe13].

1.1 Basic concepts

1.1.1 Integrity

The integrity of an information is established if one can prove that it has not been modified, intentionally, or not. For instance, during a file exchange, it is essential that the received file is the one that has been sent: there is no difference between them. *Hash functions* (detailed in chapter 2) are used in such an objective.

1.1.2 Confidentiality

In information security, *confidentiality* is the property, that information is not made available or disclosed to unauthorized individuals, entities, or processes. Such a property may be governed by laws in some cases: medical records, banking data, personal data... It is often the first property we have in mind when we think about security. Tools that allow to obtain such a property are cryptosystems. Examples of such tools are detailed in chapter 3.

1.1.3 Availability

System availability is evaluated thanks to a metric that expresses the percent of time during the system is able to exactly do the awaited task. This is usually measured in terms of nines: for instance, five-9's (99.999%) means less than 5 minutes when the system is not operating correctly over the span of one year. We won't study this property.

1.1.4 Authenticity

Authenticity is assurance that a message, transaction, or other exchange of information is from the source it claims to be from. Authenticity involves proof of identity of the origin. For instance, one can add an indelible mark inside the information. This method further denoted as *watermarking* is presented in chapter 4.

Authentication is a particular property of authenticity: in this case, one have to ensure that identifiers are authentic. This property is often proven thanks to digital signature which are based on cryptosystems. This approach is presented in chapter 3.

Exercise 1.1. *You have to work on a project aiming at storing medical items in a database. Such database should be accessible to medical staff of an hospital. Can you explain which security concepts are linked to this project?*

1.2 Attack through Internet

1.2.1 Cyber-attack

It is not possible to provide an exhaustive panel of all the types of cyber-attacks. However each cyber-attack follows the next outline:

1. Data collect: security implementation approaches (like identification, authentication, cryptography) are studied. Technical or human weaknesses are furthermore analyzed.
2. re-use of existing attack tools or;
3. computation of new tools;
4. system is under attack!
5. exfiltration to mask the attack (removing all traces allowing to identify the attacker).

1.2.2 Actives or Passive Attacks

An attack that modifies data is said to be an *active* attack. When an attack only spies behaviors, it is a *passive* attack.

Exercise 1.2. *Why the opening of information systems by telecommunication networks is a security problems?*

Exercise 1.3. *What did change with the attacks carried out against computer science infrastructure?*

Chapter 2

Integrity and Hash Functions

In its french version this chapter is inspired from [Ver15, SCE08].

2.1 Introduction

A hash function is an algorithm aiming at characterizing an information. From each input data d, it produces a digest which allows to identify this data d. Moreover, the approach is reproducible. What follows gives an example of executing the sha256 hash function on two close messages.

```
$ python xplhash.py
Please give your text 1: ISIFC security
31dfb5202e384fa22709a1435be4819509bfcd4089d2989f8cf1898cef0ce0a5
Please give your text 2: ISIFC seburity
1fb6bd2385da6b21578dcb459119491ec3c540bf5c4d145914f2aa1f1bd18f7f
```

We first can see that the digest has fixed length: (64 hexadecimal numbers, *i.e.*, 64 * 4 = 256 bits). Next, it is obvious to see that a small modification of message ("c" is replace by "b") produces large variations. Such a function is relevant when data integrity is necessary.

A *cryptographic* hash function H has to ensure the following constraints:

- *Avalanche effect*: when an input is slightly changed (for example, flipping a single bit) the output changes significantly; in the previous example, you can notice that "c" is changed to "b". Only one bit is changed. And the result is completely different.
- *Pre-image resistance.* Given a hash value h, it should be very difficult to find any message m such that would lead such a digest (*i.e.*, $h = H(m)$);
- *Second pre-image resistance.* Given an input m_1, it should be difficult to find different input m_2 that would have the same digest than m_1 (*i.e.*, s. t. $H(m_1) = H(m_2)$). If this property is not established, it is possible to generate a false data that would be valid for the user.
- *Collision resistance.* It should be difficult to find two different messages m_1 and m_2 such that $H(m_1) = H(m_2)$. Such a pair is called a cryptographic hash collision.

Practical Work 2.1. *The* `hashlib` *python module contains a family of hash functions: md5, sha1, sha224, sha256, sha384 and sha512.*

1. *Explain the following code*

```
# coding: utf-8
import hashlib
txt1 = input('Please give your text 1: ')
hash_object_1 = hashlib.sha256(txt1.encode('utf-8'))
hex_dig_1 = hash_object_1.hexdigest()
print(hex_dig_1)
```

2. *For each hash method of the* `hashlib` *module, calculate the fingerprint of the message 'Hello World'.*
3. *Develop the program that:*
   (a) *Asks the user to enter a password;*

(b) Memorizes the fingerprint of this password by using sha256;

(c) Asks the user to re-enter his/her password;

(d) If the fingerprints are equal, the program thanks the user; in the opposite case it stops with an error message.

2.2 Resistance to collisions

One can show that the resistance to collision does not imply resistance to the pre-image.

Example 2.1. *Let $g : \{0,1\}^* \to \{0,1\}^n$ be a function which is resistant to collisions. We build $h : \{0,1\}^* \to \{0,1\}^{n+1}$ defined by*

$$h(x) = \begin{cases} 0x \text{ if } x \text{ is of length } n. \\ 1g(x) \text{ otherwise} \end{cases}$$

Let us show that h is resistant to collisions. Let x and x' two sequences of bits such as $h(x) = h(x')$. The two strings are not of size n: in this case we would have $0x = 0x'$, i.e., $x = x'$. Thus if $h(x) = h(x')$, then $g(x) = g(x')$ and x would collide with x' for g, which is resistant by assumption.

Show that h is not resistant to the pre-image. If the fingerprint is $0h_1h_2 \dots h_n$ (i.e., starts with 0), we know immediately how to find a data message which admits such a mark: $h_1h_2 \dots h_n$.

We know today that the MD5 hash function is not collision resistant.

Practical Work 2.2. *Which conclusion(s) can be inferred from the following program?*

```
import hashlib
from array import array

# normal behavior
a1 = array('I',[123,456])
a2 = array('I',[124,456])
print(hashlib.md5(a1).hexdigest())
print(hashlib.md5(a2).hexdigest())


# collision detected
input1 = array('I',  [1634021390, 2275908181, 4156951504, 889109771, 1694748420, 2236575902, 4212113460,
                      2269944933, 798280768, 362884587, 1544942755, 1225230011, 1835370099, 2754879357,
                      2413254944, 4023487834])

input2 = array('I',  [1634021390, 2275908181, 4156951504, 889109771, 1694748420, 2236576926, 4212113460,
                      2269944933, 798280768, 362884587, 3692426403, 1225230011, 1835370099, 2754879357,
                      2413254944, 4023487834])

print(input1[5]+" vs "+ input2[5])
print(input1[10]+" vs "+ input2[10])

print(hashlib.md5(input1).hexdigest())
print(hashlib.md5(input2).hexdigest())
```

When trying to achieve a collision, arises the question of how many fingerprints it is sufficient to calculate to detect a collision. If the fingerprint is a word of n bits, taking $2^n + 1$ different data, will necessarily lead to a collision. However, this is not practically possible to perform if n is large: for example, let $n = 64$ bits, it would take $2^{64} + 1$ operations knowing that since the creation of the universe, about 2^{70} ms have passed. The next section shows how to reduce the number of experiments to perform.

2.3 Birthday attack

The next proposal greatly reduces the number of fingerprints to build to check whether a hash function is resistant to collisions.

PROPOSITION 2.1. If k different data are selected in $\{0,1\}^*$, with

$$k \geq \frac{1 + \sqrt{1 + 2^{n+3} \ln 2}}{2} \tag{2.1}$$

then the probability that two fingerprints are equal is superior to $\frac{1}{2}$.

It is assumed that there are m possible fingerprints. In the case of a fingerprint of n bits, it leads to no more than $m = 2^n$. Let p_k the probability that two data among k yield the same fingerprint. Let $q_k = 1 - p_k$ the probability that the k data generate all a different fingerprint.

For:

- $q_1 = 1$;
- $q_2 = 1(1 - \frac{1}{m})$;
- $q_3 = 1(1 - \frac{1}{m})(1 - \frac{2}{m})$;
- $\ldots$
- $q_k = \Pi_{i=1}^{k-1}(1 - \frac{i}{m})$

Since $1 + x \leq e^x$ for each real number x. The previous formula gives

$$q_k \leq \Pi_{i=1}^{k-1} e^{-\frac{i}{m}} = e^{-\sum_{i=1}^{k-1} \frac{i}{m}} = e^{-\frac{k(k-1)}{2m}}.$$

Thus, we successively have

$$\begin{aligned}
& 1 - e^{-\frac{k(k-1)}{2m}} \geq \frac{1}{2} \\
\Leftrightarrow \quad & e^{-\frac{k(k-1)}{2m}} \leq \frac{1}{2} \\
\Leftrightarrow \quad & -\frac{k(k-1)}{2m} \leq -\ln 2 \\
\Leftrightarrow \quad & k^2 - k - 2m \ln 2 \geq 0 \\
\Leftrightarrow \quad & k \geq \frac{1 + \sqrt{1 + 8m \ln 2}}{2} \\
\Leftrightarrow \quad & k \geq f(n) = \frac{1 + \sqrt{1 + 2^{n+3} \ln 2}}{2}
\end{aligned}$$

For a given number n of bits, if we generate all the words of length $\log_2(f(n))$ bits, the probability to obtain a collision is larger than 1/2. The following array gives the value of $\log_2(f(n))$ for some n.

n	50	100	150	200
$\log_2(f(n))$	$25,24$	$50,24$	$75,24$	$100,24$

We can conclude that if we build $2^{\frac{n}{2}}$ fingerprints, we have about one chance over two to find a collision. Today, it is forbidden to use either MD5 or SHA1. Hash functions that can be used are:

- SHA2 (224, 256, 384)
- SHA3

Chapter 3

Confidentiality and Cryptography

In its french version, this document is mainly inspired from [CR09, Ver15]. The presentation of the digital signature principle is principally extracted https://en.wikipedia.org/wiki/Digital_signature.

3.1 Introduction

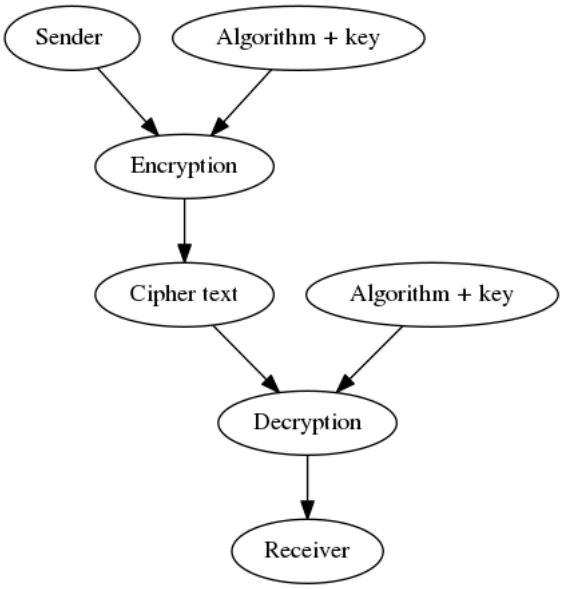


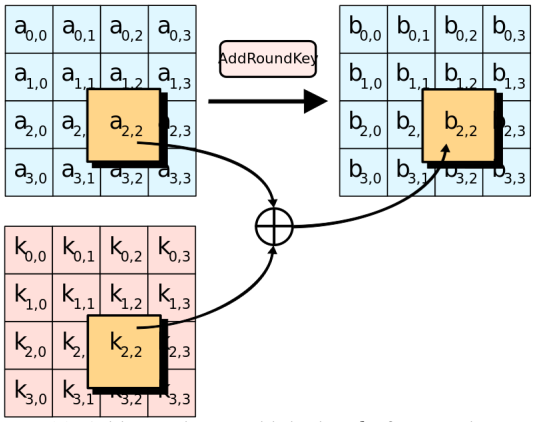
Figure 3.1: General outline of a method of encryption/decryption

Cryptography, which is the art of writing with a key, appeared at the same time as writing. Once information must be transmitted in a safe manner, the message must be protected from interception: it is encrypted by the transmitter and decrypted by the receiver. In the case where one uses an encryption key, we have the diagram shown in Figure 3.1. However in this figure, nothing indicates that the encryption key is the same as that of decryption.

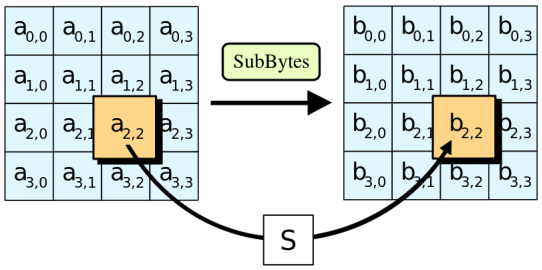
A method that relies on a single key to encrypt and to decrypt a message is called *symmetric cryptography*. The problem of confidentiality of the key and the implementation of this property arises immediately when the number of recipients is large: one key is required for each recipient. To solve this key exchange problem, *asymmetric cryptography* has been developed over the years 1970. It is based on the principle of a public key that is broadcast and a private key that is guarded by the initiator of the exchange. However, if this solves the key exchange problem, it has a cost: slowness. Also, this type of cryptography is mainly used for the exchange of keys and electronic signature. TLS, a protocol used by browsers whenever a connection must be secured, exploits the two families of cryptography, symmetric and asymmetric.

3.2 Symmetric encryption: AES

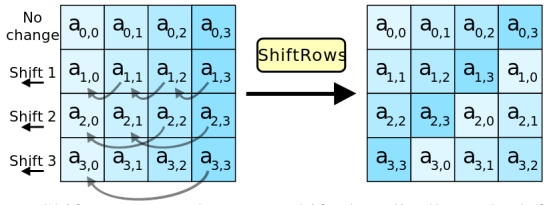
Advanced Encryption Standard (AES) was born in 2001. It is the most symmetric cryptographic algorithm used and the safest today. It is a 128-bit block encryption algorithm. The keys may be variable size (128, 192 or 256 bits).



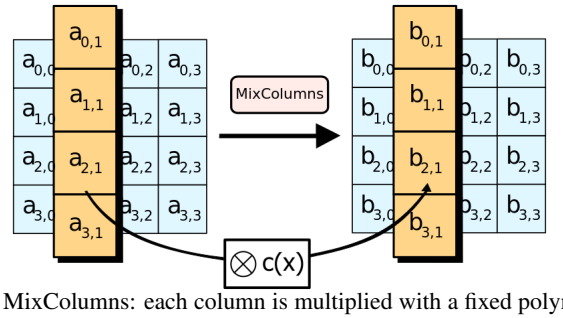
(a) AddRoundKey: add the key k_i for round i



(b) SubBytes: each byte is replaced with one entry



(c) ShiftRows: row bytes are shifted cyclically to the left



(d) MixColumns: each column is multiplied with a fixed polynomial

Figure 3.2: High-level description of AES (Wikipedia)

3.2.1 Operations of AES

In the following, let be given a 128-bit key k and we have to encrypt a message block that has 128 bits as well. Coded in bytes, each block contains 16 elements organized in the form of a 4×4 array. The encryption algorithm AES therefore takes as input two arrays a and k of size 4×4 and has mainly 4 major steps:

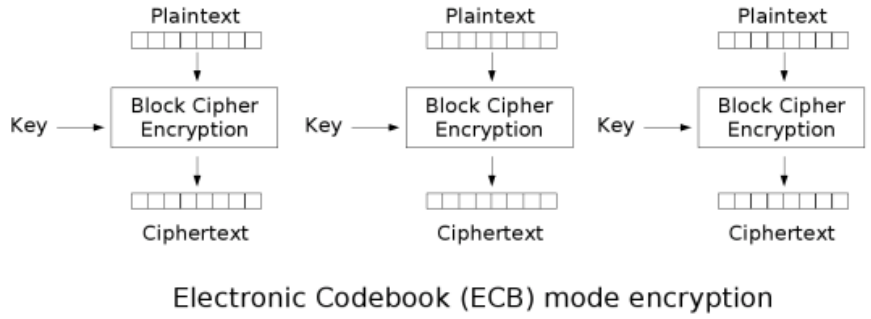
1. Generation of the 11 keys $k^0, k^1, \ldots, k^{10}$ from k. Each of them is a 4×4 size array of bytes.
2. Initial stage. For each line i and column j, $0 \leq i, j \leq 3$, a exclusive OR (XOR) with elements $a_{i,j}$ and $k^0_{i,j}$ is executed. This step is denoted as *AddRoundKey* and is shown in Figure 3.2(a).
3. A set of 9 repetitions. For $t = 1, \ldots, 9, \ldots$
   (a) *Substitution of bytes* (SubBytes) by applying a function S to each of the 16 array elements. This is shown in Figure 3.2(b).
   (b) *Line shift* (ShiftRows) to the left. The offset is 3 for the lowest line, 2 for the one above and 1 for the second line. This is shown in Figure 3.2(c).
   (c) *Mixing Columns* (MixColumns) by applying a linear transformation on each column. This is shown in Figure 3.2(d).
   (d) *Adding round key* (already shown in Figure 3.2(a)), but with the key k^t here.
4. The last repetition is the same except that it does not contain the mixture of the columns. It is therefore made of :
   (a) Substitution of bytes
   (b) Line Offset

(c) Add round key

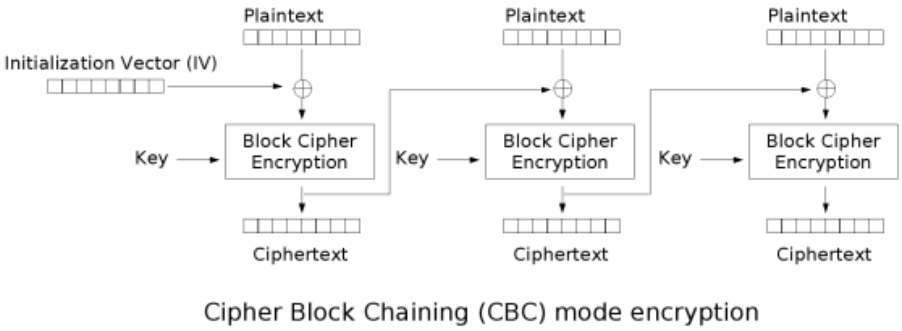
The decryption is similar but takes exactly the inverse functions.

3.2.2 Obtaining 128-bit blocks

It remains to propose methods for obtaining blocks of 128 bits. Two are presented: ECB and CBC.



(a) Electronic codebook (ECB)



(b) Cipher Block Chaining (CBC)

Figure 3.3: Block cipher mode of operation (Wikipedia)

Electronic codebook (ECB). In this procedure, the message is simply split into several blocks of 128 bits which are encrypted separately one after the other. This is represented at Figure 3.3(a). Two blocks with the same content will be encrypted in the same way! An attacker can therefore deduce information from the encrypted text by searching for identical sequences and obtaining a "codebook" establishing the correspondence between the plain text and the encrypted message.

Cipher Block Chaining (CBC). In this procedure, each block executes an "exclusive OR" with encryption of the previous block before it is itself encrypted (see Figure 3.3(b)). The process is initialized with an Initialization Vector (IV).

Exercise 3.1. *1. Explain the following program.*

2. Explain differences between Parts 1 and 2.

3. Run the program.

```
from Crypto.Cipher import AES

message = "The answer is 1.The answer is 2.The answer is 1."

#Part 1
aes_ecb_e = AES.new('This is a key123', AES.MODE_ECB)
ciphertext_aes_ecb = aes_ecb_e.encrypt(message)
```

```
print("ciphertext with ECB: " + str([x for x in ciphertext_aes_ecb]))

aes_ecb_d = AES.new('This is a key123', AES.MODE_ECB)
res=aes_ecb_d.decrypt(ciphertext_aes_ecb)
print(res.decode('utf-8'))

#Part 2
aes_cbc_e = AES.new('This is a key123', AES.MODE_CBC, 'This is an IV456')
ciphertext_aes_cbc = aes_cbc_e.encrypt(message)
print("ciphertext with ECB: " + str([x for x in ciphertext_aes_cbc]))

aes_cbc_d = AES.new('This is a key123', AES.MODE_CBC, 'This is an IV456')
res=aes_cbc_d.decrypt(ciphertext_aes_cbc)
print(res.decode('utf-8'))
```

Practical Work 3.1. *1. Encrypt and thus decrypt a string of your choice with AES and ECB.*

2. *Encrypt an image with AES and ECB. Display the encrypted image. What do you notice? You can use the following example code to manipulate images:*

```
from Crypto.Cipher import AES
from PIL import Image

im = Image.open("imgISIFC.png")
bytes_in_image = im.tobytes()
imb= Image.frombytes(im.mode,im.size,bytes_in_image)
imb.show()
```

3. *Encrypt an image with AES and CBC. Display the encrypted image. What do you notice?*

3.3 Asymetric Encryption: RSA

First, some elements of arithmetic are recalled.

3.3.1 Arithmetic Reminders

Given two integers a and b in $\mathbb{Z}$. We say that a divides b (which is denoted $a|b$) if there is an integer $q \in \mathbb{Z}$ such that $b = aq$.

3.3.2 Greatest common divisor

The greatest common divisor (GCD) of a and b denoted as $\gcd(a, b)$ is the natural number that verifies:

- $\gcd(a, b)|a$ and $\gcd(a, b)|b$;
- If $d|a$ and $d|b$, thus $d|\gcd(a, b)$.

Exercise 3.2. *Compute* $\gcd(550, 1540)$.

3.3.3 Euclide Algorithm

By definition, the GCD of a non-zero a with 0 is a (reasonable definition, because 0 is divisible by any non-zero number a, which is also divisible by a). Finally the GCD of 0 and 0 is not defined. Here we limit ourselves to the case of two integers a and b strictly positive. For example, assume $a > b$.

1. The Euclidean division of a by b can be written as $a = bq + r$ with $0 \leq r < b$.
2. Let us show that "d is a common divisor of a and b" is equivalent to "d is a common divisor of b and r".
   - Let b be of a common divisor of a and b, which can be written $a = da'$ and and $b = db'$. The equality $a = bq + r$ thus becomes $da' = db'q + r$ or $r = d(a' - b'q)$. Also d is a common divisor of b and r.
   - Conversely, let d be a common divisor of b and r, which may be written as $b = db'$ and $r = dr'$. Equality $a = bq + r$ becomes $a = d(b'q + r')$. So d is a common divisor of a and b.

Thus, the sets of common divisors of a and b on the one hand and b and r on the other hand are identical. Particularly, $\gcd(a,b) = \gcd(b,r)$.

3. If $r = 0$ we have $\gcd(a,b) = \gcd(b,0)$ which is equal to b.
4. Otherwise, r is different from 0, and one can therefore perform the Euclidean division of b by r, which gives a remainder r_1, $0 \leq r_1 < r$ and $\gcd(b,r) = \gcd(r,r_1)$.
5. This algorithm is iterated until a null remainder is obtained, which necessarily occurs because we consider natural numbers and that the sequence of constructed remainders is strictly decreasing. The GCD is then the pre-last remainder (the last non-zero).

Exercise 3.3. *Determine* $\gcd(154,35)$ *by the Euclidean algorithm.*

Exercise 3.4. *Give the code of a program that takes two integers* a *and* b *s.t.* $a > b \geq 0$ *and returns* $\gcd(a,b)$.

PROPOSITION 3.1 (BEZOUT IDENTITY). Consider two positive integers a and b. It exists a pair of integers x and y such that $ax + by = d$, where $d = \gcd(a,b)$.

PROOF. *In the proof of the preceding proposition, we had successively:*

$$\begin{aligned}
a &= b \times q_1 + r_1 && (3.1)\\
b &= r_1 \times q_2 + r_2 &&\\
r_1 &= r_2 \times q_3 + r_3 &&\\
&\vdots &&\\
r_{n-4} &= r_{n-3} \times q_{n-2} + r_{n-2} && (3.2)\\
r_{n-3} &= r_{n-2} \times q_{n-1} + r_{n-1} && (3.3)\\
r_{n-2} &= r_{n-1} \times q_n + r_n && (3.4)\\
r_{n-1} &= r_n \times q_{n+1} + 0 &&
\end{aligned}$$

We know that $\gcd(a,b)$ *is* r_n*, i.e. the last non-zero remainder. We go up the equations one by one starting from Eq. (3.4).*

$$\begin{aligned}
r_n &= r_{n-2} - r_{n-1} \times q_n\\
&= r_{n-2} - (r_{n-3} - r_{n-2} \times q_{n-1}) \times q_n \ (r_{n-1} \text{ is replaced by its expression derived from (3.3)})\\
&= r_{n-2}.(1 + q_{n-1}.q_n) - r_{n-3}.q_n \ \text{(factoring)}\\
&= (r_{n-4} - r_{n-3} \times q_{n-2}).(1 + q_{n-1}.q_n) - r_{n-3}.q_n \ (r_{n-2} \text{ is replaced by its expression derived from (3.2)})\\
&\vdots\\
&= \ldots \ (r_1 \text{ is replaced by its expression derived from (3.1)})\\
&= ax + by
\end{aligned}$$

Exercise 3.5. *Show that there exist* x *and* y *such that* $29x + 72y = 1$ *then find a value for* x *and* y.

DEFINITION 3.1 (COPRIME INTEGERS). Two integers a and b are said to be coprime if the only positive integer that divides both of them is 1, *i.e.*, $\gcd(a,b) = 1$.

Exercise 3.6. *Show that 55 and 21 are coprime.*

3.3.4 RSA algorithm

In an asymmetric cryptographic system, two keys are generated: the one that is public and the one that is private. The public key is broadcast while the one that is private is not.

When transmitting information from a transmitter to a receiver, the receiver generates the keys and disseminates the public key. Each transmitter can encrypt its message and only the receiver will be able to decrypt it.

When it comes to authenticating a sender, it is the sender who generates the keys and who broadcasts the public key. Each receiver will be able to decipher the signature and will thus have the guarantee that it is authentic.

In the following, we place ourselves in the first frame, *i.e.*, the receiver generates the keys to obtain data of anyone.

Step 1: choosing two prime numbers p and q. The receiver chooses two large prime numbers p and q and computes $n = pq$. Then he/she calculates $\varphi(n)$, where $\varphi : \mathbb{N}^* \to \mathbb{N}^*$ is the *Euler function*. that is the number of integers in the set $\{1, 2, ..., n-1\}$ that are coprime with n.

Exercise 3.7. *The recipient chooses $p = 7$, $q = 13$. Construct the set of integers that are coprime with $n = pq$ and deduce that $\varphi(91) = 72$.*

Sep 2: choice of the public key A natural number $e \in \{1, \ldots, \varphi(n)-1\}$ coprime with $\varphi(n)$ is selected. The public key is the pair (e, n). Each sender will use it to encrypt his/her message to the receiver. Encryption is detailed in the fourth step below.

Exercise 3.8. *Show that $(29, 91)$ is a acceptable public key candidate.*

Step 3: Build the private key. The receiver calculates the integer $d \in \{1, \ldots, \varphi(n) - 1\}$ such that the remainder in the Euclidean division of ed by $\varphi(n)$ is 1. This is also denoted as $ed \equiv 1[\varphi(n)]$. The pair (d, n) is the private decryption key. It is secret and allows the receiver to decrypt all received messages that are encrypted with (e, n).

Exercise 3.9. *Find the associated private key.*

Step 4: message encryption. The sender may encrypt any written message in the form of a number m belonging to $\{1, \ldots, n-1\}$ and which is coprime with n. The coded message is the remainder a of dividing by n of m^e. We thus have $m^e \equiv a[n]$, where $a \in \{1, \ldots, n-1\}$.

Exercise 3.10.

1. *Show that the sender can encrypt the message $m = 59$.*
2. *Build the encrypted message a using the public key.*

Step 5: decryption of the message. The receiver has a and its private key (d, n). To decrypt, he/she calculates the remainder when divided by n of a^d (*i.e.*, $a^d[n]$). If no calculation error has been made, it is the m initial message.

Exercise 3.11. *Decrypt the message using the private key.*

3.3.5 The key points of the algorithm

The RSA algorithm relies on several key points encountered successively:

- the generation of two large prime numbers p and q;
- modular arithmetic;
- the Euclidean algorithm of generation of GCD and its corollary of Bézout;
- factorization, which as long as it is not feasible on large numbers, guarantees the security of the encryption process.

Exercise 3.12.

1. *Explain the following program.*
2. *With your neighbor:*
   (a) *generate the public and private keys each;*

(b) transmit to him/her your public key;

(c) encrypt a message with his/her public key and transmit the encrypted message;

(d) decrypt his/her message.

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

# 1
rsa_obj = RSA.generate(2048)
rsa_pub = rsa_obj.publickey()

#2
f = open('mPubKey.pem','w')
st = rsa_pub.exportKey('PEM')
f.write(st.decode('utf-8'))
f.close()

#3
f = open('mPrivKey.pem','w')
st = rsa_obj.exportKey('PEM')
f.write(st.decode('utf-8'))
f.close()

#4
pubKey = RSA.importKey(open('mPubKey.pem').read())
encryptor = PKCS1_OAEP.new(pubKey)
message_to_encrypt = 'To be encrypted'
ciphertext = encryptor.encrypt(message_to_encrypt.encode('utf-8'))

#5
privKey = RSA.importKey(open('mPrivKey.pem').read())
decryptor = PKCS1_OAEP.new(privKey)
decrypted_message = decryptor.decrypt(ciphertext)

print(decrypted_message.decode('utf-8'))
```

3.4 Digital Signature by asymmetric encryption

Digital signature (or electronic signature) is a mechanism for ensuring the *integrity* of an electronic document and to *authenticate* the author, by analogy with the handwritten signature of a paper document. It combines the fingerprint of the document by hash function and asymmetric encryption of the document to guarantee respectively these two properties.

3.4.1 General Principle

Suppose Alice wants to digitally sign a message m.

Setting up.

1. She chooses
   - An asymmetric encryption algorithm (RSA for example), consisting of an encryption function C and a decryption function D;
   - A hash function H (SHA2 for example).

   and informs all persons concerned of her choice.
2. She generates a private key K_{pr} and public key K_{pu} for asymmetric encryption algorithm.
3. She distributes the public key K_{pu} through an unsecured channel and keeps the secrecy of K_{pr}.

Generating the signed message.

1. She produces fingerprint $H(m)$ of the message m with the selected hash function H.
2. She encrypts this fingerprint with encryption function C using her private key K_{pr}. The digital signature of this message is $S = C(K_{pr}, H(m))$.
3. The signed message is the pair (S, m) that can be transmitted.

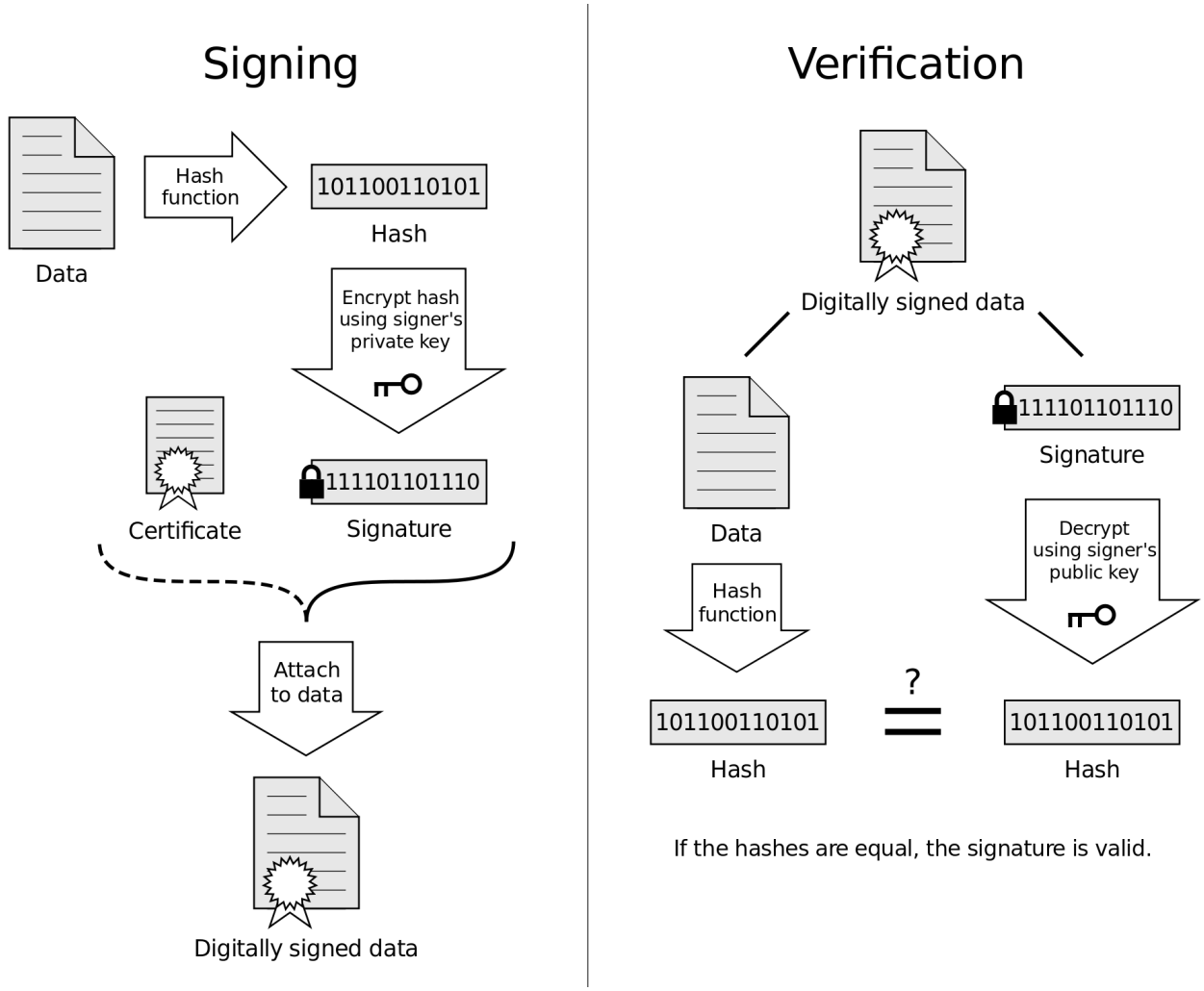


Figure 3.4: General scheme of digital signature and verification thereof (Wikipedia)

Verification of the signed message. Upon receipt of a signed message (S, m), to check its authenticity and integrity:

1. Bob builds a fingerprint $H(m)$ of the message m using the agreed hash function H.
2. Then he decrypts the signature S using the decryption function D with the public key K_{pu}. Let $S'' = D(K_{pu}, S')$.
3. Finally he compares S with $H(m)$: if the two fingerprints are the same, the signature is authentic, the message is authenticated.

Figure 3.4 summarizes this.

Chapter 4

Authenticity and Watermarking

This chapter is inspired by [CMea08, KSMM14]

4.1 Introduction

Digital watermarking is a computational method for inserting information in a digital document. This method should address some of the following properties:

- *Imperceptibility*: in this case, it must be difficult to distinguish the original document with document containing the mark;
- *Fragility*: in this case, a tiny change in the marked paper will make it impossible to extract the mark;
- *Robustness*: in this case, on the contrary, the mark is still present even if the document has been modified;
- *Reversibility*: in this case, it is possible to restore the document in its original form by replacing bits containing the mark with those of the original document.

4.2 Applications of watermarking

4.2.1 Proof of media ownership

Often the photo agencies add a visible mark to freely available versions of their photographs. These mark make it unusable in practice these photos in a professional context. These marked photos are just appeal products the high resolution pictures.

These high-resolution images usually embed an identifier property, but invisibly (by modifying only a few bits). Also, if the watermarking is robust, the agency can next justify to be the owner of the image if this one is traded without his consent, or if it is a slightly modified version that is found (partial, having undergone transformation...). The best-known example in omission of watermarking and support property loss is that of Lena Sjööblom.

4.2.2 Copy control

The watermarking can be used to identify the person to whom a digital document is given. The document proposer adds the person's identification as a mark inside it. This can be traced back to that person if the document should not be released, but it is.

A known example where this has been implemented in the early 2000s. Hollywood studios had found that a vast majority of illegal downloading movies available came from Hollywood industry itself. From that date, each copy distributed to an employee has been marked with a unique identifier. The Carmine Caridi actor received digital versions of films (marked) in order to vote in the Oscars 2004. Part of his films ("Mystic River," "The Last Samurai"...) -containing its watermark- were found on the download networks. This made possible to identify him as one of the authors of this share.

4.2.3 Content integrity

The watermarking can be used to ensure that the document has not been altered. In this context one can think of building the fingerprint of the document (via a hash function for example). If this document is accessed again and if the fingerprint is is the same, there is a very high probability that the document has not been changed.

This raises the issue of storage of this fingerprint. One idea may be to build it from the significant parts of the document and only then insert into its non-significant parts.

4.3 Implementing a robust image watermarking scheme

4.3.1 Naive watermarking by LSB replacement

LSB means *Least Significant Bit*. The mark is inserted into the lower bits. This simple watermarking algorithm holds in a single line:

```
y = [int(bin(x[i])[:-1]+str(mark[i]),2) for i in range(len(x))]
```

In this line,

- `mark` is the list of message bit: `mark=[0,1,0,0...]`;
- `x` is the data list; it is the medium in which the watermark is embedded; its size is the same as that of the mark;
- For each index `i` browsing the entire list `x`, for example `i`= 2;
  - Converting `x[i]` in binary: there is thus a string 45 becomes "101101" for example;
  - When writing `[:-1]`, all of this chain is preserved except for the last bit, "10110" for example;
  - The last bit is added the `i`th value of the mark, 0 in this example;
  - When writing `int(''101100'',2)`, the bit string is converted into integer, here 44.

Exercise 4.1. *Explain this code except the line that has already been presented below.*

```
import random as rd
from PIL import Image

support = Image.open("lena512.png")
support.show()

x = list(support.getdata())
mark=[rd.randint(0,1) for _ in range(len(x))]
y = [int(bin(x[i])[:-1]+str(mark[i]),2) for i in range(len(x))]

watermarked_image = Image.new("L",support.size)
watermarked_image.putdata(y)
watermarked_image.show()
```

Practical Work 4.1.

1. *Implement the code seen in the previous example.*
2. *Watermark the image of your choice with a simple message using the LSB replacement scheme. Extract the mark of the watermarked image.*
3. *Implement an attack that changes at random every non-significant bits. Is it possible to recover the mark after the attack.*

4.3.2 Quantization based Watermarking

Quantization is the process which allows to approach a continuous signal by the values of a finite small set. The underlying idea of the watermarking by quantization is that a mark even in a a watermarked but attacked document can be restored if its value lies in its quantization area. STDM (Spread Transform Dither Modulation) is an algorithm of this family. In a basic version, the watermarking function is parameterized by:

- a variable Δ used in the quantization: the larger the value is, the more robust is the approach, but it may become perceptible;
- A projection vector $(p_1, \ldots, p_n)$ that is used as a key.

Exercise 4.2. *Explain each line of the following code*

```
##################################################
N=10                # size of the  mark
size_x=5000         # size of the support
size_p=int(size_x/N)# taille du vecteur de projection
Delta = 20

x=[50*rd.random() for _ in range(size_x)]
p=[rd.random() for _ in range(size_p)]
normep = math.sqrt(sum([e**2 for e in p]))
p = [pi/normep for pi in p]
m=[rd.randint(0,1) for _ in range(N)]

y     = emb_stdm(x,p,m,Delta)
mp    = dec_stdm(y,p,Delta)
yp    = [yi + rd.gauss(0,1) for yi in y]
mpp   = dec_stdm(yp,p,Delta)

print(y[:15])
print(yp[:15])

print(mp)
print(mpp)
```

Practical Work 4.2. *1. Get the* `stdm_etu.py` *file and run the code from the previous example.*

2. *Watermark the image of your choice with a simple message with STDM. Notice that in a RAW image, each pixel is defined by an integer value while stdm returns float values. Use of functions provided in the file.*
3. *Vary Δ. Does it affect the visual quality of the marked image?*
4. *Extract the mark of the watermarked image.*
5. *Implement an attack that randomly changes every non-significant bits. Show the watermarked attacked image. Can you use in practice? Is it possible to recover the mark after the attack?*
6. *Implement an attack that adds a Gaussian noise with zero mean and a σ standard deviation. Vary σ until this image can be usable.*

4.4 Reversible watermarking

When the processed data is sensitive (military, medical diagnostic ,...) end users can not to allow the information that analysis is not the original. For a radiologist who seeks to detect pathology with a radio image, it is intolerable that the diagnosis is wrong for the patient's health and for the radiologist himself. In this case, the irreversible watermarking (which irreversibly alters the original signal) is not an acceptable solution.

To solve this problem, the reversible watermarking was created: the original content can be restored from the watermarked document. The assessment may well be on the original signal. Thus watermarking step has a positive impact on the initial document as it includes additional data to it. It is important to note that the visual impact of the mark embedding is not as important that in the classic watermarking, since the original document can be restored. We can refer also to [KSMM14] for a comprehensive review of recent algorithms reversible watermarking

Bibliography

[AJOP15] Gildas Avoine, Pascal Junod, Philippe Oechslin, and Sylvain Pasini. *Sécurité informatique, cours et exercices corrigés*. Vuibert, 2015.

[CMea08] Ingemar J. Cox, Matthew L. Miller, and Jeffrey A. Bloom... [et al.]. *Digital watermarking and steganography*. The Morgan Kaufmann series in multimedia information and systems. Morgan Kaufmann Publishers, Burlington (Mass.), 2008.

[CR09] Yvan Saint-Aubin Christiane Rousseau. La cryptographie à cle publique: le code rsa (1978). In *Mathématiques et Technologie*, Springer Undergraduate Texts in Mathematics and Technology, pages 213–244. Springer New York, 2009.

[Ghe13] S. Ghernaouti. *Sécurité informatique et réseaux - 3e éd.: Cours avec plus de 100 exercices corrigés*. Informatique. Dunod, 2013.

[KSMM14] Asifullah Khan, Ayesha Siddiqa, Summuyya Munib, and Sana Ambreen Malik. A recent survey of reversible watermarking techniques. *Information Sciences*, 279:251 – 272, 2014.

[SCE08] SCEI. Travaux 'initiative personnelle encadrés (tipe), les fonctions de hachage. Document en ligne, 2008. https://www.scei-concours.fr/tipe/TIPE_2008/sujets_2008/informatique_2008.pdf.

[Ver15] Damien Vergnaud. *Exercices et problèmes de cryptographie - 2ème édition*. Sciences Sup. Dunod, January 2015.