

# CRYPTOGRAPHIE

Prof. K. EL GHOLAMI

# OBJECTIFS

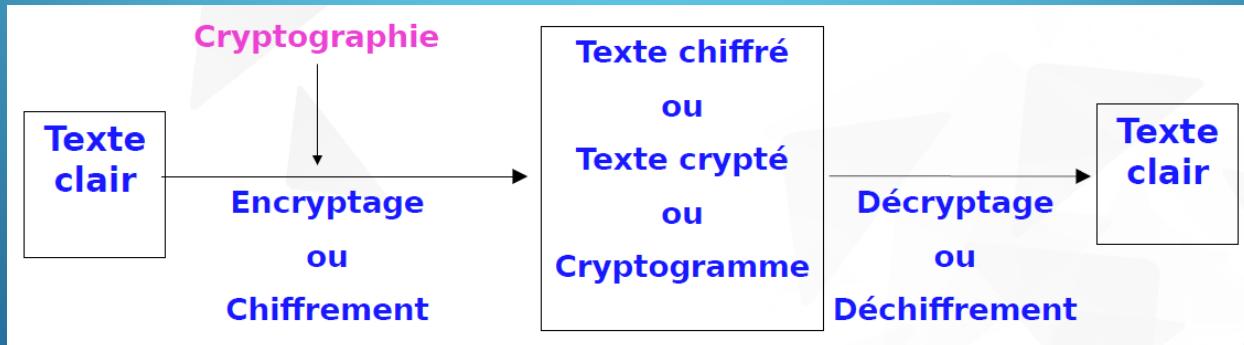
- ▶ **Objectif principal:**
  - ▶ Assurer la sécurité des communications sur un canal non sécurisé
- ▶ **Sécurité des communications?**
  - ▶ Confidentialité, Intégrité, authenticité, non répudiation
- ▶ **Canal non sécurisé?**
  - ▶ Attaques passives: écouter des communications
  - ▶ Attaques actives: contrôler la communication (Man in the middle)
    - ▶ Injection,
    - ▶ Suppression,
    - ▶ modification

# TERMINOLOGIE

- ▶ **Cryptologie = cryptographie + cryptanalyse**
- ▶ **Science (branche des mathématiques) des communications secrètes.**
- ▶ **Composée de deux domaines d'études complémentaires :**
  - ▶ **Cryptographie : conception d'algorithmes/protocoles de cryptage**
  - ▶ **Cryptanalyse : science analysant les cryptogrammes en vue de les décrypter (casser des algorithmes/protocoles de cryptage)**

# TERMINOLOGIE

- ▶ Cryptographie (cryptography) = Chiffrement=Encryptage
- ▶ C'est l'ensemble des méthodes et techniques qui permettent de transformer un message afin de le rendre incompréhensible pour quiconque qui n'est pas doté du moyen de le déchiffrer (inintelligible à autre que qui-de-droit)
  - ▶ On parle d'encrypter (chiffrer) un message,
  - ▶ Le code résultant s'appelle cryptogramme.
  - ▶ L'action inverse s'appelle décryptage (déchiffrement).



# TERMINOLOGIE

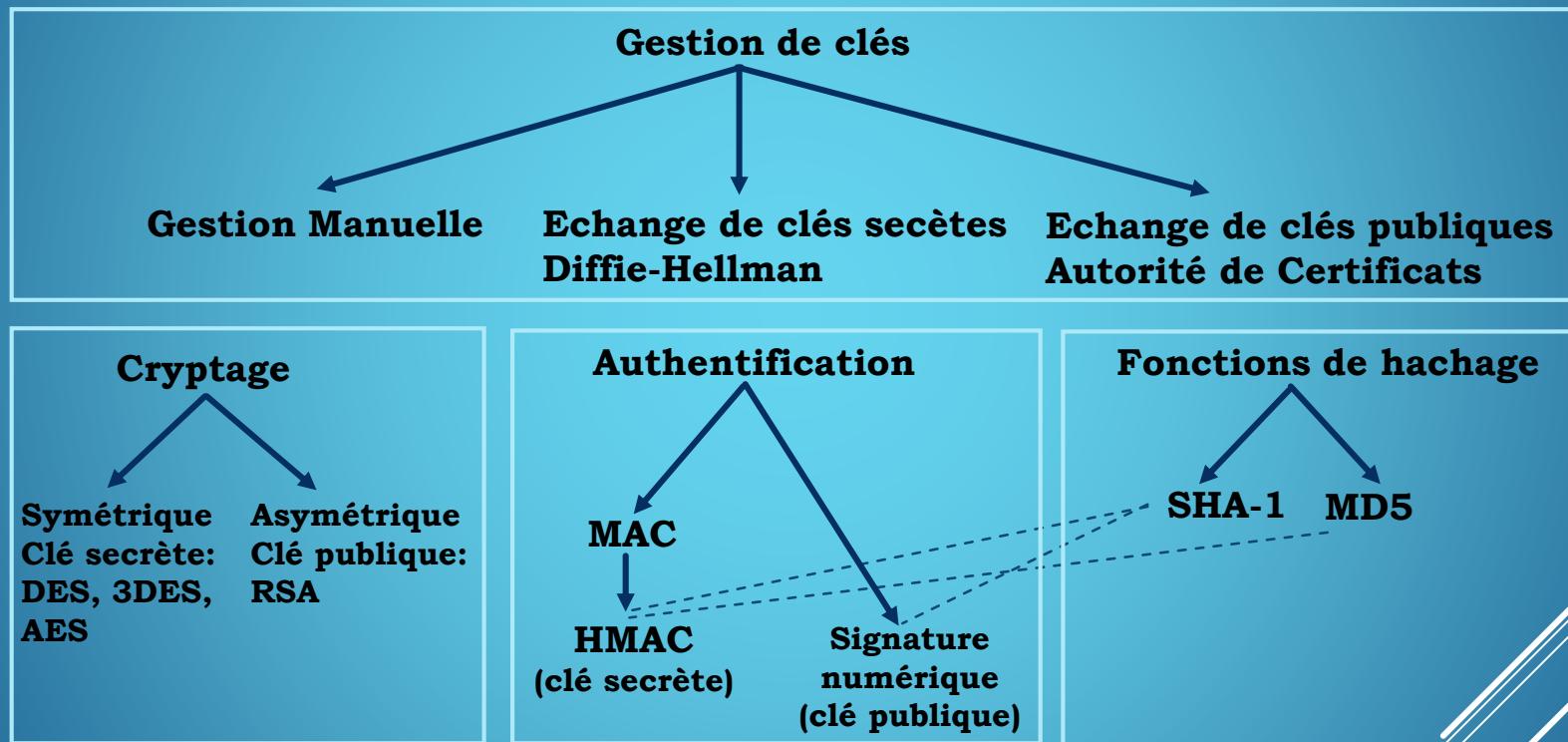
## ► Clé :

- ▶ Information qui sera utilisée pour encrypter et/ou décrypter un message.
- ▶ On peut cependant concevoir un algorithme qui n'utilise pas de clé, dans ce cas c'est lui-même qui constitue le secret et son principe représente la clé

## ► Crypto système: un ensemble composé

- ▶ d'un algorithme,
- ▶ de tous les textes en clair (**espace des textes clairs**),
- ▶ de tous textes chiffrés (**espace des textes chiffrés**)
- ▶ de toutes les clés possibles (**espace des clés**).

# SYSTÈME DE CRYPTAGE



# ALGORITHMES DE CRYPTAGE

## Principe de Kerckhoffs:

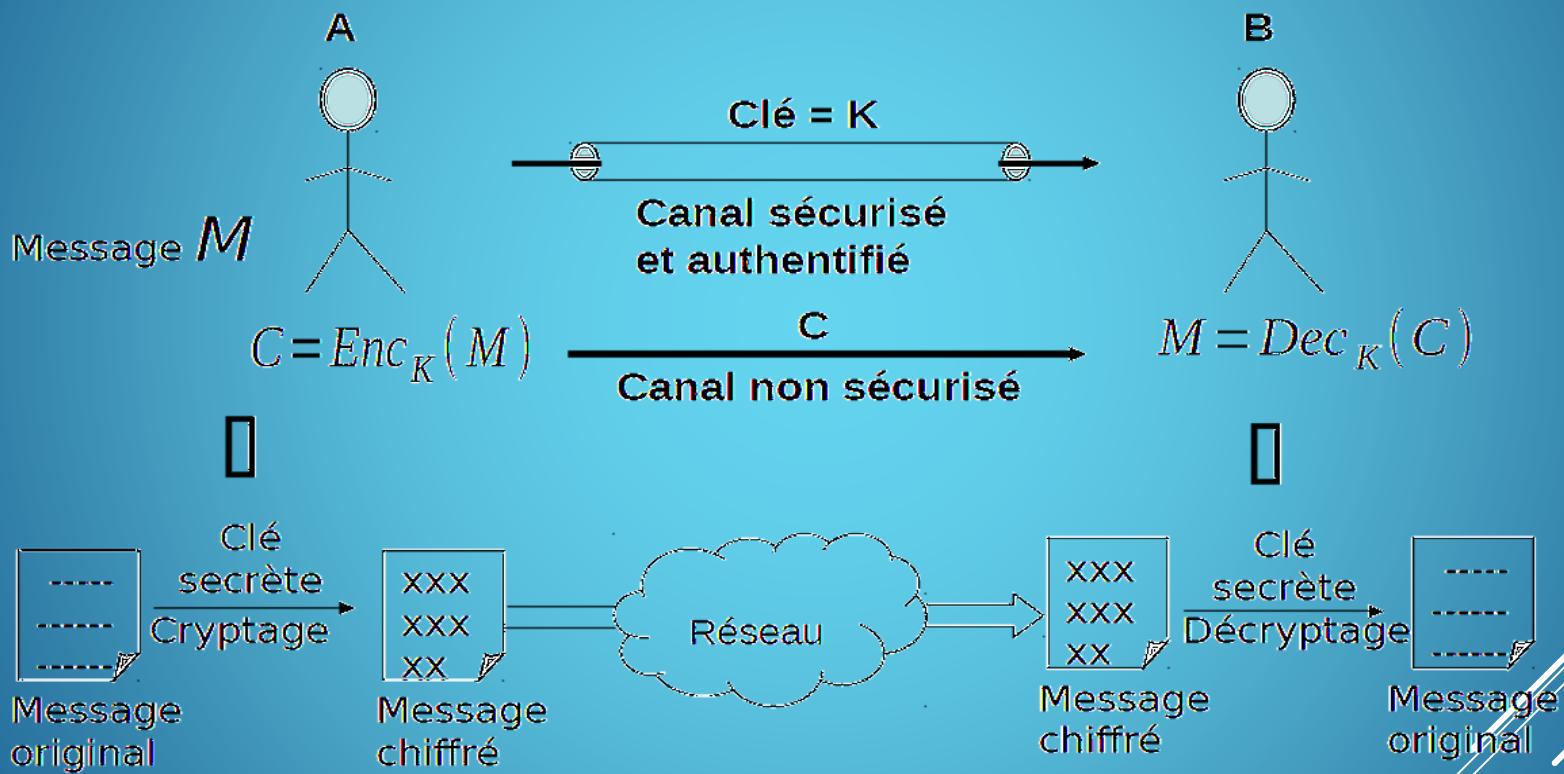
**(La cryptographie militaire 1883)**

*La sécurité d'un système cryptographique ne doit pas reposer sur la non divulgation des fonctions de chiffrement et de déchiffrement utilisées mais sur la non divulgation des clés utilisées pour les paramétrier.*

- ▶ Les algorithmes de cryptage se divisent en deux grandes familles selon la nature de la clé utilisée :
  - ▶ Algorithmes Symétriques (à clé secrète)
  - ▶ Algorithmes Asymétriques (à clé publique)

# ALGORITHMES DE CHIFFREMENT SYMÉTRIQUES

# CRYPTAGE SYMÉTRIQUE



# LES CHIFFREMENTS SYMÉTRIQUES

- ▶ Les chiffrements symétriques sont les héritiers des méthodes anciennes des cryptographies (comme les substitutions, les transpositions ou le chiffre de Vigenère).
- ▶ L'expéditeur et le destinataire disposent chacun d'un algorithme pour respectivement chiffrer et déchiffrer. Ces algorithmes sont inverses l'un de l'autre. Ils dépendent d'une clé que doivent s'échanger l'expéditeur et le destinataire. Le terme "symétrique" vient de cette particularité.
- ▶ C'est la même clé qui sert au chiffrement et au déchiffrement. En particulier, expéditeur et destinataire doivent s'échanger cette clé, qui doit rester secrète sous peine qu'un tiers parvienne à déchiffrer les correspondances. Voilà pourquoi on parle aussi de chiffrement à clé secrète.
- ▶ L'échange des clés secrètes, qui doit se faire par un canal sécurisé, est souvent le point faible de ces méthodes de chiffrement

# EXEMPLES D'ALGORITHMES DE CHIFFREMENT SYMÉTRIQUES

# CHIFFREMENT PAR DÉCALAGE (SHIFT CIPHER)

- ▶ Exemple: espace des clés [0..25]
- ▶ Chiffrement en utilisant K
  - ▶ Chaque lettre du message clair est remplacé par la  $k^{\text{ième}}$  lettre en partant de la lettre à remplacer (Décalage à droite : shift right)
- ▶ Déchiffrement:
  - ▶ Décalage à gauche : shift left.
- ▶ Cryptanalyse:
  - ▶ Il est facile de déterminer K (brute force attack: tester les 26 possibilités)
- ▶ Exemple: chiffrement de cesar

- ▶ On intercepte le message

**FAGEMYREMPURZV\_EMZR\_R FMNMDAZR**

- ▶ Essayons différents décalages...

1: E\_FDLXQDLOTQYUZDLYQZQZELMLC\_YQ

2: DZECKWPCKNSPXTYCKXPYPYDKLKBZXP

3... 4... 5... 6... 7... 8... 9... 10... 11... 12...

13: **TOUS\_LES\_CHEMINS\_MENENT\_A\_ROME**

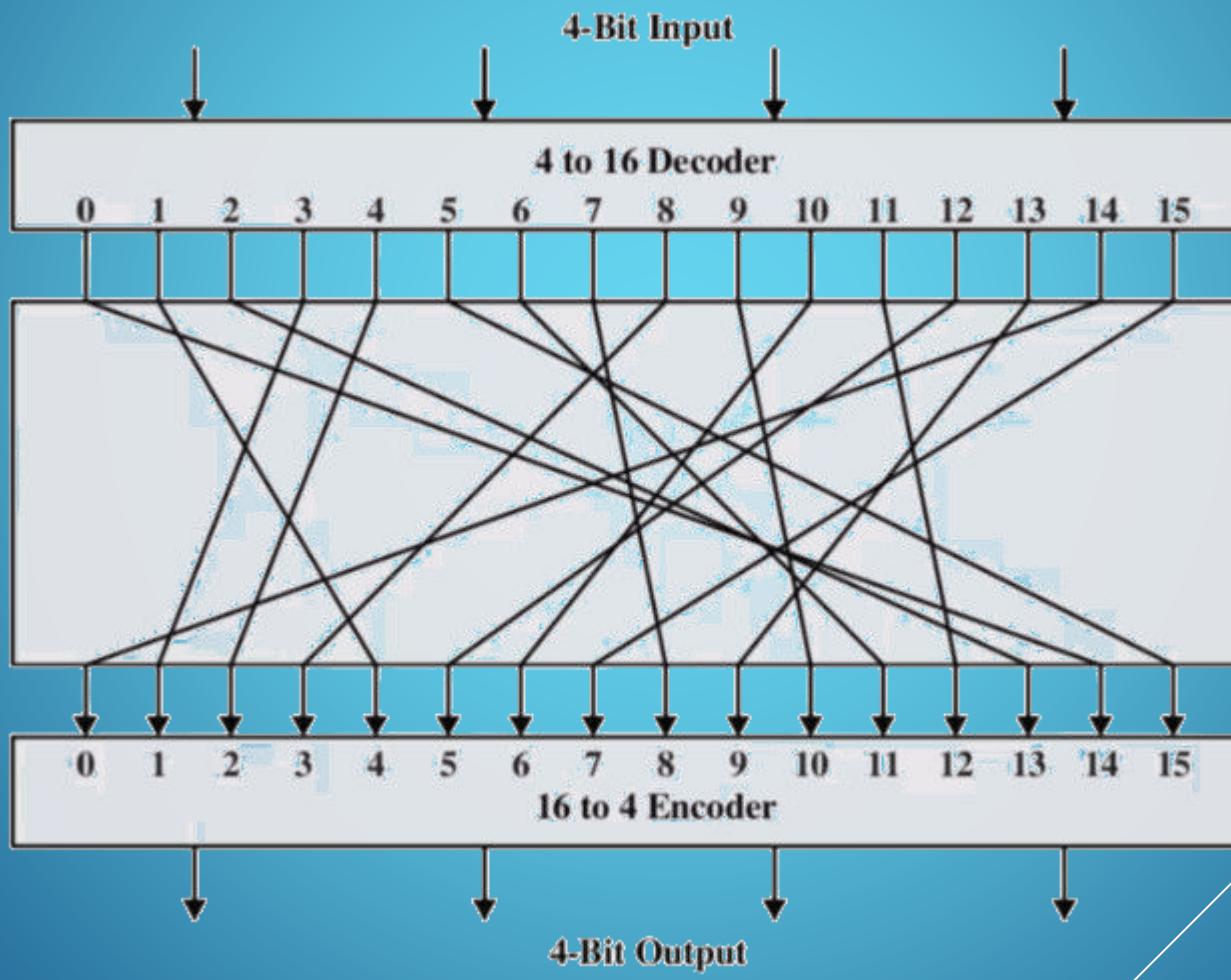
➔ le chiffrement de César n'est pas sécuritaire.

# CHIFFREMENT MONO-ALPHABÉTIQUE PAR SUBSTITUTION

- ▶ Espace des clés:
  - ▶ Exemple: toutes les permutations de  $\Sigma=\{A,B,\dots,Z\}$
- ▶ Chiffrement en utilisant une clé K
  - ▶ Chaque lettre  $X_i$  du message clair est remplacé par la lettre  $K_i(X_i)$
- ▶ Déchiffrement en utilisant la clé K
  - ▶ Chaque lettre  $Y_i$  du message chiffré est remplacé par  $K^{-1}_i(Y_i)$
- ▶ Exemple:
  - ▶ ENSA DE KHOURIBGA → WKNS CW QOFTZVBS

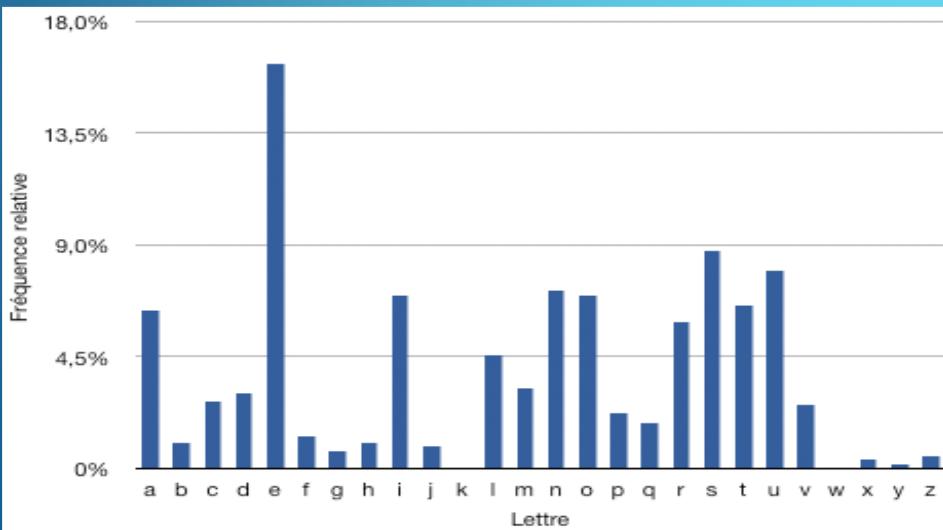
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
S	E	L	C	W	J	B	O	V	X	Q	Y	P	K	F	I	A	Z	N	M	T	R	H	D	U	G
▼	▼	▼	▼	▼	▼	▼	▼	▼	▼	▼	▼	▼	▼	▼	▼	▼	▼	▼	▼	▼	▼	▼	▼	▼	▼

# EXEMPLE DE SUBSTITUTION BINNAIRE



# CHIFFREMENT MONO-ALPHABÉTIQUE PAR SUBSTITUTION: CRYPTANALYSE

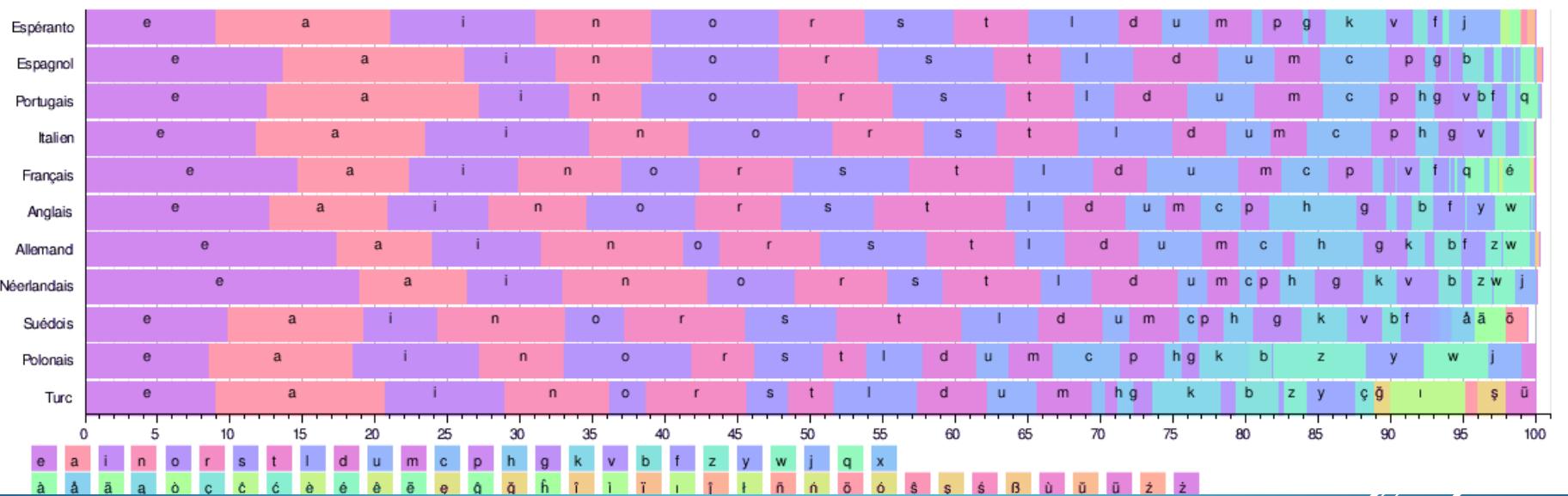
- ▶ La recherche exhaustive est difficile: Espace des clés = (26! clés possibles)
- ▶ Cryptanalyse par analyse fréquentielle: facile
  - ▶ Découverte par les arabes (AI KINDI):
    - ▶ Idée: Chaque langage possède certaines caractéristiques: fréquences des lettres, de groupes de 2 lettres....
    - ▶ insécurité du chiffrement par substitution
  - ▶ Exemples:
- fréquences des lettres en Français



## Fréquence des bigrammes

ES	3.05%	LE	2.22%	DE	2.17%	RE	2.1%	EN	2.08%	ON	1.64%	NT	1.62%	ER	1.53%	TE	1.52%	ET	1.43%	EL	1.42%	AN	1.37%	SE	1.32%
LA	1.29%	AI	1.24%	NE	1.14%	OU	1.12%	QU	1.11%	ME	1.08%	IT	1.06%	IE	1.05%	EM	1.01%	ED	1.01%	UR	1.01%	IS	0.99%	EC	0.95%
UE	0.92%	TI	0.9%	RA	0.86%	NS	0.84%	IN	0.84%	TA	0.82%	CE	0.81%	AR	0.8%	EE	0.79%	EU	0.78%	SA	0.76%	CO	0.74%	EP	0.71%
ND	0.7%	IL	0.7%	SS	0.68%	ST	0.66%	SI	0.65%	TR	0.64%	AL	0.64%	UN	0.63%	PA	0.62%	AU	0.61%	EA	0.6%	AT	0.58%	MA	0.58%
RI	0.58%	SD	0.57%	SO	0.57%	US	0.57%	UI	0.56%	LL	0.53%	NC	0.53%	VE	0.52%	LI	0.51%	RO	0.51%	IO	0.51%	OR	0.5%	PE	0.48%
OI	0.48%	PR	0.47%	PO	0.46%	IR	0.46%	NA	0.45%	UT	0.44%	TD	0.44%	CH	0.44%	OM	0.43%	SP	0.43%	SL	0.42%	DA	0.42%	AS	0.42%
MO	0.41%	AC	0.4%	DI	0.4%	RS	0.39%	DU	0.39%	TL	0.38%	TO	0.38%	TS	0.38%	RT	0.37%	AM	0.37%	AP	0.37%	SC	0.36%	LO	0.36%
AV	0.35%	SU	0.35%	EV	0.34%	NO	0.33%	RL	0.33%	NI	0.32%	GE	0.31%	RD	0.31%	LU	0.31%	NN	0.3%	HE	0.29%	PL	0.28%	IQ	0.28%
EE	0.35%	MI	0.35%	VA	0.34%	TU	0.33%	VI	0.33%	CA	0.32%	FO	0.31%	CI	0.31%	TT	0.31%	IC	0.3%	IU	0.29%	MM	0.28%	OL	0.28%

# DIAGRAMME COMPARATIF DE LA FRÉQUENCE DES LETTRES DANS 11 LANGUES



# DÉFENSE CONTRE L'ANALYSE FRÉQUENTIELLE

- ▶ Utiliser des blocs plus long (64 ou 128 bits) pour la substitution au lieu de faire la substitution lettre par lettre
  - ▶ exemple de chiffrement par bloc: DES, 3DES, AES...
- ▶ Utiliser différents substitution
- ▶ Faire correspondre à chaque caractère du texte en clair un ensemble d'équivalents possibles appelé homophones (généralement des chiffres)
  - ▶ analyse fréquentielle difficile mais possible
- ▶ Chiffrement par substitution poly-alphabétique
- ▶ ...

# CHIFFREMENT PAR SUBSTITUTION POLY-ALPHABÉTIQUE

- ▶ **Faiblesse du chiffrement mono-alphabétique**
  - ▶ Chaque lettre du message chiffré correspond à une seule lettre du message clair
  - ▶ possibilité de l'analyse fréquentielle
- ▶ **Solution: substitution poly-alphabétique**
  - ▶ Utiliser plusieurs alphabets de chiffrement et les utiliser lors du chiffrement de différentes lettres
    - ▶ les fréquences des lettres du cryptogramme sont similaires
- ▶ **Exemple: chiffrement de viginère (1586)**

# CHIFFREMENT DE VIGINÈRE

- ▶ Traiter les lettres comme nombres: {A=0, B=1,...Z=25}
- ▶ Notation  $Z_n = \{0, 1, 2, \dots, (n-1)\}$
- ▶ Définition:
  - ▶ Soit :
    - ▶ P: plaintext, C: Ciphertext
    - ▶ m un entier  $> 0$ ,  $P=C=(Z_{26})$  et  $K=\{k_1, k_2, \dots, k_m\}$
- ▶ Encryptage :  $e_k(p_1, p_2, \dots, p_m) = (p_1+k_1, p_2+k_2, \dots, p_m+k_m) \pmod{26}$
- ▶ Décryptage :  $d_k(c_1, c_2, \dots, c_m) = (c_1-k_1, c_2-k_2, \dots, c_m-k_m) \pmod{26}$
- ▶ Exemple
  - ▶ Plaintext: **ATTAQUERATROISHEURE**
  - ▶ Key: GUEREGUEREGE... (la clé « GUERE » est réécrite)
  - ▶ Ciphertext: **GNXRUDYVRXAIMGLNOV**

# TABLEAU DE VIGINÈRE

## Lettres du message

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
B	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a
C	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b
D	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c
E	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d
F	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e
G	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f
H	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g
I	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h
J	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i
K	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j
L	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k
M	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l
N	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m
O	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n
P	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
Q	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
R	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q
S	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r
T	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s
U	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t
V	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u
W	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
X	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w
Y	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x
Z	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y

Clé K<sub>i</sub>

# CHIFFREMENT DE VIGINÈRE: SÉCURITÉ ET CRYPTANALYSE

- ▶ La fréquence d'apparition des lettres est masqué  
→ complique l'analyse fréquentielle
- ▶ Un chiffrement viginère = collection de plusieurs chiffrement par décalage (plusieurs ie la longueur de la clé)
- ▶ Cryptanalyse: Quoi faire?
  - ▶ Trouver la longueur de la clé Kasisky test
  - ▶ Index of coincidence (Freidman)
- ▶ Diviser le message en plusieurs substitutions simples à résoudre Comment?

# KASISKY TEST (1863)

## ► Constat:

- Deux segments identiques du texte clair donnent le même cryptogramme s'ils apparaissent dans le texte à la distance  $\Delta$  ( $\Delta \equiv 0 \pmod m$ ).  $m$  est la longueur du mot clé.

## ► Algorithme

- Rechercher les paires de segments identiques de longueur au minimum 3
- Enregistrer les distances  $\Delta_1, \Delta_2, \dots$
- $M$  divise le PGCD( $\Delta_1, \Delta_2, \dots$ ).

# TEST DE KASISKI: EXEMPLE

Wakeupalicedearsaidhersisterw  
hywhatlongsleepyouvehadohi  
vehadsuchacuriousdREAm sAID  
aliceandshetoldHERSISTERaswell  
asshecouLdrememberthemallth  
eSestrangeadventuresoFherstha  
tyouhavejustBeenREADINGabout  
andwHenSHEhadfiniSHEDHERSIST  
ERkissEDHerandsAlditwasacurio  
usdreamdearcertainlybutnowru  
nintoyourteaitsgettinglate

Hegmmaehquphaijdeelzpqok  
einezjadillpkvydpamhjsqdwsez  
wztzapsowqkzlgqzazyolJPEiaSTH  
wtaniwvvdlabgwHDMJDMOBW  
Ceoewwpwaksiywmwhnmepqx  
mjelauswpppwdxobjlrcmsozavlf  
vaagqlazkelwbqzydinpnqalmia  
vJPEzqfrexwmeejlosijAZPlwlxtre  
AZPHDMJDMOBWCoeakPHDmjlr  
zaSTHebolwwkmcmkckovaieoi  
wzupvpiaypujmerkejfrevlzckcjei  
wqldkablitrctsei

- ▶ Distances observées:
  - ▶ JPE(110), STH(160), HDMJDMOBWC(120), PHD(15), AZP(10)
  - longueur de la clé a une forte probabilité d'être  $\text{PGCD}(110, 160, 120, 15, 10) = 5$

# ONE TIME PAD

- ▶ Résout la faiblesse du chiffrement de viginère
  - ▶ Solution: taille de la clé  $\geq$  taille du message
  - ▶ Une clé est utilisé pour chiffrer un seul message (One Time)
- ▶ Soit l'alphabet  $Z_m = \{0, 1, 2, \dots, (m-1)\}$ 
  - ▶ Espace des clés=espace des textes clair=espace des textes chiffrés=  $(Z_m)$
  - ▶  $P = \{p_1, p_2, \dots, p_n\}$ ,  $K = \{k_1, k_2, \dots, k_n\}$  et  $C = \{c_1, c_2, \dots, c_n\}$ 
    - ▶ Encryption:  $e_k(p_1, p_2, \dots, p_n) = (p_1 + k_1, p_2 + k_2, \dots, p_n + k_n) \pmod m$
    - ▶ Decryption:  $d_k(c_1, c_2, \dots, c_n) = (c_1 - k_1, c_2 - k_2, \dots, c_n - k_n) \pmod m$
- ▶ One Time PAD has perfect secrecy (proved)
- ▶ Le texte chiffré ne donne aucune idée sur le texte clair ni sur la clé.

# LE CHIFFRE DE VERNAM, PARFAITEMENT SÛR (PERFECT SECRECY)

## Qu'est-ce qu'un chiffre parfaitement sûr?

- ▶ Un chiffre parfaitement sûr est un chiffre tel que, l'adversaire interceptant le message, même ayant à sa disposition une puissance de calcul infinie, ne peut pas retrouver la moindre information concernant le message clair à partir du message chiffré.
- ▶ Ceci peut se traduire très bien avec des probabilités. Si on intercepte le message crypté DSJMSZERT, on souhaite qu'avec un chiffre parfaitement sûr, il n'y ait pas plus de chances que le message clair soit ORDINATEUR, CAISSIERES ou DKIRJSMOTS. Tous les messages clairs possibles sont équiprobables!

# LE CHIFFREMENT DE VERNAM | CONTRAINTE

- ▶ Inventé par Gilbert Vernam et publié en 1926.
- ▶ Il peut être décrit simplement comme un chiffre de Vigenère, mais où la clé répond aux trois impératifs suivants :
  - ▶ elle est aussi longue que le texte à chiffrer;
  - ▶ elle est parfaitement aléatoire;
  - ▶ elle n'est utilisée que pour chiffrer un seul message, puis est immédiatement détruite (d'où le nom chiffrement par masque jetable).
- ▶ C'est Claude Shannon qui prouva en 1949 le fait que ce chiffre est parfaitement sûr.

# LE CHIFFREMENT DE VERNAM | PRINCIPE

- ▶ le chiffre de Vernam est implémenté de la façon suivante:
  - ▶ Le message est d'abord converti en suites de bits.
  - ▶ On génère une clé (**complètement aléatoire !**) traduite elle aussi en suite de 0 et de 1. Sa taille doit être aussi longue que le message à chiffrer.
  - ▶ On applique l'opération « **XOR : ou exclusif** » **Bit par bit** entre chaque bit du **message** et le bit qui lui correspond dans la **clé**.
  - ▶ **Décryptage** par **XOR** entre le **cryptogramme** et la **clé** (**Bit par bit**).

Message clair	1	0	1	1	1	0	0	1	1
Clé	0	1	1	1	0	1	0	0	0
Message chiffré	1	1	0	0	1	1	0	1	1

# LE CHIFFREMENT DE VERNAM | DÉFAUTS

- ▶ Ce chiffrement exige qu'une clé serve une seule fois (un seul message). En effet, si on envoie les deux messages  $m_1$  et  $m_2$  avec la même clé  $k$ , on obtient les cryptogrammes  $c_1=m_1 \oplus k$  et  $c_2=m_2 \oplus k$ . Mais si on effectue  $c_1 \oplus c_2$ , lors on obtient.

$$c_1 \oplus c_2 = m_1 \oplus k \oplus k \oplus m_2 = m_1 \oplus m_2$$

- ▶ le chiffre de Vernam exige des clés extrêmement longues, et une parfaite synchronisation des clés.
- C'est pourquoi ce chiffre n'est mis en œuvre que dans des cas très particuliers.

# CHIFFREMENT PAR BLOCS

## ► Méthode:

- ▶ Le message  $M$  à chiffrer est scindé en un nombre de bloc de taille fixe:  $M=m_1, m_2, \dots, m_n$
- ▶ Cryptage des blocs
- ▶ Le cryptogramme  $C$  est obtenu en concaténant les cryptogrammes des blocs

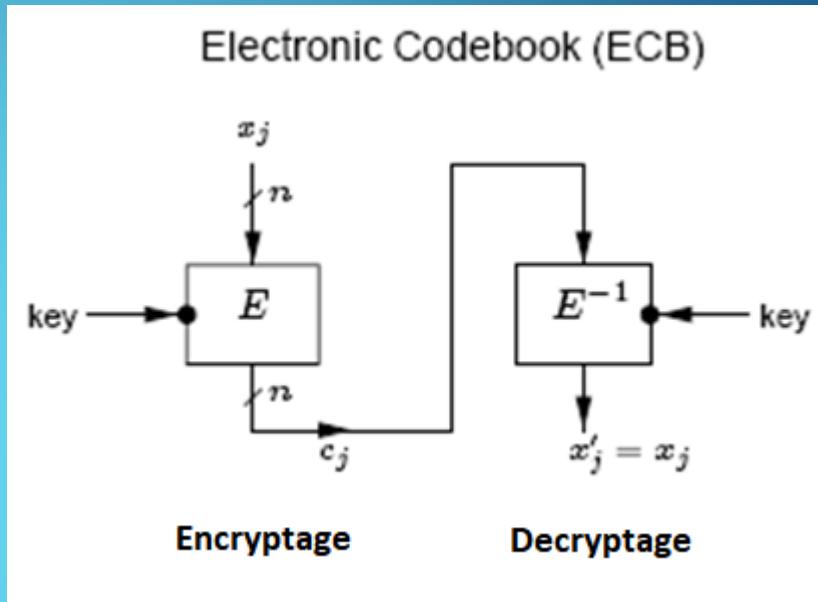
## ► Modes de chiffrement par bloc

- ▶ ECB (Electronic CodeBook)
- ▶ CBC(Cipher bloc Chaining)
- ▶ CFB (Cipher FeedBack)
- ▶ OFB (Output FeedBack)
- ▶ ...

# ECB : (ELECTRONIC CODEBOOK)

## CARNET DE CODAGE ÉLECTRONIQUE

- ▶ ECB est la façon la plus simple de mettre en œuvre un chiffrement par blocs. Chaque bloc est chiffré à part:  $c_i = C_k(m_i)$ .
- ▶ le message chiffré est simplement  $c_1 \dots c_n$ . Le déchiffrement se fait en calculant pour chaque bloc  $D_k(c_i)$ .

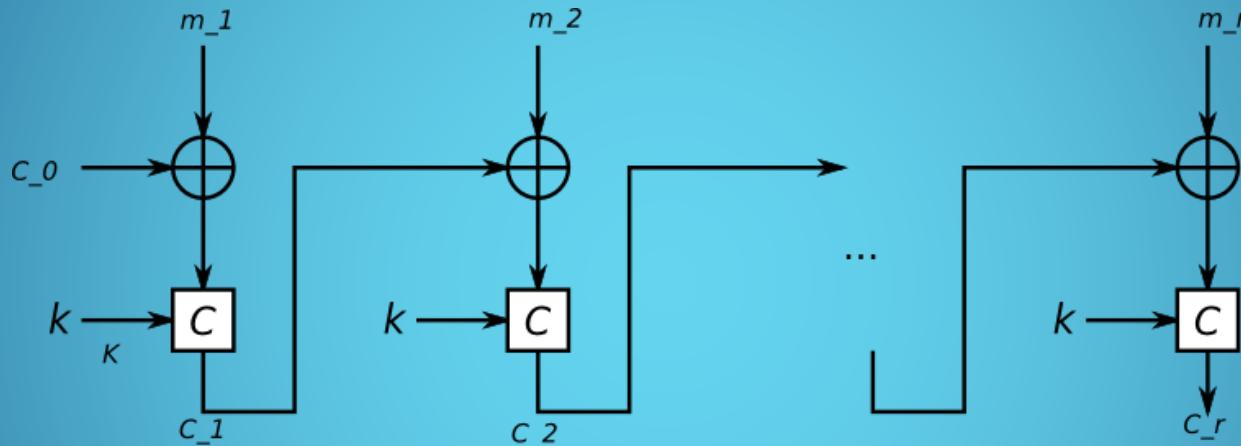


Ce mode souffre de plusieurs défauts de sécurité :

- ▶ Deux blocs clairs identiques sont chiffrés de la même façon. Ceci facilite les attaques statistiques, notamment cela permet de repérer les blocs clairs utilisés les plus fréquemment.
- ▶ Le mode ECB ne garantie pas l'intégrité des données. Un attaquant peut remplacer certains blocs chiffrés par d'autres blocs chiffrés du message, ou permutez deux blocs, sans que le destinataire s'en aperçoive.

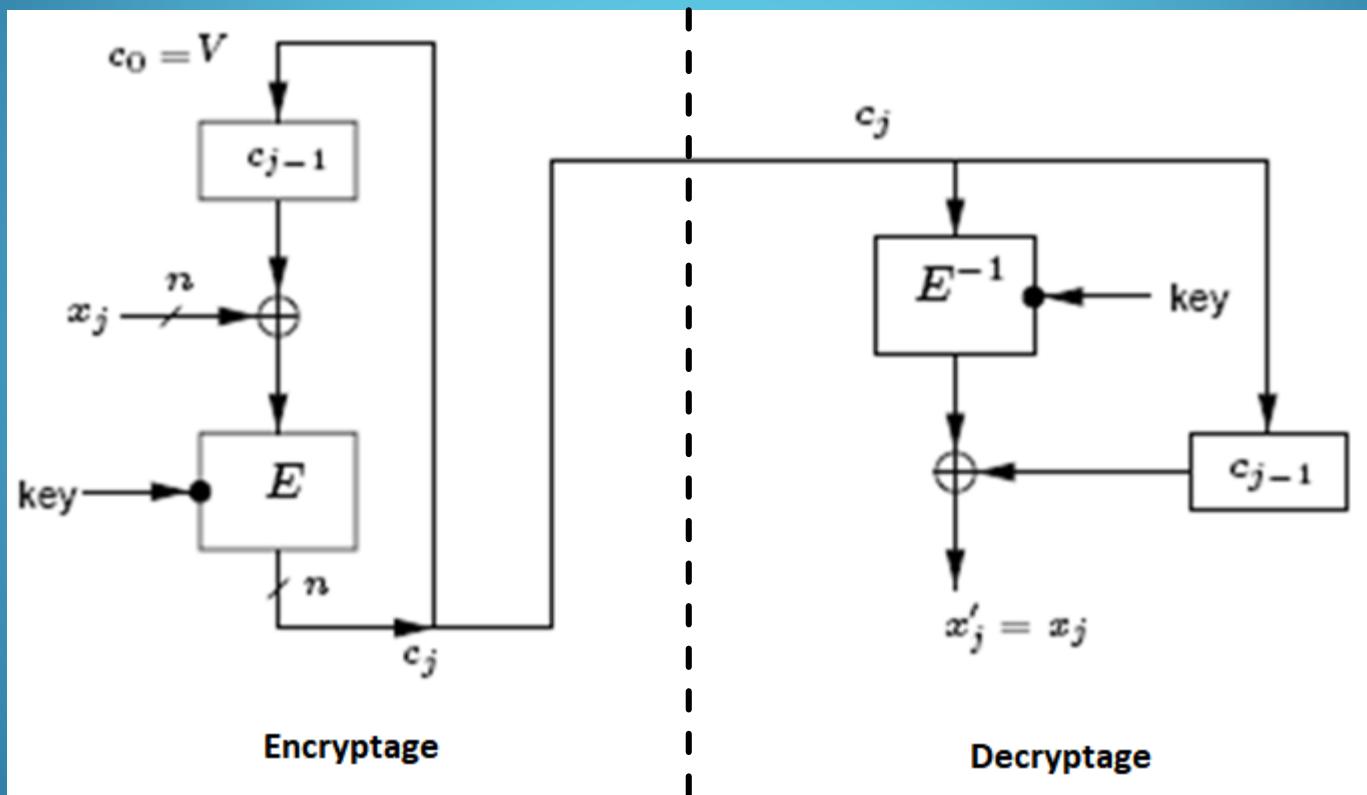
# CBC( CIPHER BLOC CHAINING) CHIFFREMENT PAR CHAÎNAGE DE BLOCS

- Ici, chaque bloc du cryptogramme dépend du bloc de texte en clair et de tous les blocs précédents (donc le chiffrement dépend du contexte)



- On commence par fixer un mot initial de  $n$  bits  $V$ , et on pose  $c_0=V$ . Le message clair est toujours fractionné en  $m_1, m_2, \dots, m_r$ . Le chiffrement du  $i^{\text{ème}}$  bloc est alors obtenu par  $c_i = C_k(c_{i-1} \oplus m_i)$ . Ainsi, il dépend non seulement du  $i^{\text{ème}}$  bloc du message clair, mais aussi du message chiffré précédent.
- Pour déchiffrer le message  $c_1 \dots c_r$ , il faut aussi connaître le mot initial  $V$ . On calcule alors, pour chaque bloc  $i$ ,  $m_i = c_{i-1} \oplus D_k(c_i)$ , et on peut prouver que  $c_1 \dots c_r$  est bien le message initial (Intégrité).
- L'inconvénient principal de ce mode est sa lenteur.

# CBC( CIPHER BLOC CHAINING)

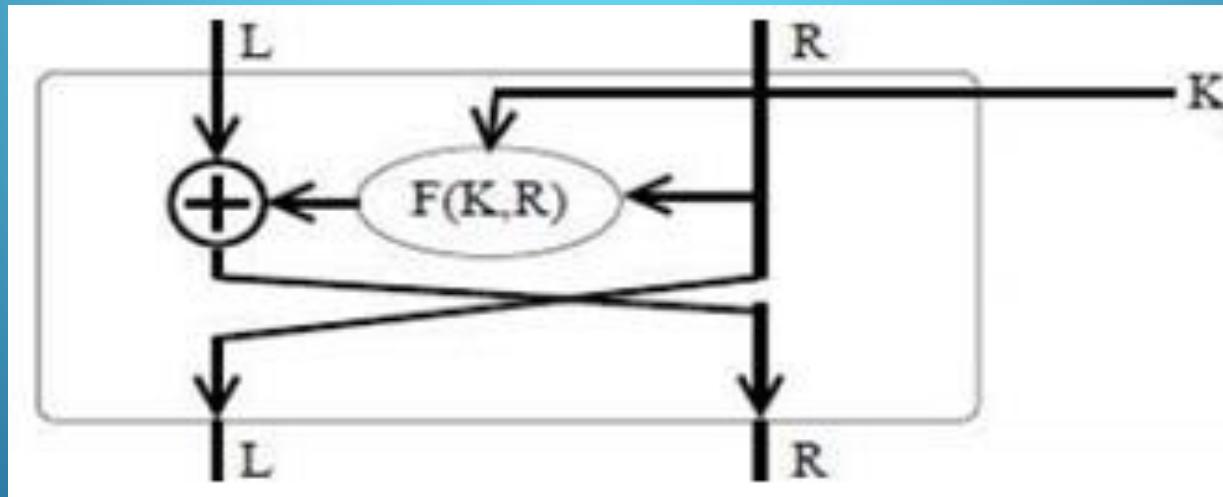


# FEISTEL BLOCK CIPHER

- ▶ Le chiffrement de Feistel n'est pas un schéma spécifique de chiffrement par bloc. C'est un modèle de conception à partir duquel de nombreux chiffrements par blocs sont dérivés. (Exemple : DES).
- ▶ Un système cryptographique basé sur la structure de chiffrement de Feistel utilise le même algorithme pour le chiffrement et le déchiffrement.

# PROCESSUS DE CRYPTAGE

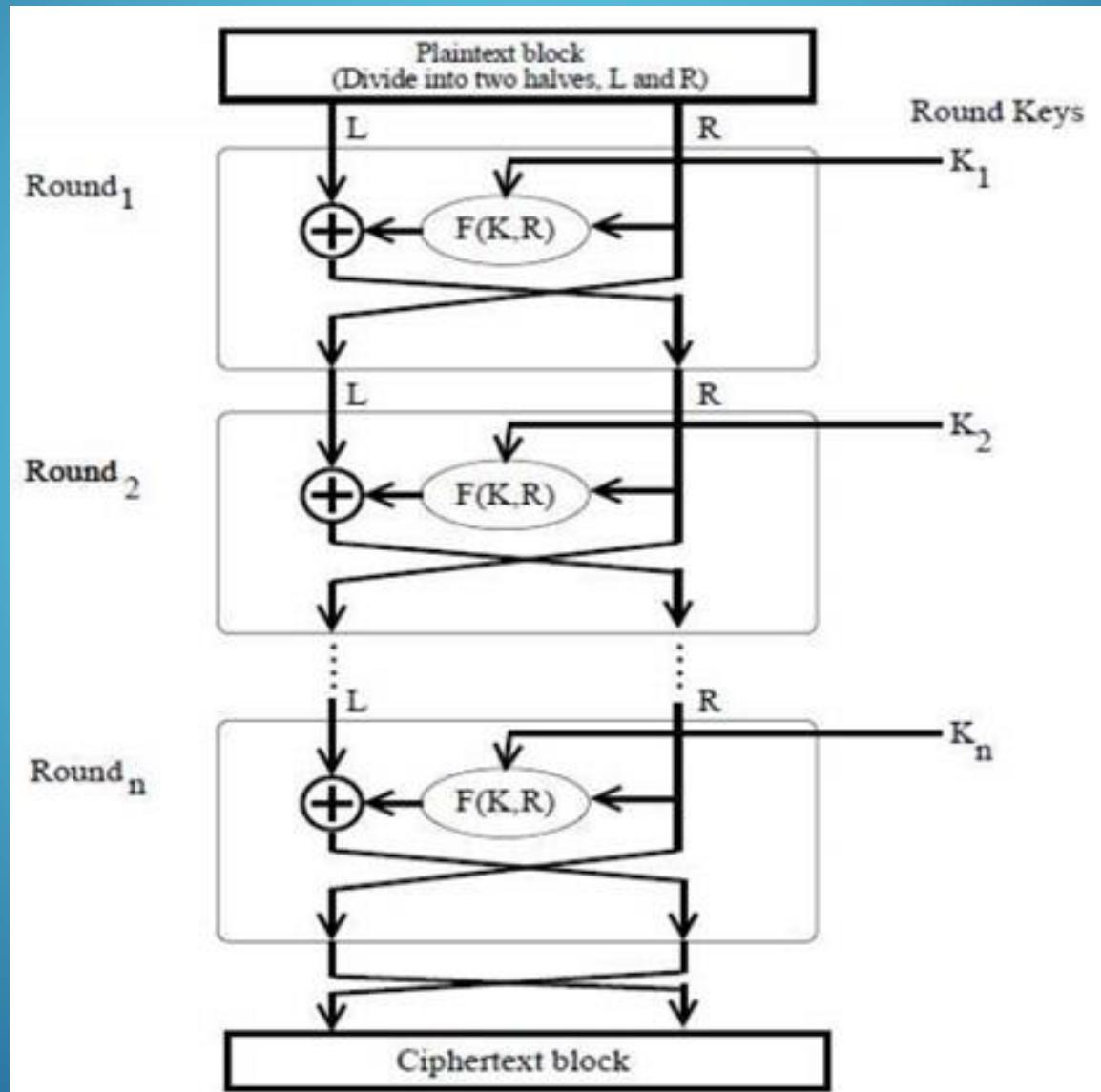
- Le processus de cryptage qui utilise la structure de Feistel consiste en plusieurs cycles de traitement du texte en clair, chaque cycle étant constitué d'une étape de "substitution" suivie d'une étape de permutation.



# FEISTEL | DÉTAIL DU PROCESSUS DE CRYPTAGE PAR TOUR

- ▶ Le bloc d'entrée de chaque tour est divisé en deux moitiés pouvant être désignées par L et R pour la moitié gauche et la moitié droite.
- ▶ À chaque tour, la moitié droite du bloc, R, passe sans changement. Mais la moitié gauche, L, passe par une opération qui dépend de R et de la clé de cryptage. Tout d'abord, nous appliquons une **fonction** de chiffrement « **f** » qui prend deux entrées - les clés K et R. La fonction produit la sortie  $f(R, K)$ . Ensuite, on applique une opération XOR sur la sortie de la fonction mathématique avec L.
- ▶ Dans la mise en œuvre réelle du chiffre de Feistel, tel que DES, au lieu d'utiliser la clé de chiffrement complète à chaque tour, une clé dépendante du tour (une sous-clé) est dérivée de la clé de chiffrement. Cela signifie que chaque tour utilise une clé différente, bien que toutes ces sous-clés soient liées à la clé d'origine.
- ▶ Le pas de permutation à la fin de chaque tour permute le L (modifié) et le R (non modifié). Par conséquent, le L du tour suivant serait R du tour en cours. Et R pour le tour suivant sera la sortie L du tour en cours.
- ▶ Les étapes de substitution et de permutation ci-dessus forment un **tours** « Round ». Le nombre de tours est spécifié par la conception de l'algorithme.
- ▶ Une fois le dernier tour terminé, les deux sous-blocs « R » et « L » sont concaténés dans cet ordre pour former le bloc de texte chiffré.

# FEISTEL | SCHÉMATISATION DU PROCESSUS GLOBALE DE CRYPTAGE



# CHALLENGE

- ▶ La difficulté de la conception d'un chiffre de Feistel est la sélection de la fonction du tour «  $f$  ». Pour être un schéma incassable, cette fonction doit avoir plusieurs propriétés importantes qui sortent du cadre de notre discussion.

# PROCESSUS DE DÉCRYPTAGE

- ▶ Le processus de déchiffrement dans le chiffrement de Feistel est presque similaire. Au lieu de commencer par un bloc de texte en clair, le bloc de texte chiffré est introduit au début de la structure de Feistel, puis le processus est ensuite identique à celui décrit dans l'illustration donnée.
- ▶ On dit que le processus est presque similaire et pas exactement le même. Dans le cas du déchiffrement, la seule différence est que les sous-clés utilisées dans le chiffrement sont utilisées dans l'ordre inverse.
- ▶ L'échange final de « L » et de « R » à la dernière étape du chiffrement de Feistel est essentiel. Si ceux-ci ne sont pas échangés, le texte chiffré résultant ne pourrait pas être déchiffré à l'aide du même algorithme.

# PRINCIPE DE CONCEPTION D'UN ALGORITHME DE CHIFFREMENT PAR BLOC A BASE DE FEISTEL

- ▶ **Division du Bloc de texte clair en deux moitiés**
- ▶ **Taille du Bloc**
- ▶ **Taille de la clé**
- ▶ **Nombre de tours (répétition)**
- ▶ **Nombre de sous-clés**
- ▶ **Choix de la fonction F**

# REMARQUE : NOMBRE DE TOURS

- ▶ Le nombre de tours utilisés dans un chiffre de Feistel dépend de la sécurité souhaitée du système. Plus le nombre de tours est élevé, plus le système est sécurisé. Mais dans le même temps, plus de tours signifient que les processus cryptage et de décryptage deviennent plus lents. Le nombre de tours dans les systèmes dépend donc du **compromis efficacité-sécurité**.

# DES (DATA ENCRYPTION STANDARD)

- ▶ Fruit des efforts conjoints d'IBM, qui propose Lucifer fin 1974, et de la NSA (*National Security Agency*).
- ▶ Le cahier des charges était le suivant :
  - ▶ l'algorithme repose sur une clé relativement petite, qui sert à la fois au chiffrement et au déchiffrement.
  - ▶ l'algorithme doit être facile à implémenter, logiciellement et matériellement, et doit être très rapide.
  - ▶ le chiffrement doit avoir un haut niveau de sûreté, uniquement lié à la clé, et non à la confidentialité de l'algorithme.

# DES (DATA ENCRYPTION STANDARD)

- ▶ Chiffrement par bloc : 64bits
- ▶ Clé de taille variable:
  - ▶ Entre 64 et 128 bits selon le niveau de sécurité désiré
  - ▶ Version initiale du DES utilise une clé de taille 64 bits dont 56 sont réellement utilisés
- ▶ Utilise un chiffrement produit
  - ▶ Combine des algorithmes de substitution et des algorithmes de transposition → maximiser la complexité de l'algorithme
- ▶ Basé sur le chiffrement de Feistel



# DES | IMPLÉMENTATION DE LA STRUCTURE FEISTEL

- ▶ **Division du texte clair en deux moitiés**
- ▶ **Taille du Bloc = 64 bits**
- ▶ **Taille de la clé = 64/128 bits**
- ▶ **Nombre de tours = 16**
- ▶ **Nombre de sous-clés = 16**
- ▶ **Taille des sous-clés = 48**
- ▶ **Choix de la fonction F = ...**

# DES | PRINCIPE DE FONCTIONNEMENT

D'une manière générale, on peut dire que DES fonctionne en trois étapes :

1. **permutation initiale** et fixe d'un bloc (sans aucune incidence sur le niveau de sécurité) ;
2. le résultat est soumis à **16 itérations** d'une transformation, ces itérations dépendent à **chaque tour** d'une autre **clé partielle** de **48 bits**. Cette clé de tour intermédiaire est calculée à partir de la clé initiale de l'utilisateur (grâce à un réseau de **tables de substitution** et d'opérateurs **XOR**). Lors de chaque tour, le bloc de 64 bits est découpé en deux blocs de 32 bits, et ces blocs sont échangés l'un avec l'autre selon un **schéma de Feistel**. Le bloc de 32 bits ayant le poids le plus fort (celui qui s'étend du bit 32 au bit 64) subira une transformation ;
3. le résultat du dernier tour est transformé par la fonction **inverse de la permutation initiale**.

# DES | ALGORITHME

- ▶ Soit  $X$  est le texte clair de 64 bits.

$$(G_0, D_0) = IP(X)$$

- ▶ Pour  $i=1$  à  $16$

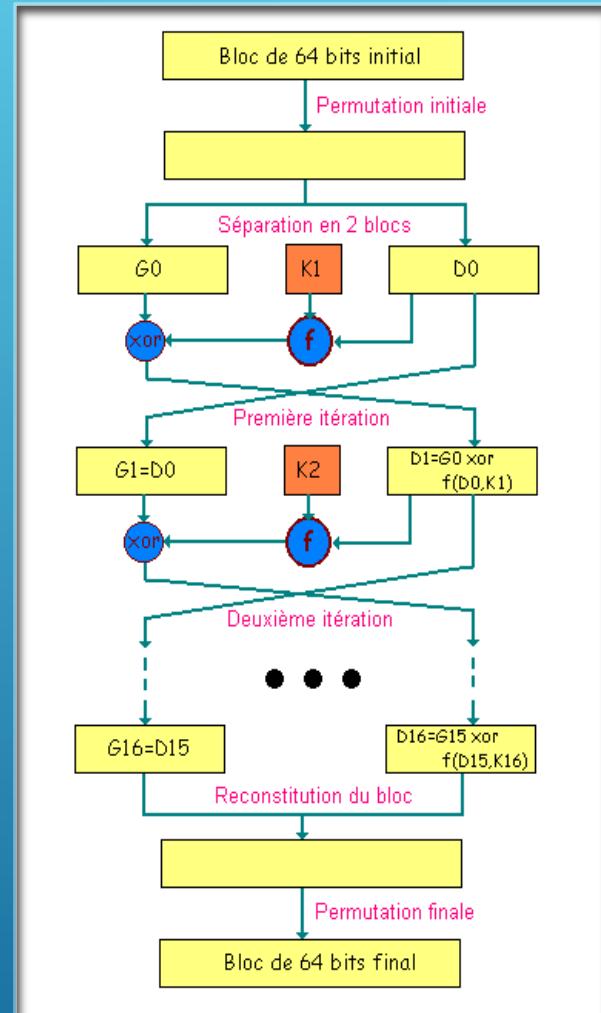
$$(G_i, D_i) = (D_{i-1}, G_{i-1} \oplus f(D_{i-1}, K_i))$$

- ▶ Soit  $Y$  est le texte chiffré de 64 bits.

$$Y = IP^{-1}(D_{16}, G_{16})$$

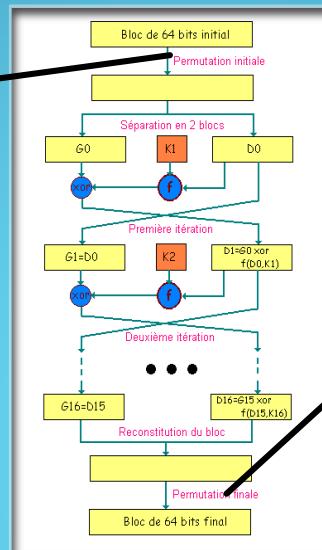
- ▶ Chaque  $k_i$  est une chaîne de 48 bits provenant de  $K$ .

- ▶ Pour déchiffrer, on utilise le même algorithme avec les clefs  $K_i$  utilisées dans l'ordre inverse.



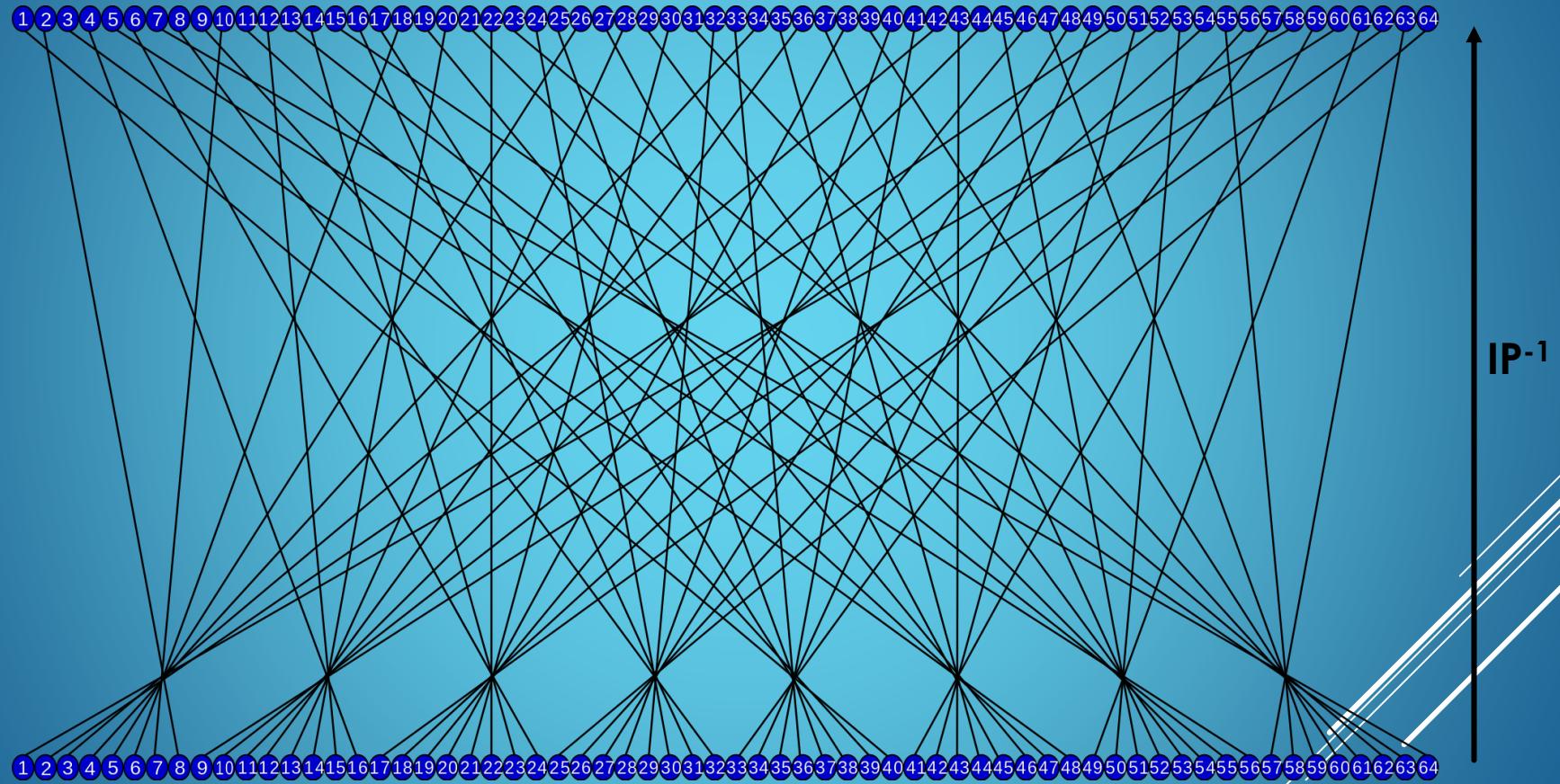
# P | MATRICES DE PERMUTATIONS INITIALES

58 50 42 34 26 18 10 2  
60 52 44 36 28 20 12 4  
62 54 46 38 30 22 14 6  
64 56 48 40 32 24 16 8  
57 49 41 33 25 17 9 1  
59 51 43 35 27 19 11 3  
61 53 45 37 29 21 13 5  
63 55 47 39 31 23 15 7



40 8 48 16 56 24 64 32  
39 7 47 15 55 23 63 31  
38 6 46 14 54 22 62 30  
37 5 45 13 53 21 61 29  
36 4 44 12 52 20 60 28  
35 3 43 11 51 19 59 27  
34 2 42 10 50 18 58 26  
33 1 41 9 49 17 57 25

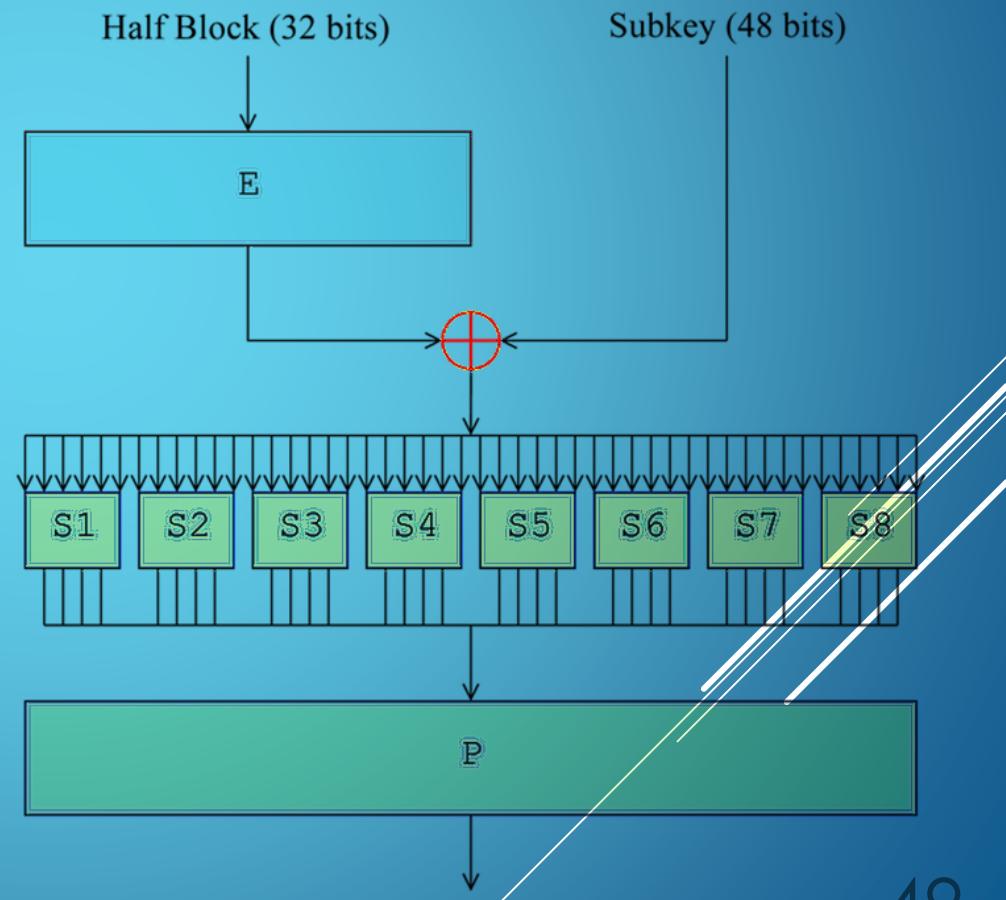
# P | SCHÉMATISATION DES MATRICES IP ET IP<sup>-1</sup>



# LA FONCTION $F_K$

- DES utilise huit tables de substitution (les S-Boxes) et une table d'expansion en plus des matrices de permutation

$$f(D_{i-1}, K_i) = P(S(E(D_{i-1}) \oplus K_i))$$



$K_i$  : Clé du tour en cours

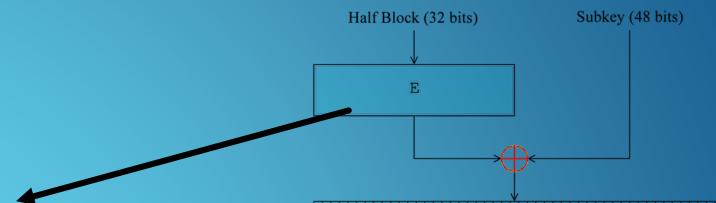
E : Expansion

S : Substitution

P : Permutation

$D_{i-1}$  : La partie droite du tour précédent

# E | MATRICE D'EXPANSION



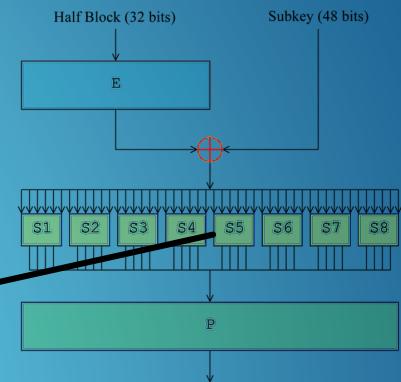
32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	01

# S | MATRICES DE SUBSTITUTION (S-BOX)

Exemple : **011011**

**01**  
**1101**

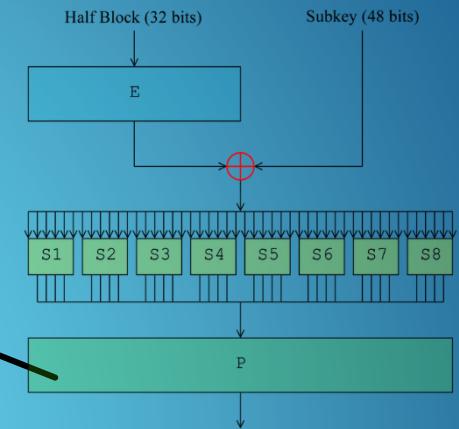
4 bits au centre de l'entrée																	
		<b>s<sub>5</sub></b>															
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Bits externes	00	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011



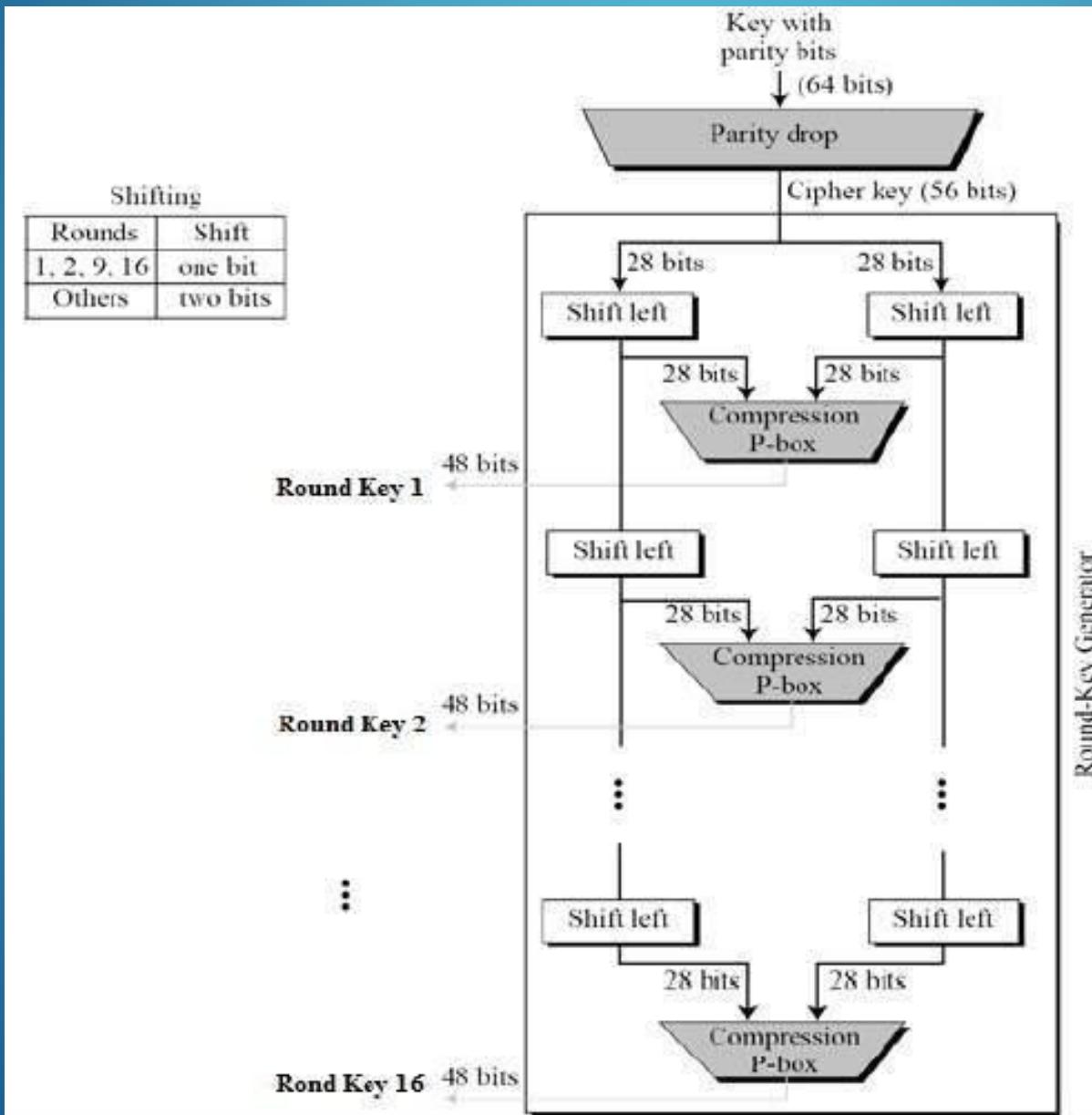
# P | MATRICES DE PERMUTATION

► P

<b>16</b>	<b>07</b>	<b>20</b>	<b>21</b>
<b>29</b>	<b>12</b>	<b>28</b>	<b>17</b>
<b>01</b>	<b>15</b>	<b>23</b>	<b>26</b>
<b>05</b>	<b>18</b>	<b>31</b>	<b>10</b>
<b>02</b>	<b>08</b>	<b>24</b>	<b>14</b>
<b>32</b>	<b>27</b>	<b>03</b>	<b>09</b>
<b>19</b>	<b>13</b>	<b>30</b>	<b>06</b>
<b>22</b>	<b>11</b>	<b>04</b>	<b>25</b>



# GÉNÉRATION DES CLÉS DE TOURS



# P-BOX (CHOIX 1 & 2 DE PERMUTATION)

**PC-1**

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

**PC-2**

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

# DES | REMARQUE

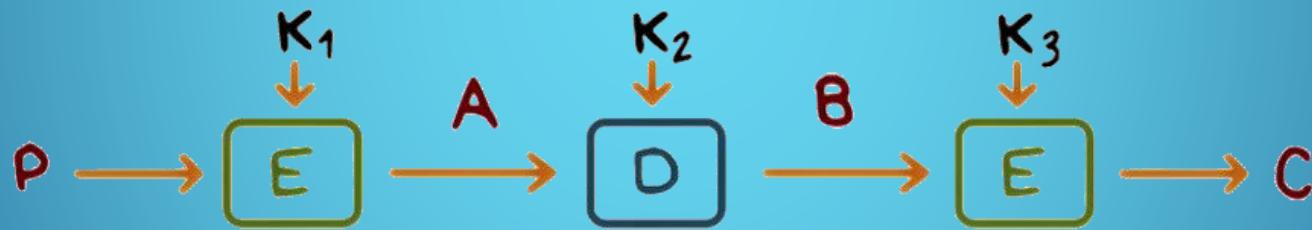
- ▶ Seulement 56 bits de la clef de 64 bits sont utilisées. Les 8 autres sont des bits de vérification

# PARTICULARITÉS DU DES

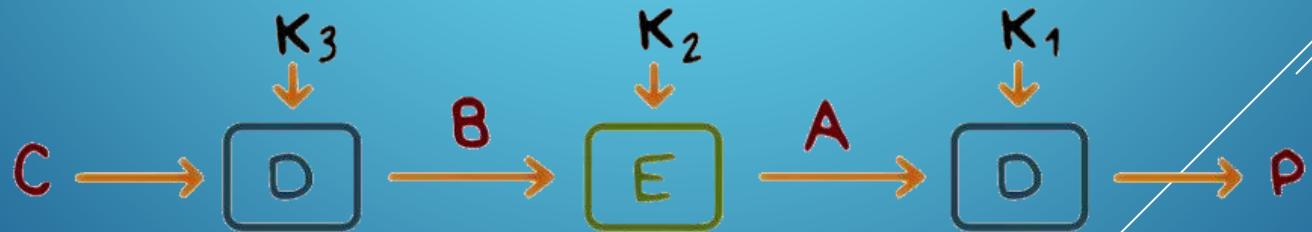
- ▶ **Souplesse d'implémentation:**
  - ▶ ECB ou CBC (en modifiant la phase de pré traitement des blocs de données)
  - ▶ Différentes implémentations en modifiant les fonctions d'expansion ou de sélection
- ▶ **Faiblesses**
  - ▶ **Conservation de la taille**
    - ▶ sensible aux attaques d'analyse de flux: on peut connaître la taille exacte de chaque message
  - ▶ **La clé est réduite à 56 bits**
    - ▶ réduit la sécurité de l'algorithme
  - ▶ **Avec une clé de taille 128 bits**
    - ▶ algorithme coûteux en temps
  - ▶ Peut être cassé par les processeurs actuels (exhaustive key search)
- ▶ **Alternatives :**
  - ▶ Première alternative : Triple DES (DES avec trois clés différentes)
  - ▶ Deuxième alternative : AES, remplaçant du DES

# TRIPLE DES | PRINCIPE

- Le Triple DES (aussi appelé 3DES) est un algorithme de chiffrement symétrique par bloc, enchaînant 3 applications successives de l'algorithme DES sur le même bloc de données de 64 bits, avec 2 ou 3 clés DES différentes.
- Encryptage :



- Décryptage :



# TRIPLE DES | CLÉS

- ▶ Trois clés : K1, K2 et K3 différentes est l'équivalent de DES-168bits
- ▶ Deux clés : K1 = K3 est l'équivalent de DES-112bits
- ▶ Si K1 = K2 → 3DES = DES (avec comme clé K3). Ceci peut être utilisé pour des raison de compatibilité.

# AES : ADVENCED ENCRYPTION STADARD

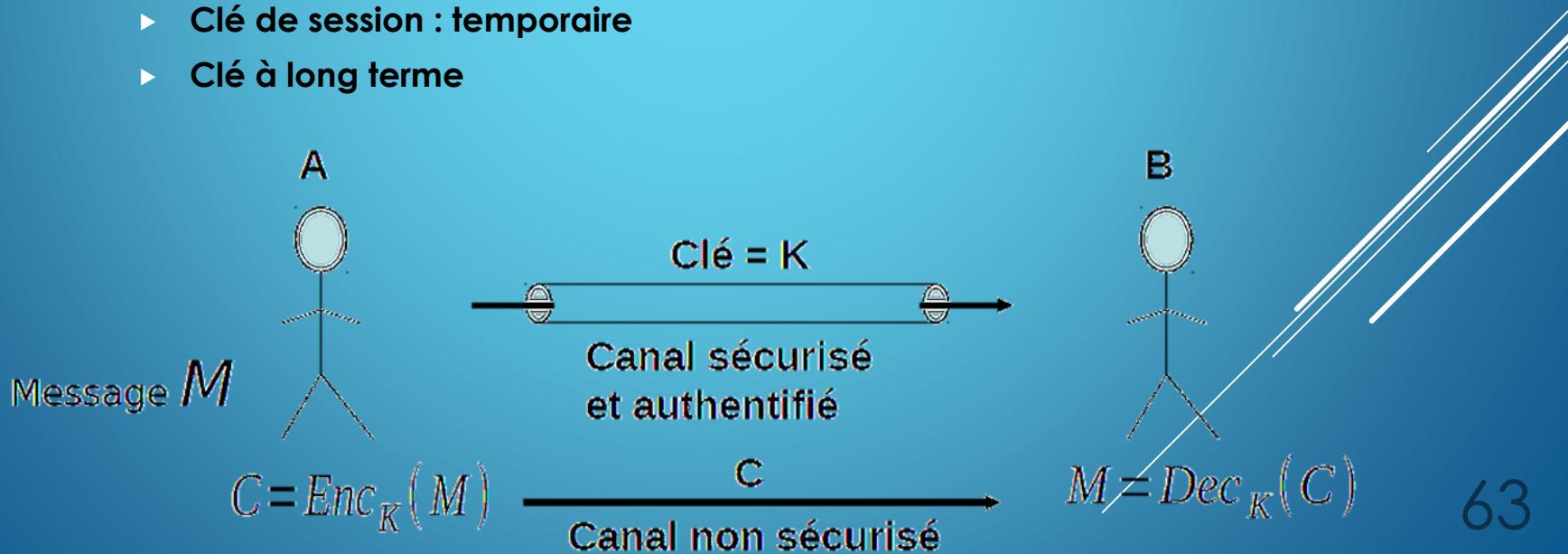
- ▶ AES, aussi connu sous le nom « Rijndael », a été développé par deux belges Joan Daemen et Vincent Rijmen. Il a été standardisé en 2002.
- ▶ C'est l'algorithme le plus utilisé pour le cryptage symétrique.
- ▶ Il s'agit d'un algorithme de cryptage symétrique par bloc.
- ▶ C'est l'algorithme recommandé par le gouvernement américain pour crypter des données sur internet.
- ▶ Utilise des clés de 128, 192 et 256 bits
- ▶ Son fonctionnement est décrit dans l'animation qui suit...

# ALGORITHME DE CRYPTAGE ASYMÉTRIQUES

# GESTION DE CLÉS

## ► Chiffrement symétrique (revisité):

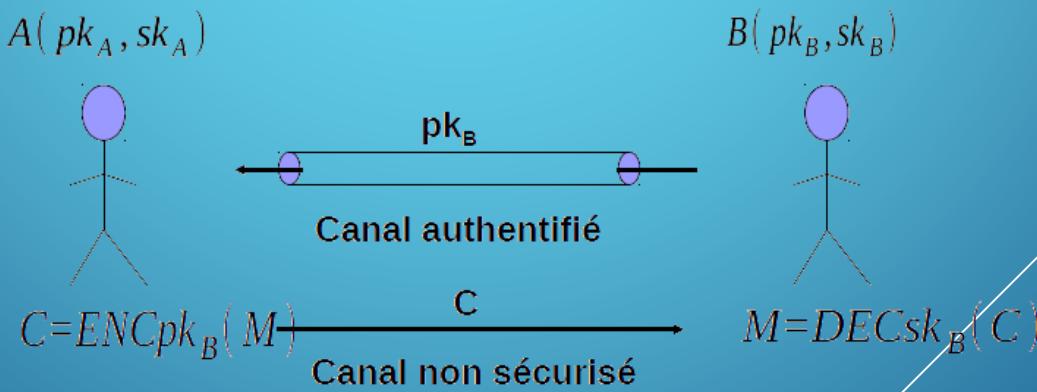
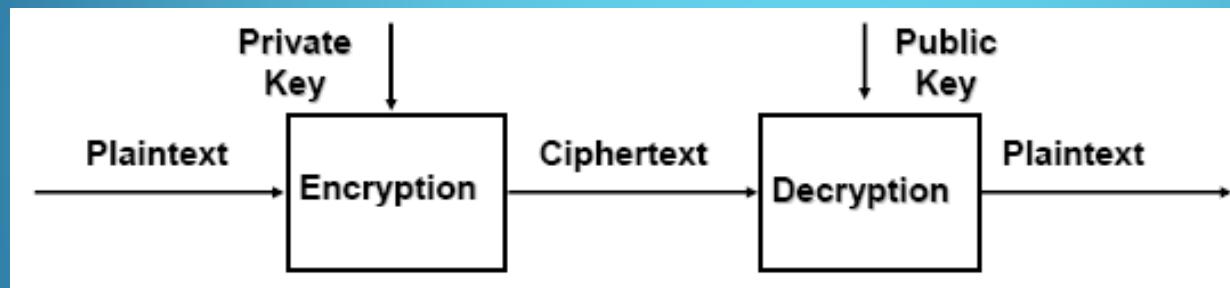
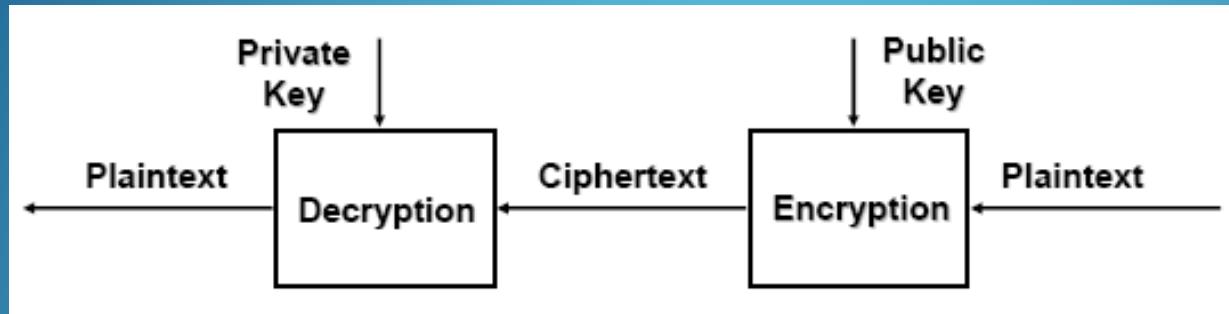
- ▶ Comment établir la clé K d'une manière sécurisé ?
- ▶ Key transport : l'un crée la clé et la transmet à l'autre
- ▶ Problème :
  - ▶ Besoin de  $N(N-1)/2$  clés pour N utilisateur => trop de clés
- ▶ Solutions :
  - ▶ Key Distribution Center (KDC) : utiliser un centre de distribution de clé. Chaque entité partage une clé avec le centre.
  - ▶ Key agreement : une clé partagée est dérivée par 2 entités (ou +).
- ▶ Comment mettre à jour la clé K ?
  - ▶ Clé de session : temporaire
  - ▶ Clé à long terme



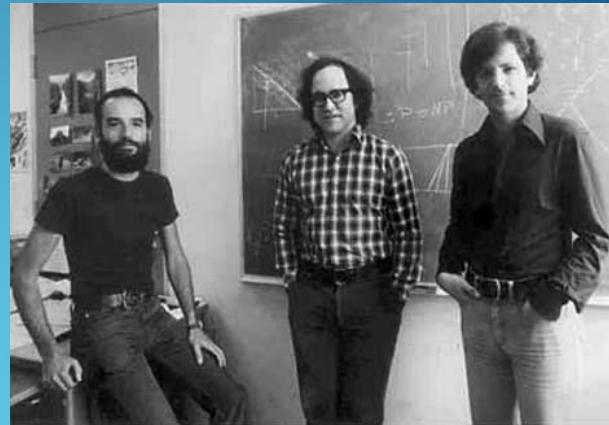
# CRYPTAGE ASYMÉTRIQUE (À CLÉ PUBLIQUE)

- ▶ Utilisation d'une paire de clés:
  - ▶ **Publique**: Connue par tout le monde, utilisée généralement pour crypter ou vérifier la signature des messages.
  - ▶ **Privée**: Connue uniquement par le détenteur, utilisée pour décrypter et signer des messages.
- ▶ Propriété importante :
  - ▶ Impossible de trouver la clé privée à partir de la clé publique.
- ▶ Exemples: RSA, Diffie-Hellman, El Gamal.
- ▶ Généralement dix fois plus lent que le cryptage symétrique.
- ▶ Utilisé généralement pour
  - ▶ Cryptage / décryptage: assurer la confidentialité.
  - ▶ Signature numérique: assurer l'authentification et la non répudiation.
  - ▶ Distribution de clés: se mettre d'accord sur une clé de session.
- ▶ Clés à grande taille (ex: RSA: 1024-2048-...)

# PRINCIPE | UTILISATION DES CLÉS



# RSA | SOLUTION AU SYSTÈME À CLÉ PUBLIQUE



- ▶ **Rivest, Shamir, Adleman (MIT, 1976)**
- ▶ **Le plus connu et le plus utilisé comme algorithme de cryptage asymétrique.**
- ▶ **Breveté par RSA, et cette patente a expiré en 2000.**
- ▶ **Utilise des entiers très larges 1024+ bits**
- ▶ **Le fonctionnement du cryptosystème RSA est basé sur la difficulté de factoriser de grands entiers.**
- ▶ **Rappel:**
  - ▶ **Multiplication A.B = C     $47 \times 71 = 3337$**
  - ▶ **Factorisation C=A.B     $3337 = ?\cdot?$**

# DIFFICULTÉ DE FACTORISER DE GRANDS ENTIERS

```
1143816257578888676692357799761466120102182967212423625625618429  
35706935245733897830597123563958705058989075147599290026879543541  
=  
3490529510847650949147849619903898133417764638493387843990820577  
*  
32769132993266709549961988190834461413177642967992942539798288533
```

- Concours RSA-129
- Il a fallu 8 mois à 600 ordinateurs pour factoriser ce nombre!
- La vérification se fait en moins d'un millième de seconde.
- THE MAGIC WORDS ARE **SQUEAMISH OSSIFRAGE**

# DERNIÈRE COMPÉTITION DE FACTORISATION RSA

Compétition	Prix	Statut	Date de factorisation	Par
RSA-576	USD 10 000	Factorisé	3 décembre 2003	J. Franke et al.
RSA-640	USD 20 000	Factorisé	2 novembre 2005	F. Bahr et al.
RSA-704	USD 30 000	Annulé	2 juillet 2012	Shi Bai, Emmanuel Thomé et Paul Zimmermann
RSA-768	USD 50 000	Factorisé	15 janvier 2010	Divers organismes
RSA-896	USD 75 000	Annulé	-	-
RSA-1024	USD 100 000	Annulé	-	-
RSA-1536	USD 150 000	Annulé	-	-
RSA-2048	USD 200 000	Annulé	-	-

# RAPPEL MATHÉMATIQUE

- ▶ A mod B est le reste de la division entière de A par B

- ▶ La multiplication et le modulo

$$(A \text{ mod } B) (C \text{ mod } B) = A * C \text{ mod } B$$

- ▶ L'exponentielle et le modulo

$$a^n \text{ mod } m = (a \text{ mod } m)^n \text{ mod } m$$

- ▶ Pierre Fermat : Si on utilise un nombre premier comme module, alors quand on élève un nombre à la puissance (nombre premier -1), on obtient 1

- ▶ Pour n'importe quel nombre m et pour p premier :

$$m^{(p-1)} \text{ mod } p = 1$$

- ▶ Exemple :  $7^{10} \text{ mod } 11 = 1$  ...pas besoin de calcul car 11 est premier

# RAPPEL MATHÉMATIQUE

- Leonhard Euler : Lorsqu'on utilise un module comme étant le produit de deux nombres premiers on a :

Soit  $n = p * q$ , avec  $p$  et  $q$  premiers, et quelque soit  $m$

$$m^{(p-1)(q-1)} \bmod n = 1$$

- Exemple :

soit  $p = 11$  et  $q = 5$ ,  $n = 55$  et  $(p - 1)(q - 1) = 10 * 4 = 40$

$$38^{40} \bmod 55 = 1$$

- Si on manipule le résultat d'Euler en multipliant par  $m$  l'équation :

$$m * m^{(p-1)(q-1)} \bmod n = m$$

$$m^{(p-1)(q-1)+1} \bmod n = m$$

# NOMBRES PREMIERS & CONGRUENCES MODULO N

## Nombres premiers

- ▶ **Pgcd (a,n)=1**
  - ▶ a et n sont premiers entre eux
- ▶ **n est premier : seuls diviseurs sont 1 et n.**
  - ▶  $A^{n-1} \equiv 1(n)$  si  $a < n$  (Fermat)
  - ▶  $A^{(p-1).(q-1)} \equiv 1(n)$  si  $a < n=p.q$  (Euler)

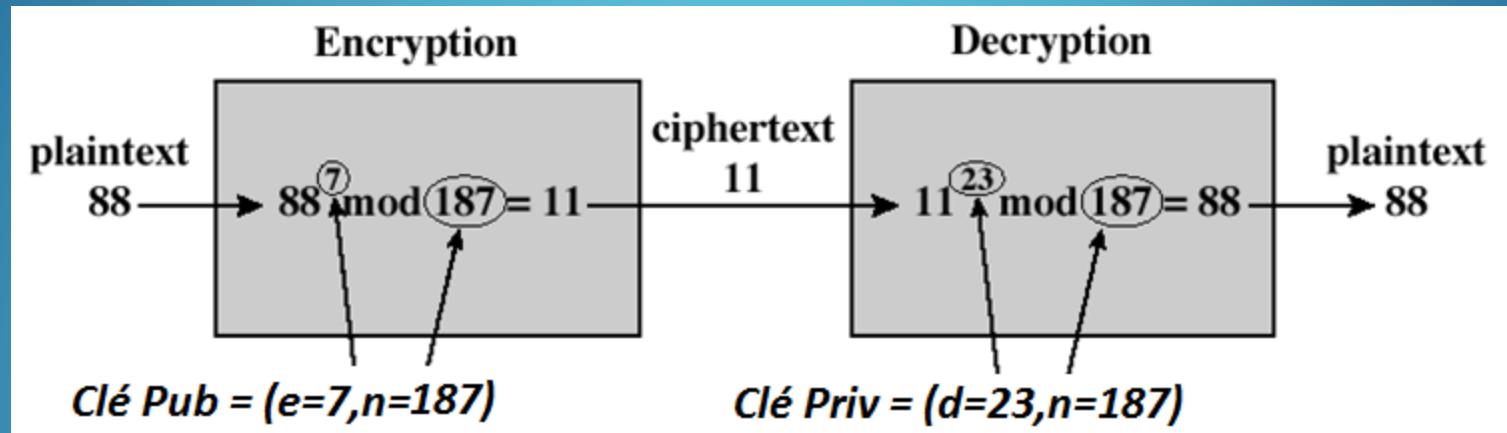
## Congruences modulo n

- ▶  $x.a \equiv 1(n) : x, a < n$  et  $x.a = k.n + 1$ 
  - ▶ par exemple  $x.3 = 1 (14)$
- ▶ **Solutions si (a,n)=1**
  - ▶ si  $a = 3$ , alors  $x = 5$  car  $5.3 = 15 = 1.14 + 1$

# RSA | ÉTAPES DE L'ALGORITHME

- ▶ **Génération des clés (Privée/publique)**
  1. Sélectionner deux entiers premiers entre eux «  $p$  » et «  $q$  »
  2. Calculer  $n = p \times q$
  3. Calculer  $\varphi(n) = (p - 1)(q - 1)$
  4. Sélectionner "e" (au hazard) tel que:  $\text{pgcd}(\varphi(n), e) = 1 ; 1 < e < \varphi(n)$   
*En général « e » est un entier de petite taille.*
  5. Calculer  $d = e^{-1} \bmod \varphi(n)$ . En d'autre terme:  $d \cdot e = 1 \bmod (\varphi(n))$
  6. Clé publique:  $K_{pu} = \{e, n\}$
  7. Clé privée  $K_{pr} = \{d, n\}$
- ▶ Pour **crypter** un message  $M < n$ , l'émetteur:
  - ▶ Obtient une clé publique du récepteur et calcule «  $C = M^e \bmod n$  »
- ▶ Pour **décrypter** un message crypté  $C$  le récepteur
  - ▶ Utilise sa clé privée et calcule «  $M = C^d \bmod n$  »

# RSA | EXEMPLE D'APPLICATION



- $p = 17, q = 11, n = p \times q = 187$
- $\Phi(n) = (p-1) \times (q-1) = 16 \times 10 = 160$
- **Choisir e = 7**
- $d \cdot e \equiv 1 \pmod{\Phi(n)} \rightarrow d = 23$

# RSA | EXEMPLE D'UTILISATION 1

**Création de la paire de clés:**

- ▶ Soient deux nombres premiers au hasard:  $p = 29$ ,  $q = 37$ , on calcule  $n = pq = 29 * 37 = 1073$ .
- ▶ On doit choisir  $e$  au hasard tel que  $e$  n'ai aucun facteur en commun avec  $\varphi(n) = (p - 1)(q - 1)$ :
- ▶  $\varphi(n) = (p - 1)(q - 1) = (29 - 1)(37 - 1) = 1008$
- ▶ On prend  $e = 71$
- ▶ On choisit  $d$  tel que  $71*d \bmod 1008 = 1$ , on trouve  $d = 1079$ .
- ▶ On a maintenant les clés :
  - ▶ la clé publique est  $(e,n) = (71,1073)$  (=clé de chiffrement)
  - ▶ la clé privée est  $(d,n) = (1079,1073)$  (=clé de déchiffrement)

# RSA | EXEMPLE D'UTILISATION 2

Chiffrement du message 'HELLO'.

- ▶ On prend le code ASCII de chaque caractère et on les met bout à bout:
- ▶  $m = 7269767679$
- ▶ Il faut découper le message en blocs qui comportent moins de chiffres que  $n$ .
- ▶  $n$  comporte 4 chiffres, on découpe notre message en blocs de 3 chiffres:
- ▶ 726 976 767 900 (on complète avec des zéros)
- ▶ On chiffre chacun de ces blocs :
- ▶  $726^{71} \text{ mod } 1073 = 436$
- ▶  $976^{71} \text{ mod } 1073 = 822$
- ▶  $767^{71} \text{ mod } 1073 = 825$
- ▶  $900^{71} \text{ mod } 1073 = 552$
- ▶ Le message chiffré est 436 822 825 552.

# RSA | COMPARAISON AVEC DES

## ► RSA

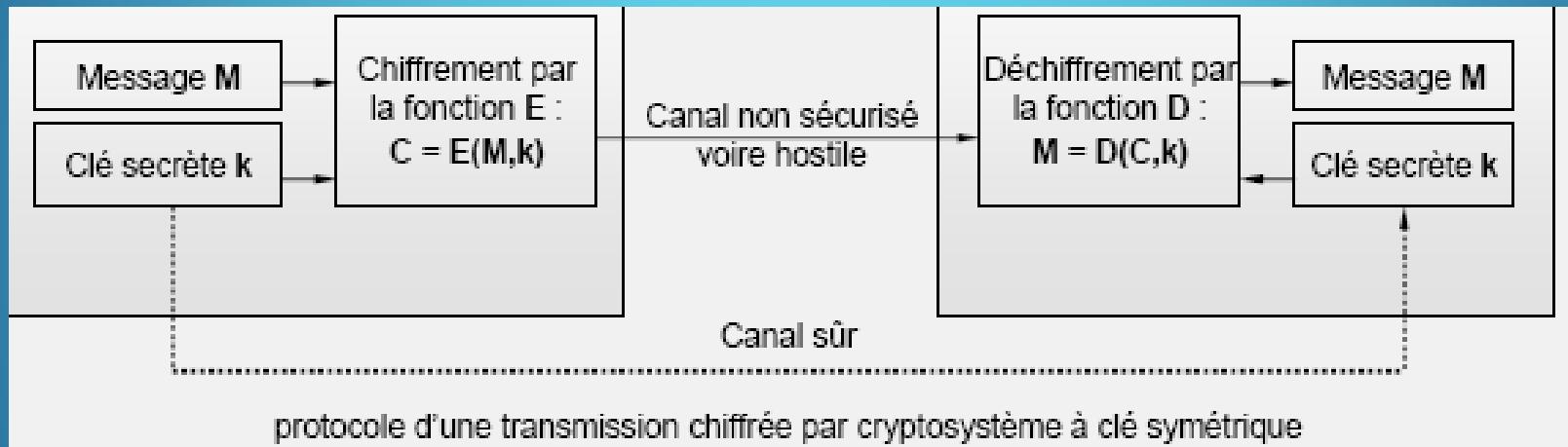
- clé de 40 bits
- chiffrement matériel : 300 Kbits/sec
- chiffrement logiciel : 21,6 Kbits/sec
- Inconvénient majeur : un pirate substitue sa propre clé publique à celle du destinataire, il peut alors intercepter et décrypter le message pour le recoder ensuite avec la vraie clé publique et le renvoyer sur le réseau. «L'attaque» ne sera pas décelée.
- usage : on ne les emploiera que pour transmettre des données courtes telles que les clés privées et les signatures électroniques.

## ► DES

- clé de 56 bits
- chiffrement matériel : 300 Mbits/sec
- chiffrement logiciel : 2,1 Mbits/sec
- Inconvénient majeur : attaque «brute force» rendue possible par la puissance des machines.
- Usage : chiffrement rapide, adapté aux échanges de données de tous les protocoles de communication sécurisés.

# RSA | ÉCHANGE SÉCURISÉ

- ▶ Résolution du problème de l'échange des clés secrètes : utilisation d'une méthode **hybride** combinant à la fois chiffrement **symétrique** et **asymétrique**.



- ▶ Avantages :
  - ▶ la clé secrète est chiffrée et échangée ;
  - ▶ après l'échange on bascule le chiffrement en utilisant un algorithme symétrique plus rapide ;
  - ▶ on démarre l'échange avec l'utilisation d'un algorithme asymétrique qui possède l'avantage d'offrir un moyen d'identifier les interlocuteurs.

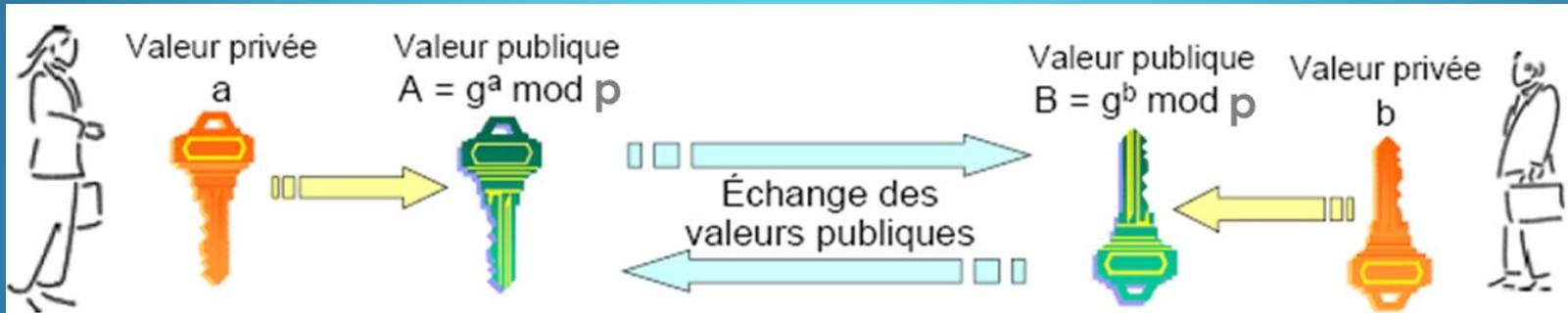
# DIFFIE-HELLMAN KEY AGREEMENT PROTOCOL

- ▶ 1976 : Deffie et Hellman ont proposé un crypto-système à clé publique
- ▶ Il s'agit d'un algorithme pour échanger des clés secret (pas de sécuriser les données)
- ▶ Utilise des algorithmes discrets
- ▶ Il est facile à calculer l'exponentiel modulo (nombre premier)
- ▶ Impossible de calculer l'inverse

# DIFFIE-HELLMAN KEY AGREEMENT PROTOCOL

- ▶ Établir une clé commune via un canal publice
- ▶  $p$  : nombre premier,  $g$  : générateur publiques
- ▶ Basé sur deux problèmes difficiles
  - ▶ Discrete Log Problem (DLP) : étant donné  $(g, h, p)$  calculer  $a$  tel que  $g^a \equiv h \pmod{p}$
  - ▶ Computational Diffie Hellman (CDH) : étant donné  $g^a \pmod{p}$  et  $g^b \pmod{p}$  calculer  $g^{ab} \pmod{p}$

- ▶ Choix de nombre primitive « p » et du générateur « g »
- ▶ Calcule et échange de valeurs publiques



- ▶ Génération d'un secret partagé :

$$K = B^a \text{ mod } p = A^b \text{ mod } p = g^{ab} \text{ mod } p$$



- ▶ Un tiers (espion) ne peut pas reconstituer le secret partagé à partir des valeurs publiques

# FONCTION DE HACHAGE

# PRINCIPE GÉNÉRAL D'UNE FONCTION DE HACHAGE

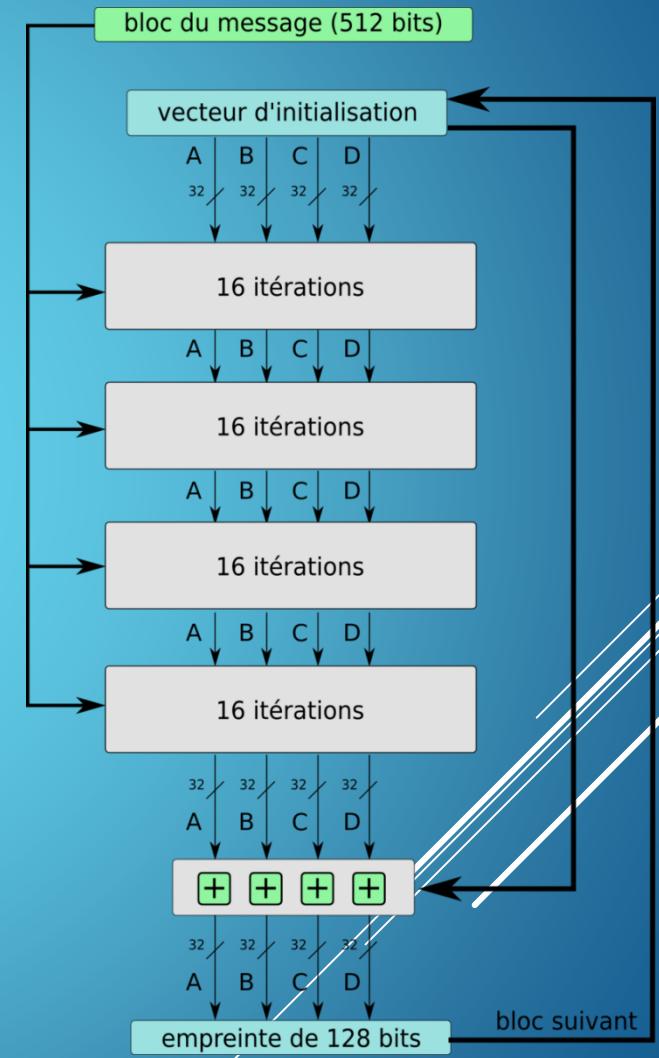
- ▶ Entrée: message  $M$  avec contenu et taille arbitraire.
- ▶ Sortie: message de taille fixe  $h=H(M)$ .
- ▶  $H(M)$  est appelé condensât ou Hash code, ou empreinte, ou fingerprint, ou message digest
- ▶ Irréversible:
  - ▶ Étant donnée  $h$ , il est difficile de trouver  $x$  tel que:  $h = H(x)$
  - ▶ Complexité de l'ordre de  $2^n$ ,  $n$  est le nombre de bits du digest.
- ▶ Résistance forte à la collision:
  - ▶ Étant donné  $x$ , il est impossible de trouver  $y$  avec  $H(x) = H(y)$
  - ▶ Il est impossible de trouver une paire  $x, y$  tel que  $H(x) = H(y)$
- ▶ Calcul facile et rapide (plus rapide que le cryptage symétrique).

# EXEMPLE DE FONCTIONS DE HASHAGE

- ▶ **MD5 : Message Digest 5**
  - ▶ Développé par Ron Rivest
  - ▶ Basé sur MD4
  - ▶ Génère une empreinte de taille 128 bits
- ▶ **SHA : Secure Hash Algorithm**
  - ▶ Développé par NIST en collaboration avec NSA
  - ▶ Génère une empreinte de taille 160 bits
- ▶ **Exemple: Hash (md5)**
  - ▶ ENSA DE KHOURIBGA →  
**8528D9FB19987E0A1A9B2F31D6CB4B27**
  - ▶ ENSA De KHOURIBGA →  
**35C8897975D5A4BD03A5088866A28EA7**

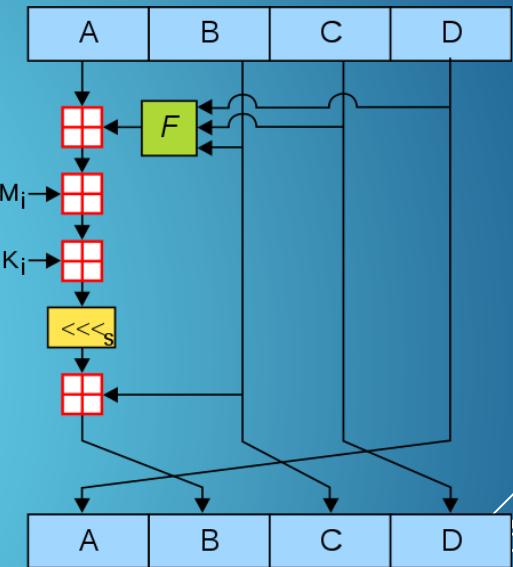
# MD5 | ÉTAPES GLOBALES

1. Compléter par (100...0) jusqu'à atteindre un multiple de 512Bits moins 64bits (Ajoutés dans l'étape 2).
2. Ajouter les 64bits qui représentent la taille du message d'origine modulo  $2^{64}$ .
3. Initialiser les 4 tampons (Buffers) A, B, C, D.
4. L'opération est effectué par Bloc de 512
5. Le résultat des quatre étapes est additionné au vecteur d'initialisation.
6. Le résultat final est la concaténation des 4 tampons.
7. S'il y a encore un autre block de 512bits, le résultat obtenu précédemment sera le vecteur d'initialisation pour le traitement du block actuel. Sinon, ce résultat représentera le Code MD5



# MD5 | TRAITEMENT PAR ÉTAPE

- ▶ Sortie de chaque étape
  - ▶  $A \leftarrow B \boxplus ((A \boxplus Fonction(B, C, D)) \boxplus M[i] \boxplus T[i] \lll s)$
  - ▶  $M[i]$  : Une partie du 512bits à Hacher (**128-bit state**)
  - ▶  $K[i]$  : constante de 32bits unique pour chaque itération.
  - ▶ **Fonctions**( $A, B, C$ ) ? | 1→F, 2→G, 3→H, 4→I
    - ▶ Chaque étape utilise une fonction différente :
      1.  $F(B, C, D) = (B \wedge C) \vee (\neg B \wedge D)$
      2.  $G(B, C, D) = (B \wedge D) \vee (C \wedge \neg D)$
      3.  $H(B, C, D) = (B \oplus C \oplus D)$
      4.  $I(B, C, D) = C \oplus (B \vee \neg D)$
- ▶ Ces opérations sont répétées 16 fois/étape



<<< $s$  : Décalage circulaire de  $s$ -bits  
 $\wedge$  : ET logique n-aire  
 $\vee$  : OU logique n-aire  
 $\neg$  : NON (Négation)  
 $\oplus$  : OU EXCLUSIF  
 $\boxplus$  : Addition modulo  $2^n$

# MD5 | CONSTANTS K[i], S ET VALEURS D'INITIALISATION

## ► valeurs d'initialisation:

A0 := 0x67452301

B0 := 0xefcdab89

C0 := 0x98badcfe

D0 := 0x10325476

## ► K[i]

Calculé par :  $K[i] = \text{floor}(2^{32} \times \text{abs}(\sin(i + 1)))$  avec i= 0 .. 63.

K[ 0.. 3] := { 0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee }

K[ 4.. 7] := { 0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501 }

K[ 8..11] := { 0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be }

K[12..15] := { 0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821 }

K[16..19] := { 0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa }

K[20..23] := { 0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fb8 }

K[24..27] := { 0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed }

K[28..31] := { 0xa9e3e905, 0xfcfa3f8, 0x676f02d9, 0x8d2a4c8a }

K[32..35] := { 0xffffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c }

K[36..39] := { 0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xebefbc70 }

K[40..43] := { 0x289b7ec6, 0xea127fa, 0xd4ef3085, 0x04881d05 }

K[44..47] := { 0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665 }

K[48..51] := { 0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039 }

K[52..55] := { 0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1 }

K[56..59] := { 0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1 }

K[60..63] := { 0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 }

## ► s

s[ 0..15] := { 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22 }

s[16..31] := { 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20 }

s[32..47] := { 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23 }

s[48..63] := { 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21 }

# SHA | SECURE HASH ALGORITHME

- ▶ SHA est une fonction de hachage cryptographique conçue par la National Security Agency des États-Unis (NSA). Elle produit un résultat (appelé « hash » ou condensat) (taille de 160 bits pour SHA-1).
- ▶ Le SHA-1 est le successeur du SHA-0 qui a été rapidement mis de côté pour des raisons de sécurité insuffisante.
- ▶ Le SHA-1 prend un message d'un maximum de 128 bits en entrée.
- ▶ Quatre fonctions booléennes sont définies, elles prennent 3 mots de 32 bits en entrée et calculent un mot de 32 bits.
- ▶ Depuis 2010, de nombreuses organisations ont recommandé son remplacement par SHA-2 ou SHA-3. Microsoft, Google et Mozilla ont annoncé que leurs navigateurs respectifs cesseraien d'accepter les certificats SHA-1 au plus tard en 2017.
- ▶ D'autres versions de SHA sont considérées sûres: SHA-256, SHA-384, SHA-512 ...

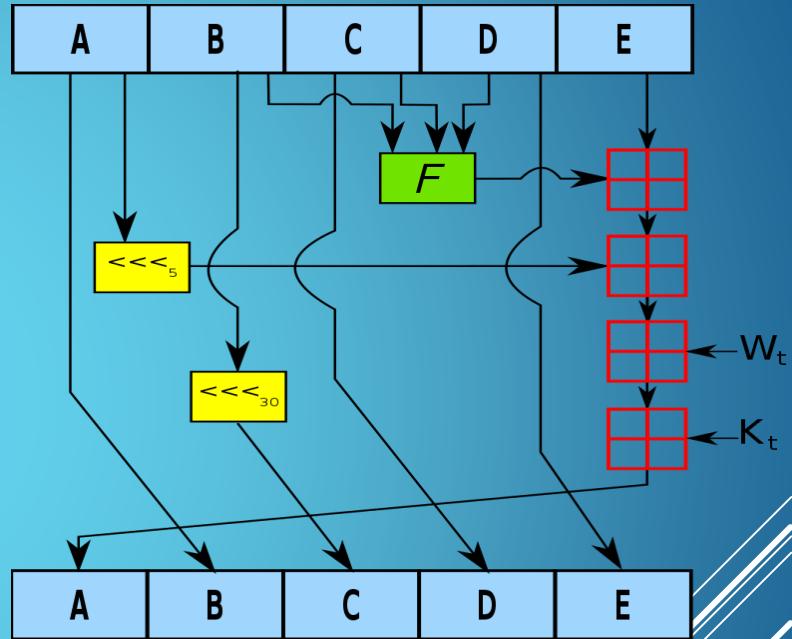
# CARACTÉRISTIQUES DE SHA-1

- ▶ **taille du message :  $2^{64}$  bits maximum**
- ▶ **taille des blocs : 512 bits**
- ▶ **taille des mots : 32 bits**
- ▶ **taille du condensé : 160 bits**
- ▶ **niveau de sécurité : collision en  $2^{63}$  opérations.**

# SHA-1 | TRAITEMENT PAR ÉTAPE

- ▶ Un tour de la fonction de compression de SHA-1 :
- ▶ A, B, C, D, E sont des mots de 32 bits ;
- ▶  $<<<n$  désigne une rotation des bits par décalage de n bits vers la gauche ;
- ▶ F est une fonction qui dépend du numéro de tour t ;
- ▶  $\boxplus$  est l'addition modulo  $2^{32}$ .
- ▶  $K_t$  est une constante (32 bits) qui dépend du numéro de tour t ;
- ▶  $W_t$  est un mot de 32 bits qui dépend du numéro de tour t ; il est obtenu par une procédure d'expansion à partir du bloc de donnée (512 bits) dont le traitement est en cours.

$$f_t(x, y, z) = \begin{cases} Ch(x, y, z) = (x \wedge y) \vee (\neg x \wedge z), & \text{si } 0 \leq t \leq 19 \\ Parity(x, y, z) = x \oplus y \oplus z, & \text{si } 20 \leq t \leq 39 \\ Maj(x, y, z) = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z), & \text{si } 40 \leq t \leq 59 \\ Parity(x, y, z) = x \oplus y \oplus z, & \text{si } 60 \leq t \leq 79 \end{cases}$$



# SHA-1 | CONSTANTES ET CALCULE

## ▶ Initialisation des buffers

$$H_0^{(0)} = 0x67452301$$

$$H_1^{(0)} = 0xefcdab89$$

$$H_2^{(0)} = 0x98badcfe$$

$$H_3^{(0)} = 0x10325476$$

$$H_4^{(0)} = 0xc3d2e1f0$$

$H_t^{(i)}$  : mot (w bits) n° t, du bloc (m bits) n° i,  
du message M

## ▶ Constantes d'étape

$$K_t = \begin{cases} 0x5a827999, & \text{si } 0 \leq t \leq 19 \\ 0xed9eba1, & \text{si } 20 \leq t \leq 39 \\ 0x8f1bbcdc, & \text{si } 40 \leq t \leq 59 \\ 0xca62c1d6, & \text{si } 60 \leq t \leq 79 \end{cases}$$

## ▶ Calcule des mots d'étape

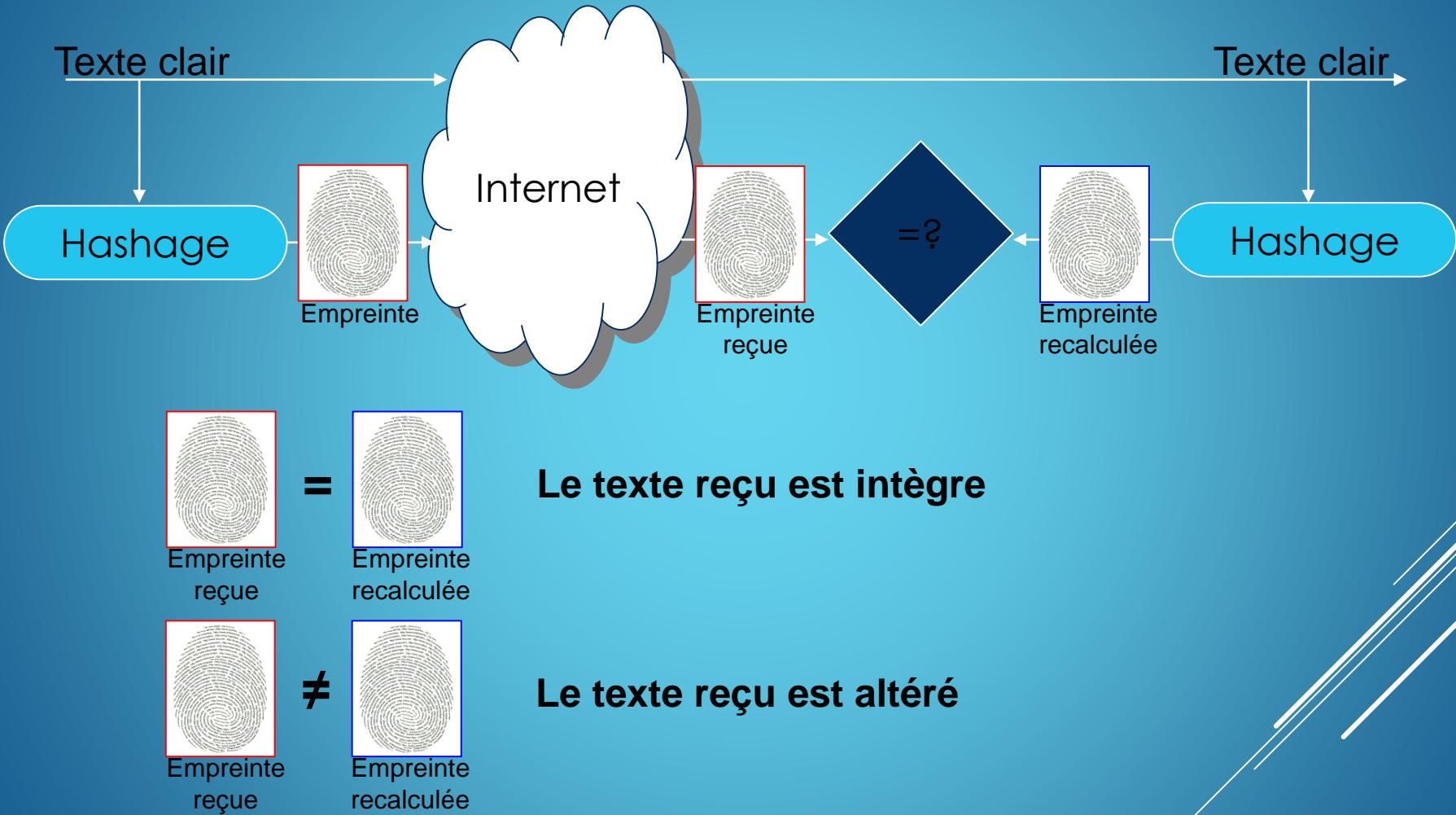
$$W_t = \begin{cases} M_t^{(i)}, & 0 \leq t \leq 15 \\ ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}), & 16 \leq t \leq 79 \end{cases}$$

$M_t^{(i)}$  : bloc (m=512 bits) n° i, du message M

# SHA-512 | ÉTAPES GLOBALES

1. Compléter 1024 Bits (ou ces multiples) moins 128bits (Ajoutés dans l'étape 2).
2. Ajouter les 128bits de représentation (taille du Message d'origine)
3. Initialiser les 8 Buffers : A, B, C, D, E, F, G et H en Hexadécimal.
4. L'opération est effectué par Bloc de 1024 bits et sur des mots de taille 64bites
5. Le résultat des 80 étapes est additionné au vecteur d'initialisation.
6. Le résultat final est la concaténation des 8 buffers (512bits au total).
7. S'il y a encore un autre block de 1024 bits, le résultat obtenu précédemment sera le vecteur d'initialisation pour le traitement du block actuel. Sinon, ce résultat représentera le Code SHA

# PRINCIPES DES FONCTIONS DE HACHAGE

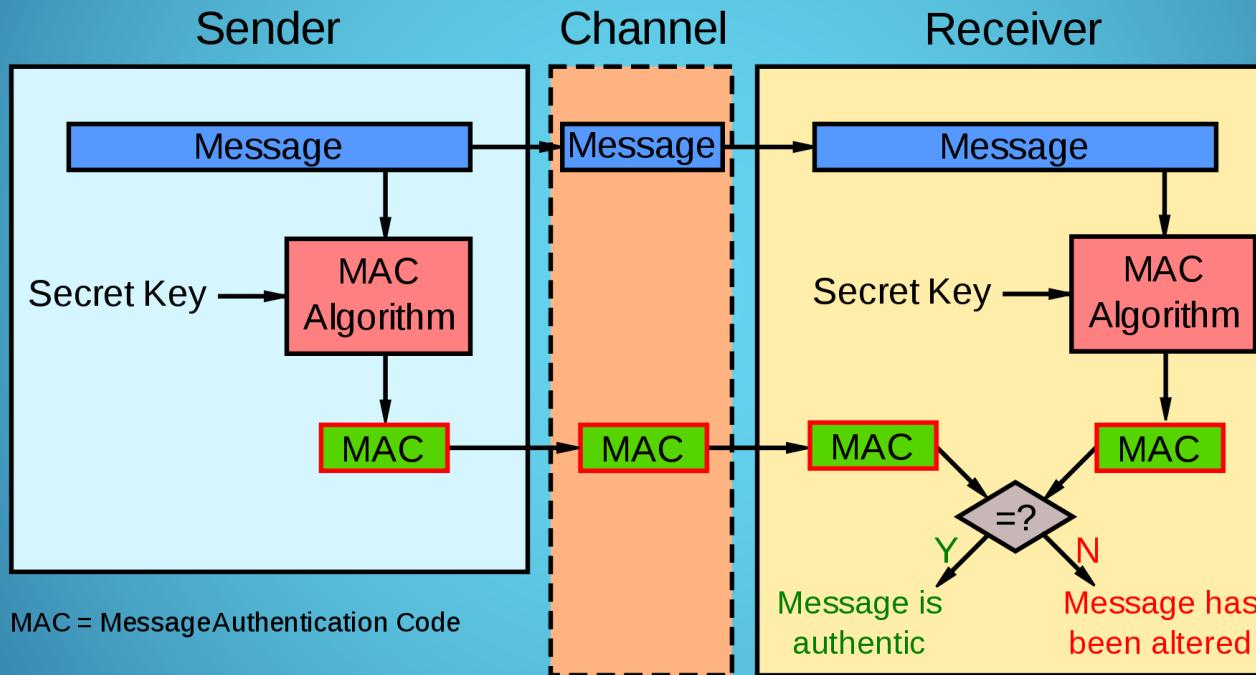


# HACHAGE | ATTAQUE

- ▶ **Outils de Hachage**
  - ▶ <https://www.fileformat.info/tool/hash.htm>
- ▶ **Calculateur de force du mots passe (Haché)**
  - ▶ <https://www.grc.com/haystack.htm>
- ▶ **Outils d'Audit (Attaque)**
  - ▶ <https://crackstation.net>
- ▶ **Salting (« Sel » algorithmes Privés à chaque organisme)**
  - ▶ Ajout de séquence de caractères insérées dans des endroits spécifique du mot de passe avant le Hachage.
  - ▶ Le résultat est Haché
  - ▶ Attaque par dictionnaire et texte Brute devient très difficiles

# SERVICE D'AUTHENTIFICATION DE MESSAGE

# MAC | CODE D'AUTHENTIFICATION DE MESSAGE (MESSAGE AUTHENTICATION CODE)



# TYPES DE SERVICE D'AUTHENTIFICATION DE MESSAGE

- ▶ Authentification par Code de Message d'Authentification (**MAC**)
  - ▶ Utilisation d'un Message de Longueur Fixe obtenue à partir du **message** et d'une **clé**. (Authentification & Intégrité)
- ▶ Authentification par Cryptage de Message (CMAC)
  - ▶ Authentificateur → **Cryptogramme**
  - ▶ Cryptage Symétrique (Confidentialité & Authentification)
  - ▶ Cryptage Asymétrique (Confidentialité OU Authentification)
  - ▶ Double Cryptage Asymétrique (Confidentialité & Authentification)
- ▶ Authentification par **Fonctions de Hachage (HMAC)**
  - ▶ Utilisation d'un Message de Longueur Fixe obtenue à partir du **message seul**. (Intégrité des données)
- ▶ Certificat Numérique (Combinaison de la plupart des méthodes précédentes)

# HMAC | KEYED-HASH MESSAGE AUTHENTICATION CODE

- ▶ Un HMAC, est un type de code d'authentification de message « MAC », calculé en utilisant une fonction de hachage cryptographique en combinaison avec une clé secrète. Comme avec n'importe quel MAC, il peut être utilisé pour vérifier simultanément l'intégrité de données et l'authenticité d'un message.
- ▶ N'importe quelle fonction itérative de hachage, comme MD5 , SHA-1, SHA-2 ... peut être utilisée dans le calcul d'un HMAC ; le nom de l'algorithme résultant est HMAC-MD5 ou HMAC-SHA-1.
- ▶ La qualité cryptographique du HMAC dépend de la qualité cryptographique de la fonction de hachage et de la taille et la qualité de la clé.
- ▶ Une fonction de hachage itérative découpe un message en blocs de taille fixe et itère dessus avec une fonction de hachage. Par exemple, MD5 et SHA-1 opèrent sur des blocs de 512 bits. La taille de la sortie HMAC est la même que celle de la fonction de hachage (128 ou 160 bits dans les cas du MD5 et SHA-1), bien qu'elle puisse être tronquée si nécessaire.

# HMAC | CONSTRUCTION

- ▶ La fonction HMAC :

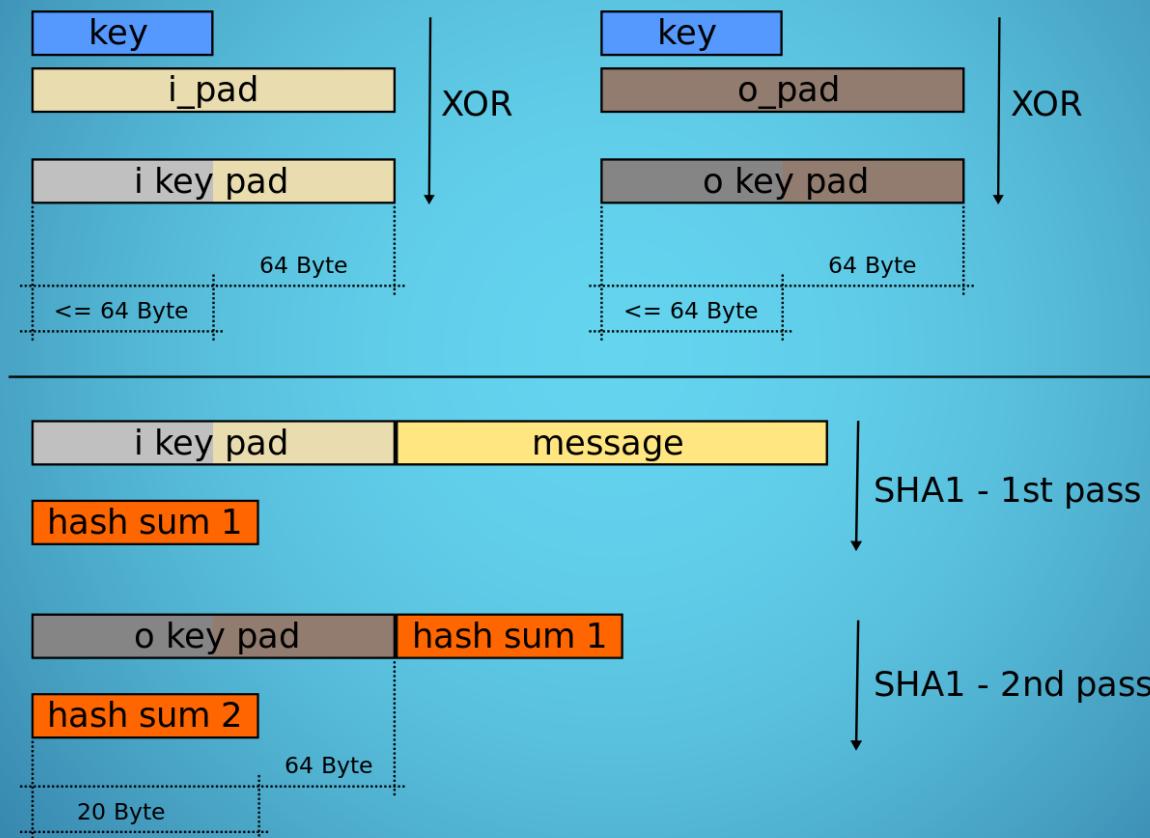
$$\text{HMAC}_K(m) = h\left((K \oplus opad) \parallel h\left((K \oplus ipad) \parallel m\right)\right)$$

avec :

- ▶  $h$  : une fonction de hachage,
- ▶  $K$  : la clé secrète complétée avec des zéros pour qu'elle atteigne la taille de bloc de la fonction  $h$
- ▶  $m$  : le message à authentifier,
- ▶ «  $\parallel$  » désigne une concaténation
- ▶ «  $\oplus$  » est un « ou » exclusif,
- ▶ «  $ipad$  » définie par : 0x363636...3636 (taille égale à celle d'un bloc)
- ▶ «  $opad$  » définie par : 0x5c5c5c...5c5c (taille égale à celle d'un bloc)

Exemple : si la taille de bloc de la fonction de hachage est 512 bits, «  $ipad$  » et «  $opad$  » sont 64 répétitions des octets, respectivement, 0x36 et 0x5c.

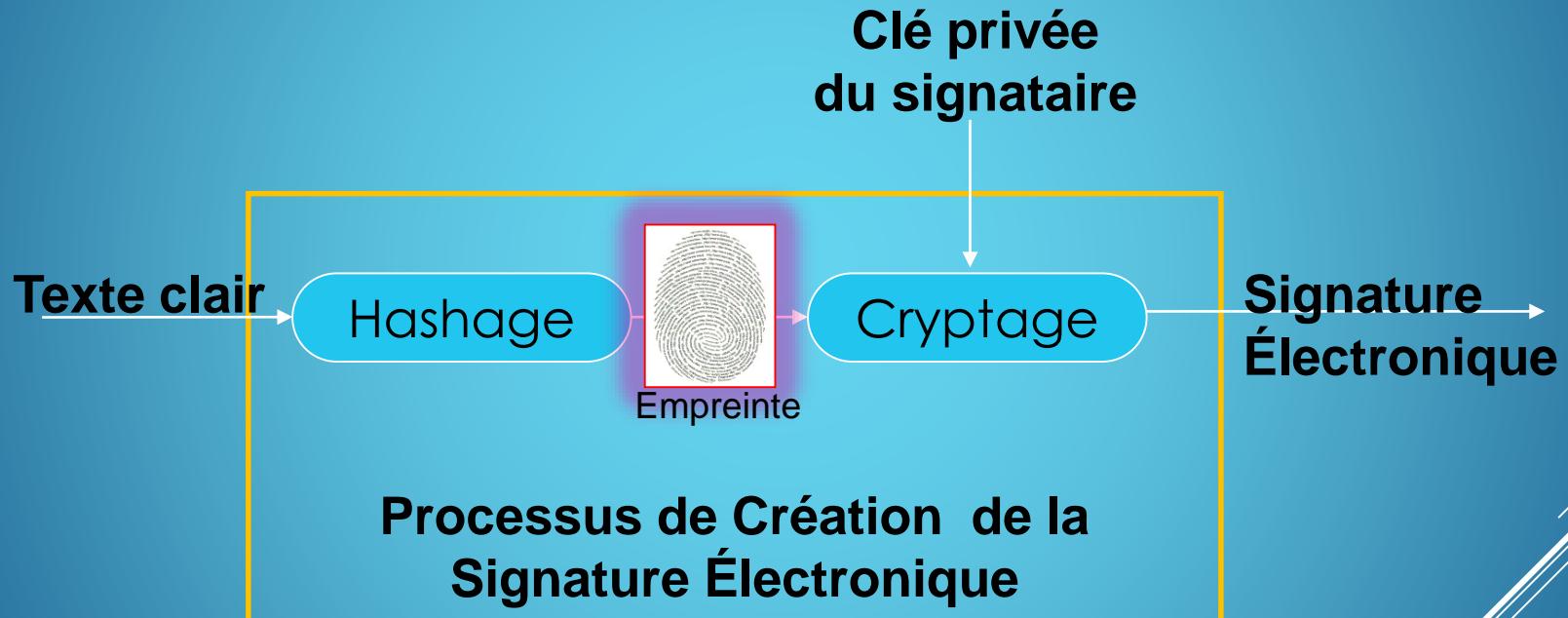
# HMAC | CONSTRUCTION



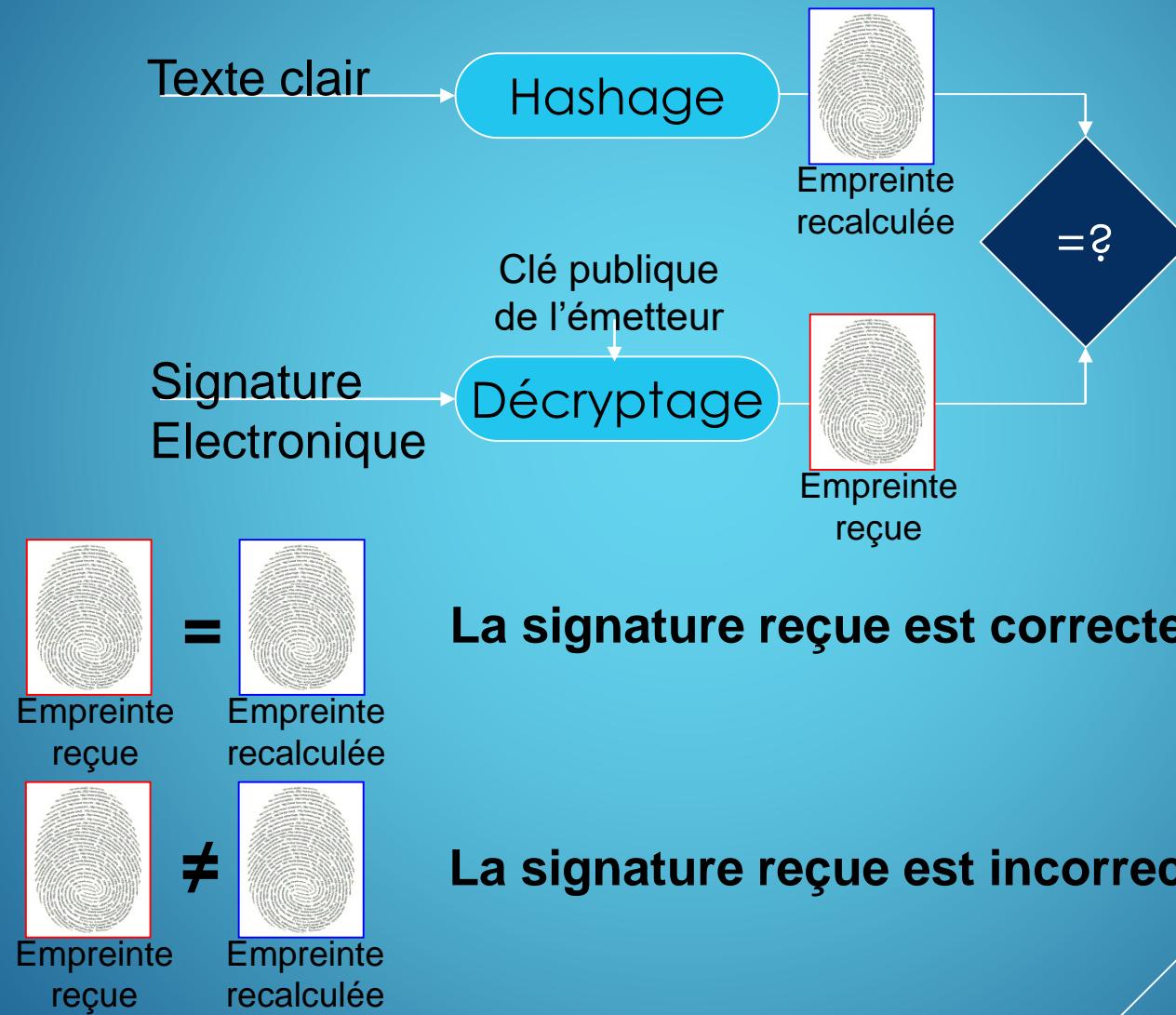
# SIGNATURE NUMÉRIQUE

- ▶ Idée clé:
  - ▶ Le Hash (résultat de la fonction de hachage) d'un message est crypté avec la clé privée de l'émetteur.
  - ▶ La clé publique est utilisée pour la vérification de la signature
- ▶ Soit:
  - ▶ M: message à signer, H: fonction de hachage
  - ▶ Kpr, Kpu: paire de clés privée / publique de l'émetteur.
  - ▶ E / D: fonction de cryptage / Décryptage en utilisant Kpu / Kpr.
- ▶ En recevant ( $M$ ,  $E_{Kpr}(H(M))$ ), le récepteur vérifie si:  
 $H(M)=D_{Kpu}(E_{Kpr}(H(M)))$

# SIGNATURE ÉLECTRONIQUE : CRÉATION



# VÉRIFICATION DE LA SIGNATURE NUMÉRIQUE



# SIGNATURE NUMÉRIQUE

- ▶ La signature permet de mettre en œuvre les services:
  - ▶ Intégrité du message
  - ▶ Authentification
  - ▶ Non-répudiation
  - ▶ Génération d'une clé de chiffrement symétrique pour le service de Confidentialité

# CERTIFICATS NUMÉRIQUE

# PROBLÉMATIQUE DES CLÉS PRIVÉES/PUBLIQUES

- ▶ Dans une architecture à clés publiques :
  - ▶ Les clés **privées** doivent être **générées et stockées** de manière sûre
  - ▶ Les clés **publiques** doivent être **diffusées** le plus largement possible.
- ▶ Lorsqu'on utilise **la clé** d'un correspondant, il est nécessaire de s'assurer :
  - ▶ qu'elle **appartient** bien à ce correspondant
  - ▶ que l'émetteur de cette clé est digne de **confiance**
  - ▶ qu'elle est toujours **valide**

# NOTION DE CONFIANCE

- ▶ Besoin de confiance sur ce que prétend être le correspondant.
- ▶ La limite de la cryptographie à clé publique réside dans la confiance que l'on donne aux informations échangées (clés publiques).
- ▶ La confiance est obtenue en associant au bi-clef un certificat délivré et géré par un tiers de confiance.



# SOLUTION

- ▶ Charger une autorité de signer les clés publiques
- ▶ Elle crypte une empreinte de la clé publique de la personne avec la clé privée de l'autorité.
- ▶ On s'assure de la provenance de la clé publique en vérifiant la signature qui y a été apposée avec la clé publique de l'autorité de certification.
- ▶ Une clé publique signée par un tiers de confiance est appelée Certificat.

# CERTIFICAT NUMÉRIQUE

- ▶ Un certificat est un document électronique émis par une tierce partie de confiance qui permet de garantir l'authenticité d'une clé publique.
- ▶ Il correspond à l'association de :
  - ▶ la clé publique
  - ▶ l'identité de son propriétaire
  - ▶ l'usage qui peut être fait de la clé
- ▶ Un certificat est l'équivalent à la légalisation des signatures manuelle.
- ▶ Il est délivré par une Autorité de Certification.

# AUTORITÉ DE CERTIFICATION



## « Tiers de confiance »

L'autorité de certification (AC) est une autorité centrale responsable de certifier les clés publiques des utilisateurs



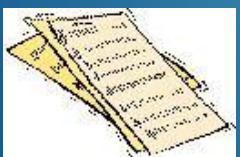
## Possède un bi-clé certifié

- Génération des certificats
- Authentification auprès d'autres autorités de certification



## Révoque les certificats

- Perte/vol/date de péremption
- Compromission de clefs

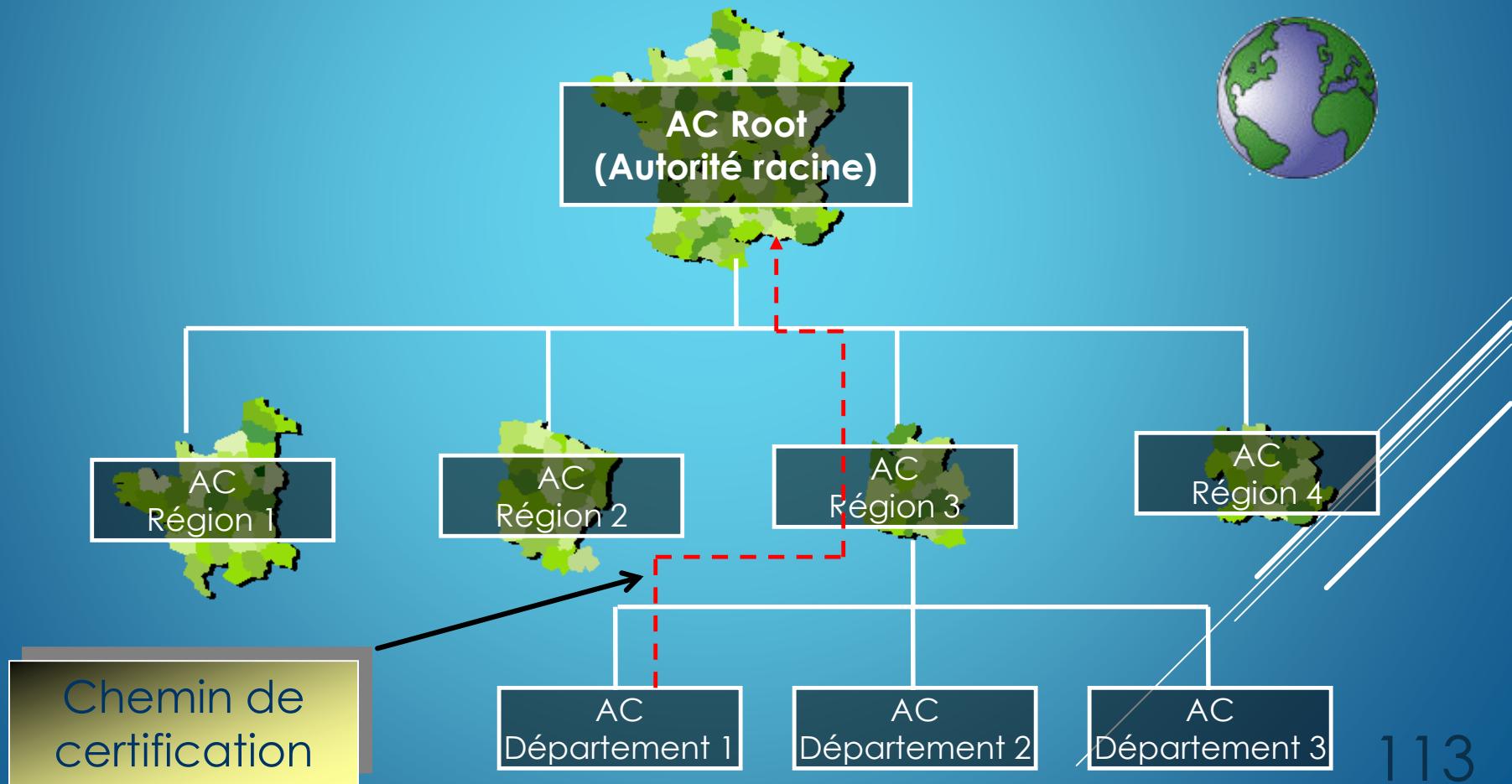


## Définit les règles d'attribution des certificats

- Politique de certification (« facteur humain » )

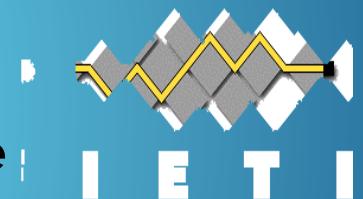
# LA HIÉRARCHIE DES AUTORITÉS DE CERTIF.

- ▶ L'Autorité de Certification racine est le point de confiance.

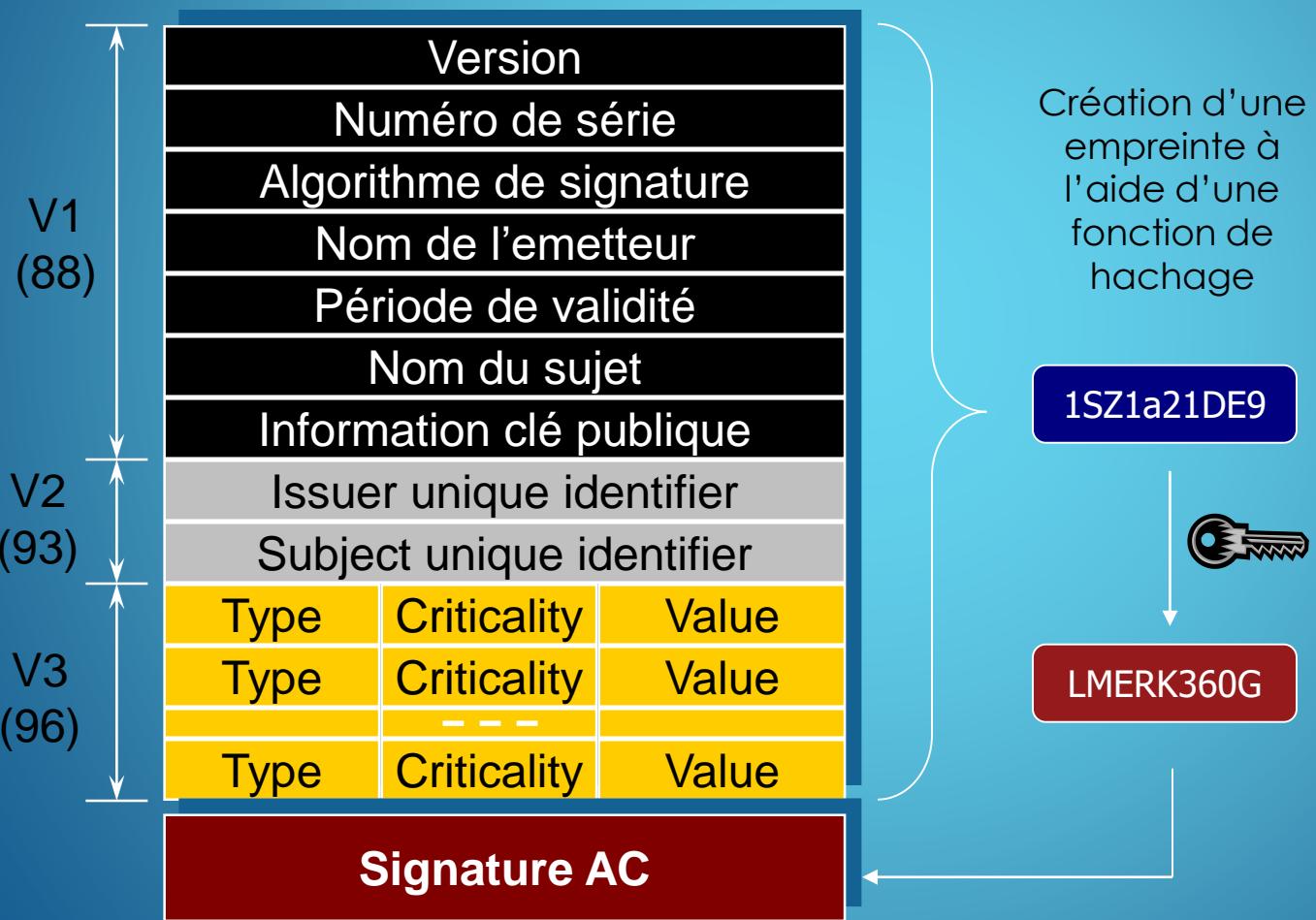


# LE STANDARD X.509

- ▶ Principal format utilisé pour les certificats
- ▶ Basé sur la norme X.500
- ▶ Normalisé IETF dans la RFC 2459
- ▶ « Internet X.509 Public Key Infrastructure Certificate and CRL Profile »
- ▶ Versions successives
  - ▶ 1988 : V1
  - ▶ 1993 : V2 = V1 + 2 nouveaux champs
  - ▶ 1996 : V3 = V2 + extensions



# FORMAT D'UN CERTIFICAT X.509 V3



# PUBLIC KEY INFRASTRUCTURE (PKI)

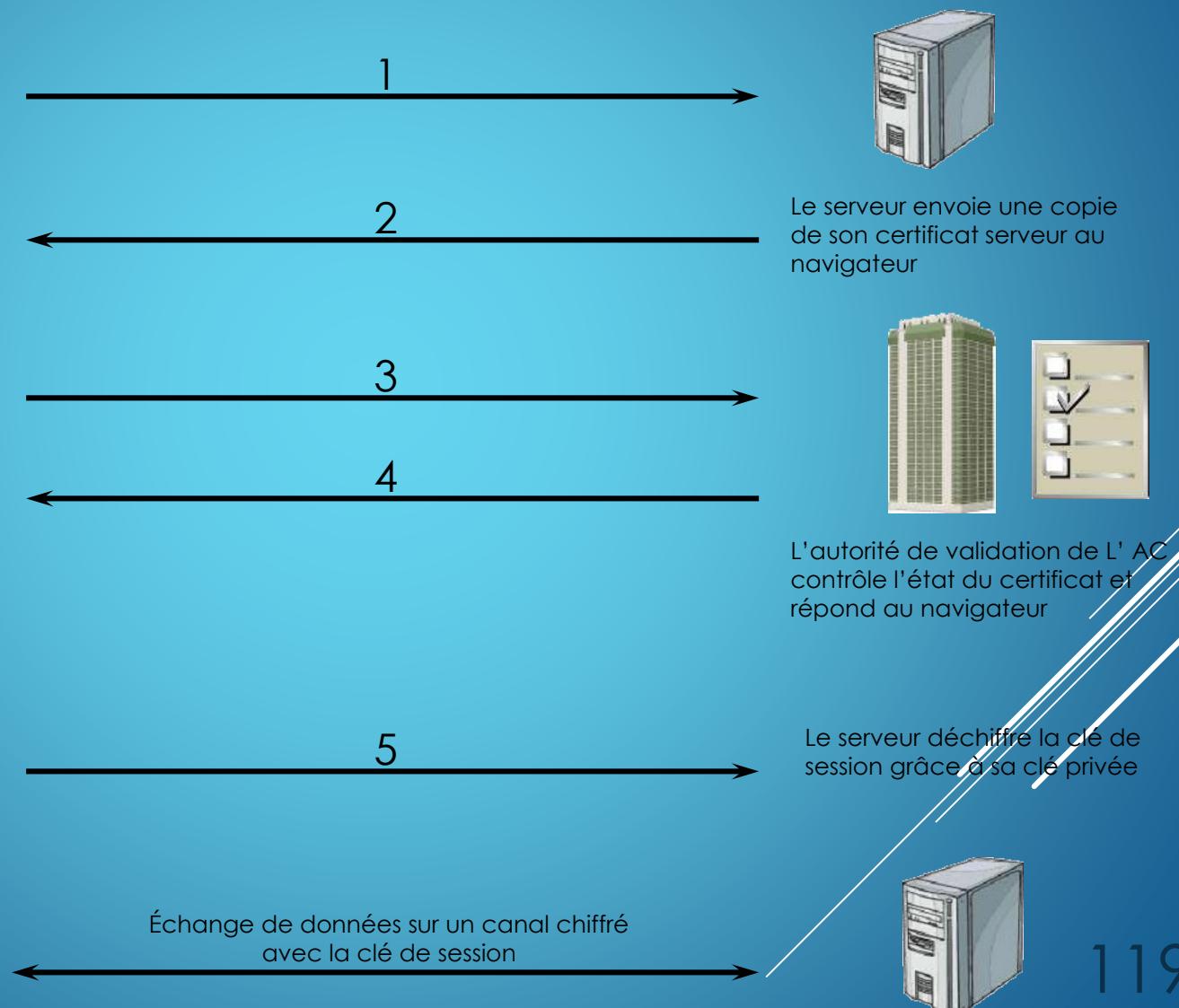
**PKI « *Public Key Infrastructure* »**

**IGC « *Infrastructures de Gestion de Clefs* »**

**ICP « *Infrastructures à Clefs Publiques* »**

- ▶ C'est un nom donné aux infrastructures permettant la mise en œuvre de la cryptographie à clés publiques. L'objectif est de créer un environnement de confiance
- ▶ Cette infrastructure constitue un ensemble de moyens matériels, logiciels et organisationnels, nécessaires pour déployer à grande échelle un système cryptographique basé sur les certificats X.509.
- ▶ Un standard : Certificats X.509

# EXEMPLE : SSL 2.0 ET HTTP (HTTPS)



# EXEMPLE : TLS ET HTTP (HTTPS)

