SR06 : Systèmes et réseaux avancés

Cryptographie avancée

Walter SCHÖN

Cryptographie : Définitions (1/3)

« Il y a deux sortes de cryptographie dans ce monde :

- Celle qui empêche votre petite sœur de lire vos fichiers
- Celle qui empêche des gouvernements puissants de lire vos fichiers

Nous allons parler de cette dernière… »

(D'après le livre de Bruce Schneier : Applied Cryptography, J. Wiley & Sons) : la référence incontournable sur le sujet, à laquelle ce cours doit beaucoup...

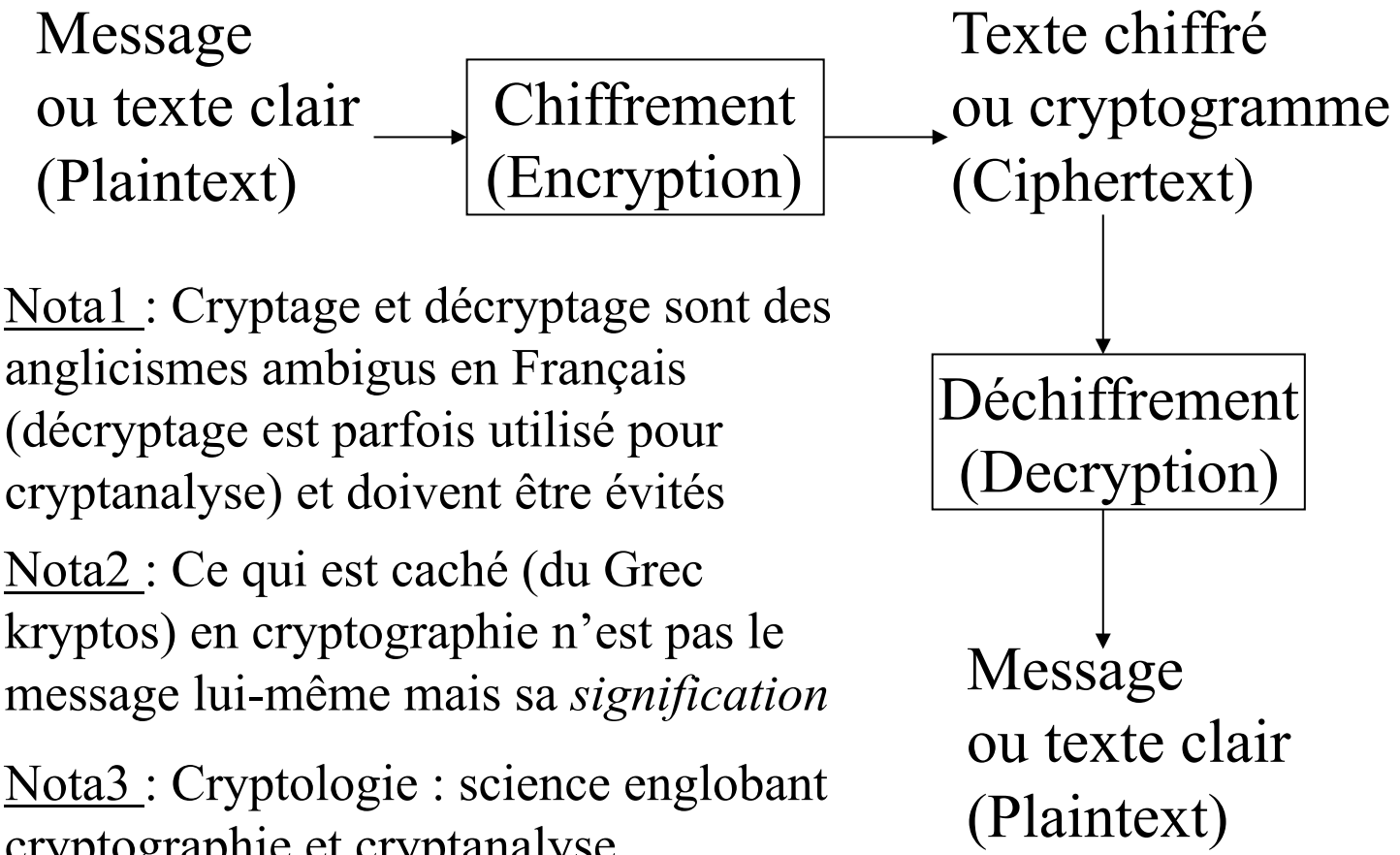
Cryptographie : Définitions (2/3)

Cryptographie : Science ayant pour but la *sécurité* des informations.

Sécurité : *Confidentialité* essentiellement, mais aussi :
Authentification (identité de l'émetteur),
Intégrité (protection contre des modifications),
Non répudiation (impossibilité pour l'émetteur d'un message de nier l'avoir envoyé)

Cryptanalyse : Science cherchant à passer outre la sécurisation apportée par les procédés de cryptographie (en exploitant leurs faiblesses)

Cryptographie : Définitions (3/3)

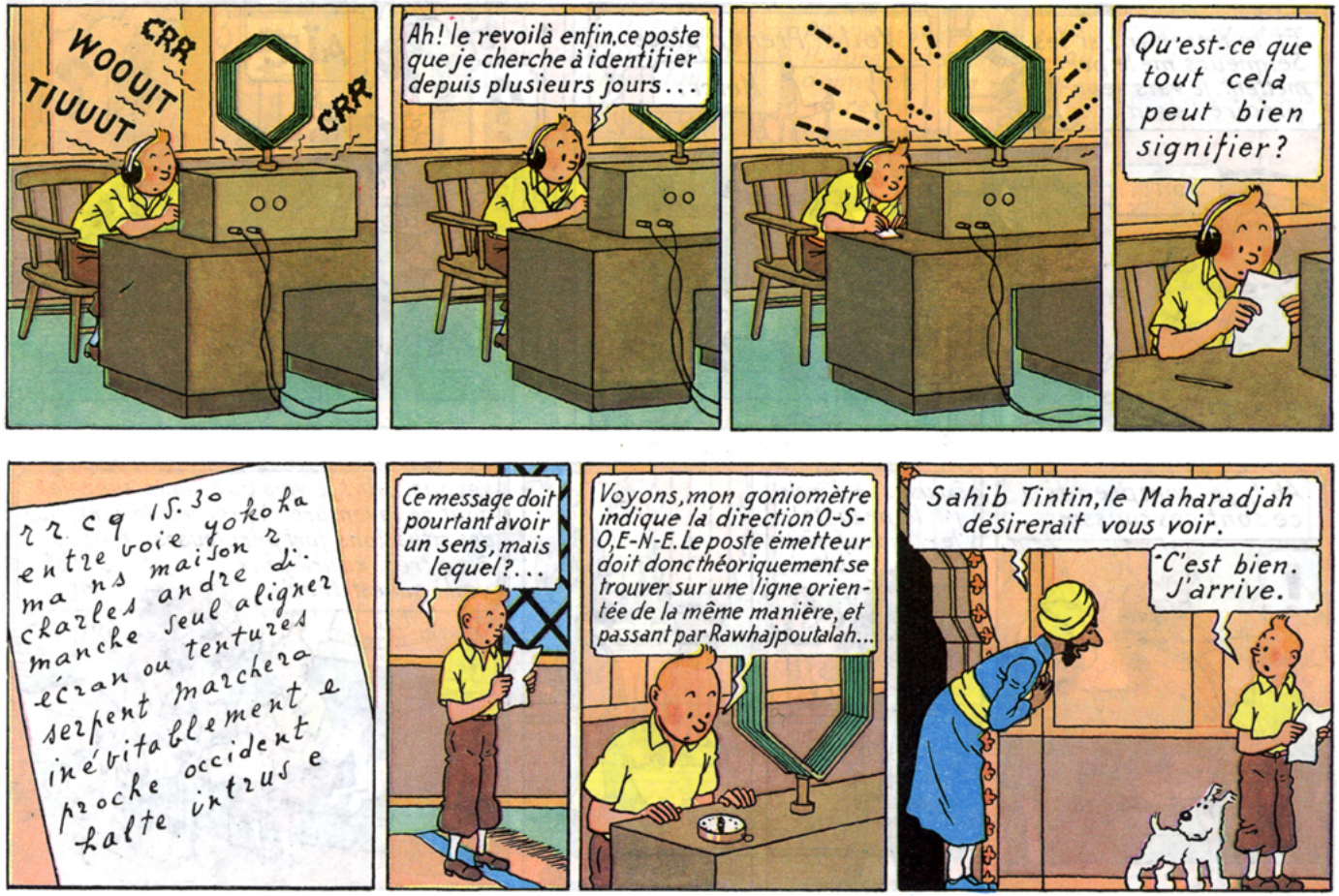


Nota1 : Cryptage et décryptage sont des anglicismes ambigus en Français (décryptage est parfois utilisé pour cryptanalyse) et doivent être évités

Nota2 : Ce qui est caché (du Grec kryptos) en cryptographie n'est pas le message lui-même mais sa *signification*

Nota3 : Cryptologie : science englobant cryptographie et cryptanalyse

Cryptographie vs Stéganographie



Stéganographie (1/3)

Stéganographie (du Grec steganos : couvert) : message caché (par un procédé secret) dans un autre d'apparence anodine

Un cryptogramme n'a pas une apparence anodine, c'est en général un inextricable charabia (gibberish).

Stéganographie (2/3)

Je suis très émue de vous dire que j'ai bien compris l'autre soir que vous aviez toujours une envie folle de me faire danser. Je garde le souvenir de votre baiser et je voudrais bien que ce soit là une preuve que je puisse être aimée par vous. Je suis prête à vous montrer mon affection toute désintéressée et sans calcul, et si vous voulez me voir aussi vous dévoiler sans artifice mon âme toute nue, venez me faire une visite. Nous causerons en amis, franchement.

Je vous prouverai que je suis la femme sincère, capable de vous offrir l'affection la plus profonde comme la plus étroite en amitié, en un mot la meilleure preuve que vous puissiez rêver, puisque votre âme est libre. Pensez que la solitude où j'habite est bien longue, bien dure et souvent difficile. Ainsi en y songeant j'ai l'âme grosse. Accourrez donc vite et venez me la faire oublier par l'amour où je veux me mettre.

(Lettre attribuée à George Sand pour Alfred de Musset)

Stéganographie (3/3)

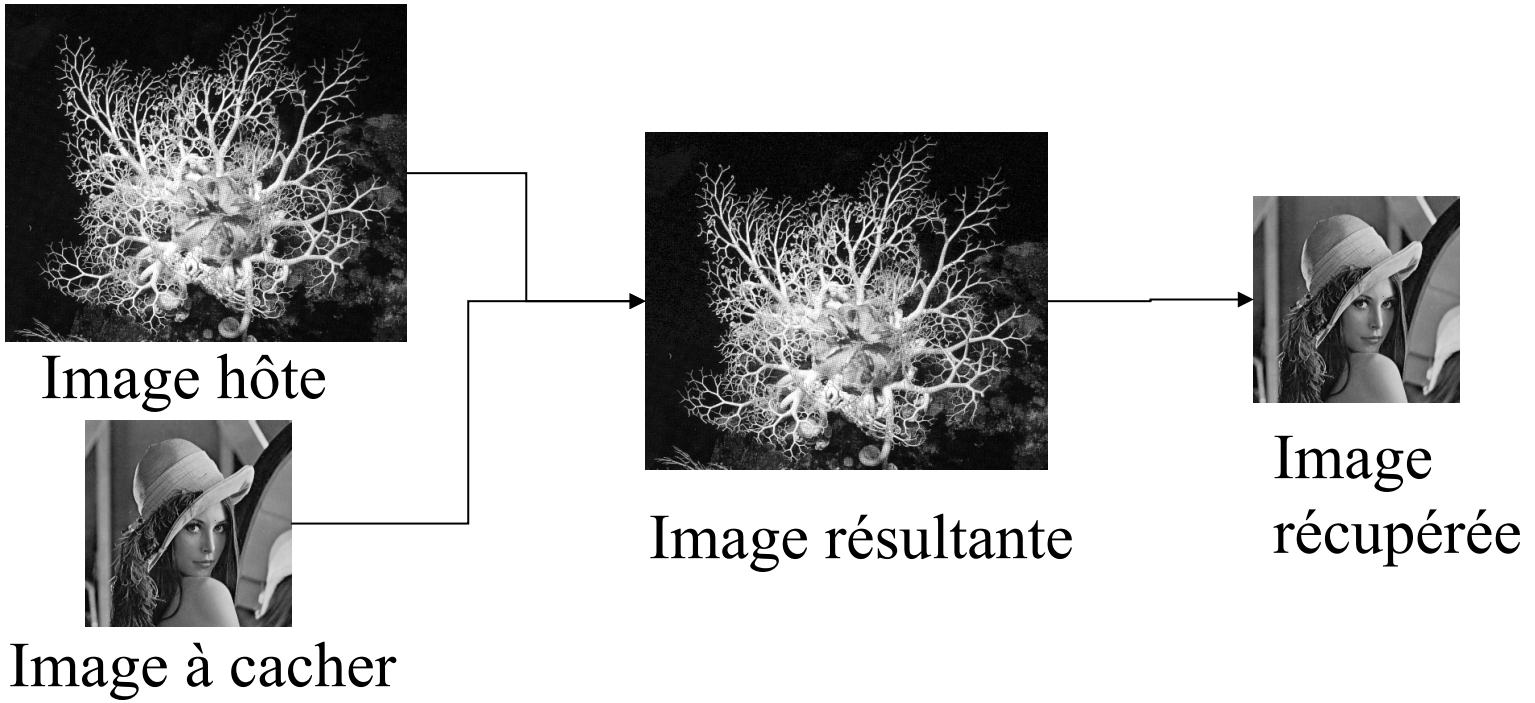
Je suis très émue de vous dire que j'ai bien compris l'autre soir que vous aviez toujours une envie folle de me faire danser. Je garde le souvenir de votre baiser et je voudrais bien que ce soit là une preuve que je puisse être aimée par vous. Je suis prête à vous montrer mon affection toute désintéressée et sans calcul, et si vous voulez me voir aussi vous dévoiler sans artifice mon âme toute nue, venez me faire une visite. Nous causerons en amis, franchement.

Je vous prouverai que je suis la femme sincère, capable de vous offrir l'affection la plus profonde comme la plus étroite en amitié, en un mot la meilleure preuve que vous puissiez rêver, puisque votre âme est libre. Pensez que la solitude où j'habite est bien longue, bien dure et souvent difficile. Ainsi en y songeant j'ai l'âme grosse. Accourrez donc vite et venez me la faire oublier par l'amour où je veux me mettre.

(Lettre attribuée à George Sand pour Alfred de Musset)

La stéganographie aujourd'hui

L'image numérique est très utilisée comme message anodin porteur (ex : remplacement de bits de poids faible par ceux du message). Sert entre autres au tatouage (watermarking).



Cryptographie à algorithme restreint

Si la sécurité repose sur le fait que l'algorithme ce chiffrement reste secret on parle de cryptographie à algorithme restreint.

Les inconvénients sont alors les mêmes que ceux de la stéganographie :

- Chaque groupe d'utilisateurs doit inventer son propre algorithme,
- Le groupe doit comporter un bon cryptographe (pour avoir une idée de la robustesse de l'algorithme),
- Le groupe doit réaliser par lui-même le matériel/logiciel de chiffrement/déchiffrement
- Il faut changer d'algorithme si une personne quitte le groupe

Cryptographie : algorithmes à clés

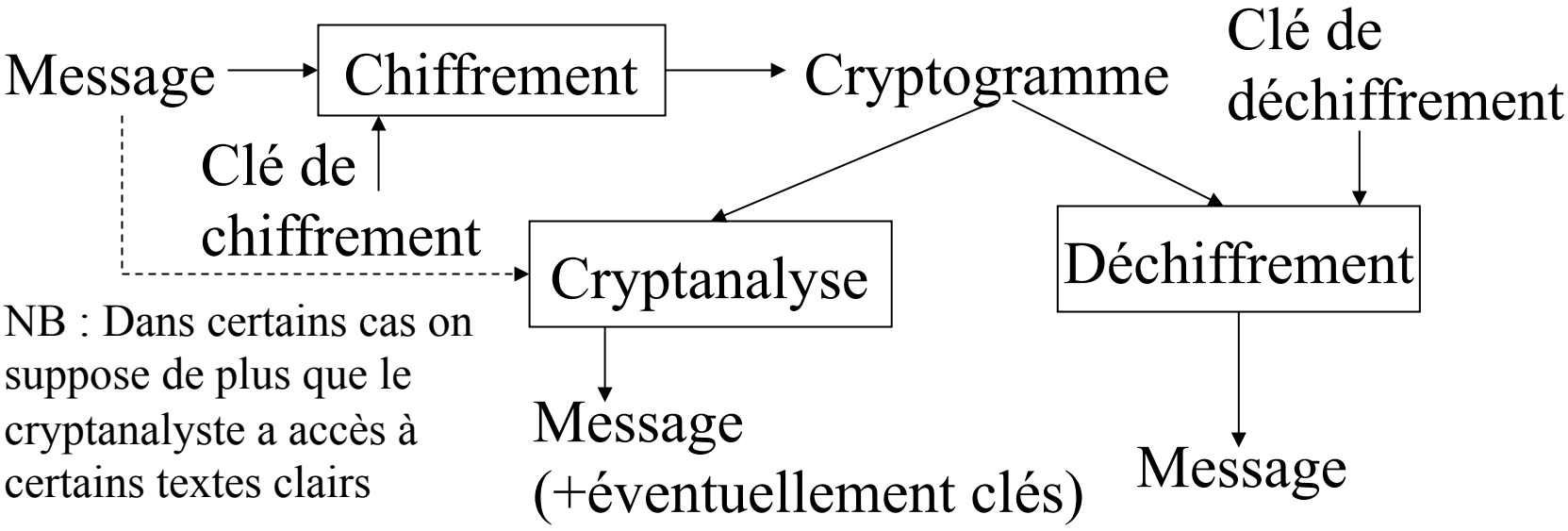
Les algorithmes de chiffrement/déchiffrement dépendent de *clés* (pouvant prendre de nombreuses valeurs), l'algorithme n'a pas besoin d'être secret (il est le plus souvent public) la sécurité repose entièrement sur les clés. Avantages :

- Utilisation du même algorithme par plusieurs groupes
- Robustesse testée par la communauté mondiale des cryptographes
- Matériels/Logiciels disponibles sur le marché
- Simple changement de clés si une personne quitte le groupe

Algorithmes à clés : cryptanalyse

Par hypothèse les indiscrets ont accès aux textes chiffrés et connaissent l'algorithme. Ils n'ignorent que les clés.

La cryptanalyse consiste alors à reconstituer texte clair (et éventuellement clés) sans connaître par avance les clés.



Algorithmes à clés : types

- Algorithmes symétriques ou à clé secrète : clé de chiffrement et de déchiffrement identiques (ou pouvant être déduites facilement l'une de l'autre) donc secrète. Peuvent manipuler un bit (ou un caractère) à la fois : chiffrement en continu (stream cipher), ou opérer par blocs (64 ou 128 bits) : chiffrement par blocs (block cipher).
- Algorithmes asymétriques ou à clé publique : clé de chiffrement (clé publique : non secrète) différente de la clé de déchiffrement (clé privée : secrète et ne pouvant être simplement déduite de la clé publique).

Histoire de la cryptographie

- Vers -1900 : premier cryptogramme connu (inscription égyptienne à hiéroglyphes volontairement non conformes)
- Vers -600 : Chiffre d'Atbash (hébreux), premier chiffre à substitution (inversion d'alphabet) : A->Z, B->Y…Z->A
- Vers -500 : Scytale (Grèce), premier chiffre à transposition

Histoire de la cryptographie : Chiffre de César et dérivés (1/3)

- Vers -50 : Substitution par simple décalage d'alphabet
- La valeur (fixe) du décalage est la clé (César utilisait 3)
- Il y a donc seulement 26 clés possibles donc cryptanalyse facile voire ludique
- Encore utilisé sur Internet avec la clé 13 (ROT13) pour éviter la lecture involontaire (solutions…)
  ROT13(ROT13(M)) = M

MKHI YTVBEX

⇩

TROP FACILE

Histoire de la cryptographie : Chiffre de César et dérivés (2/3)

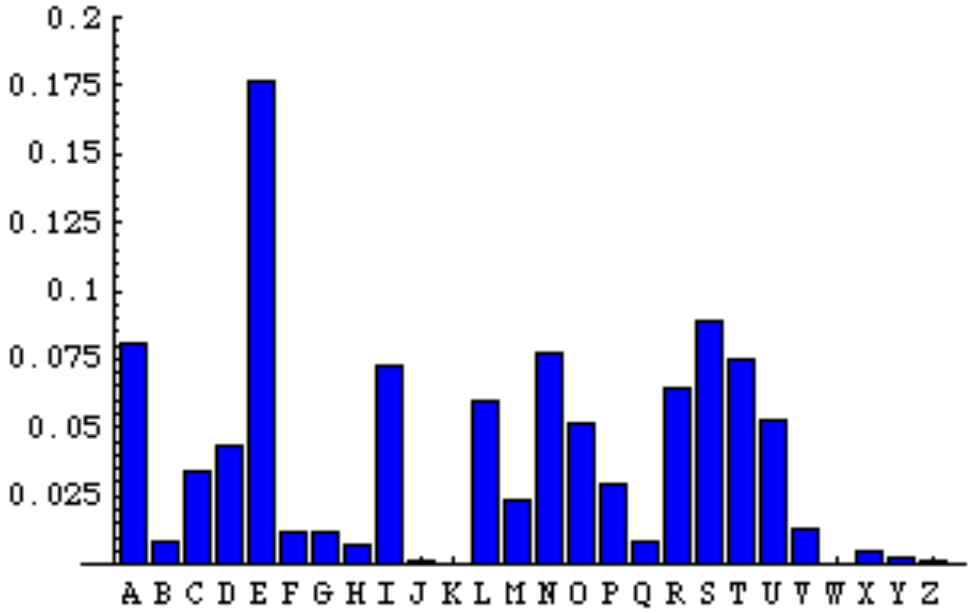
- On peut rendre le système de César plus robuste par création d'un alphabet mélangé : **ABCDEFGHIJKLMNOPQRSTUVWXYZ**
  MPLOKINJUBHYVGTCFRXDEWSZQA

Un tel système fut utilisé à la Renaissance. Il y a dans ce cas 26! soit 4.10^{26} clés possibles, la difficulté étant de les retenir :

Histoire de la cryptographie : Chiffre de César et dérivés (3/3)

- L'apparente robustesse liée au nombre de clés est illusoire. Un tel système ne résiste pas (sur un message chiffré suffisamment long) à une analyse des fréquences. C'est le propre des substitutions monoalphabétiques.

La comparaison avec l'histogramme des symboles du texte chiffré permet de retrouver l'alphabet (correspondance lettre claire <-> symbole chiffré)



Histoire de la cryptographie : Substitutions polyalphabétiques

- Pour échapper à la cryptanalyse par fréquences, différents systèmes polyalphabétiques ont été créés aux 15ème et 16ème siècle. Le plus connu est le chiffre de Vigenère (1596), obtenu en «additionnant» (A = 0 décalage, B = 1 décalage…) le texte à chiffrer avec un texte clé répété autant de fois que nécessaire :

```
  TO BE OR NOT TO BE
+ CL EC LE CLE CL EC
= VZ FG ZV PZX VZ FG
```

Revient à utiliser autant d'alphabets que de caractères de la clé, l'alphabet utilisé étant fonction de la position

Histoire de la cryptographie : Cryptanalyse des substitutions

- La cryptanalyse du chiffre de Vigenère reste très facile si la clé est de longueur faible par rapport à celle du texte.
- Une séquence de 2 lettres identiques a en effet de fortes chances d'être une séquence de 2 lettres identiques du texte clair, chiffrée avec la clé dans la même position. Le décalage correspondant est alors un multiple de la longueur de clé :

VZ FG ZV PZX **VZ FG** Longueur de clé : diviseur de 9

- Une fois la longueur de clé déterminée (grâce à plusieurs indices de ce type) l'analyse des fréquences menées sur les lettres positionnées de la même manière par rapport à la clé permet de trouver texte clair et clé.

Histoire de la cryptographie : Cryptanalyse des transpositions

- Une méthode semblable permet la cryptanalyse des transpositions rectangulaires comme la scytale

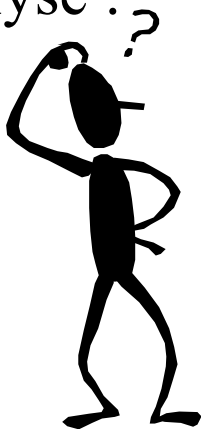
MEET ME
AFTER SCH
OOL

MAOEFOETLTE R M ES C H

Former les séquences de 2 lettres (digrammes) séparées d'un pas donné
Analyser les fréquences
Si ces fréquences coïncident avec celles du langage du message, on a le pas correct

Histoire de la cryptographie : Chiffre de Gilbert Vernam (1/2)

- Le chiffre de Vigenère peut être rendu *parfait* en utilisant une clé aléatoire à usage unique de longueur égale à celle du message (masque jetable ou One Time Pad : Vernam 1917)
- Si cette clé est bien *aléatoire* (les fonctions rand ne le sont pas, pas plus que les pages de romans parfois utilisés comme clés), *et utilisée une seule fois*, il n'y a (et n'y aura jamais) aucun moyen de cryptanalyse :




```
  TPRQSMVZYSTIJRV          TPRQSMVZYSTIJRV
- IPAMDYIHUOBPWDI        - SBEHESEXROUNVXD
= LAREPONSEESTNON        = BONJOURCHEZVOUS
```

Histoire de la cryptographie : Chiffre de Gilbert Vernam (2/2)

- La difficulté réside dans la réalisation du masque réellement aléatoire de très grande longueur. Si le masque n'est pas aléatoire (périodique ou pris dans un roman) un bon cryptanalyste le détectera certainement si le message est assez long :

```
  TPRQSMVZYSTIJRV
- CLENONALEATOIRE
= RENDEZVOUSAUBAR
```

- Le chiffre de Gilbert Vernam fut toutefois (paraît il) utilisé dans le téléphone rouge...

Histoire de la cryptographie : Le chiffre Allemand ADFGVX

- Pratiqué à la fin de la première guerre
- Dernier chiffre pratiqué (et cryptanalysé) sans machine
- Combine une substitution et une transposition
- N'utilise que les lettres ADFGVX suffisamment différentes en code Morse

Le chiffre Allemand ADFGVX : La substitution

- Type « carré de Polybe » (Nom de son inventeur vers 150 avant J.C)
- Code les 26 lettres + 10 chiffres à partir d'un tableau 6x6 :

	A	**D**	**F**	**G**	**V**	**X**
A	Q	Y	A	L	S	E
D	Z	C	R	X	H	0
F	F	O	4	M	8	7
G	3	I	T	G	U	K
V	P	D	6	2	N	V
X	1	5	J	9	W	B

RENFORT
COMPIEGNE
16H10

⇨

DF AX VV FA FD DF GF
DD FD FG VA GD AX
GG VV AX XA VF DV
XA DX

La grille définit un alphabet mélangé qui est un des deux éléments de la clé

Le chiffre Allemand ADFGVX : La transposition

- Le cryptogramme intermédiaire est rangé sous un mot, second élément de la clé, et les colonnes réarrangées en ordonnant par ordre alphabétique

D	**E**	**M**	**A**	**I**	**N**
D	F	A	X	V	V
F	A	F	D	D	F
G	F	D	D	F	G
V	A	G	D	A	X
G	G	V	V	A	X
X	A	V	F	D	V
X	A	D	X		

⇨

A	**D**	**E**	**I**	**M**	**N**
X	D	F	V	A	V
D	F	A	D	F	F
D	G	F	F	D	G
D	V	A	A	G	X
V	G	G	A	V	X
F	X	A	D	V	V
X	X	A		D	

⇨

Cryptogramme définitif :

XDFVA VDFAD
FFDGF FDGDV
AAGXV GGAVX
FXADV VXXAD

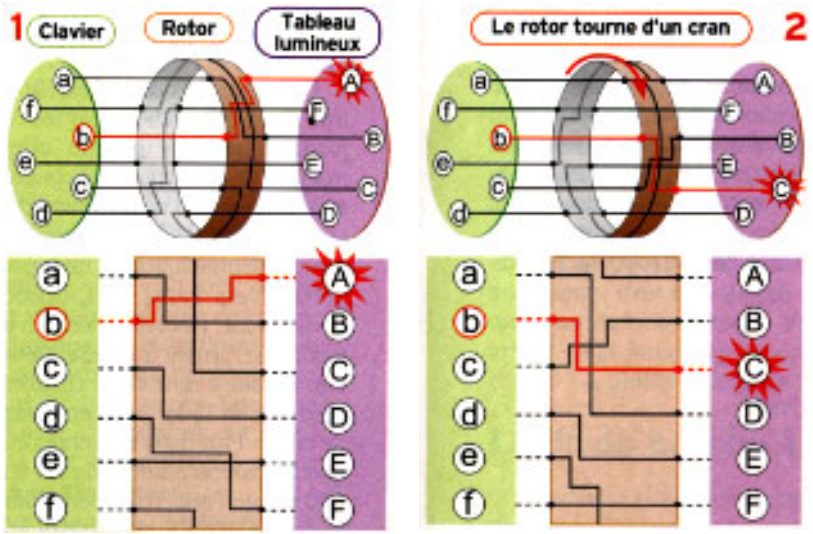
Le chiffre Allemand ADFGVX : La cryptanalyse

- Bien que très compliqué pour son époque ADFGVX fut cryptanalysé par Georges PAINVIN. Qui parvint ainsi à déchiffrer le 3 Juin 1918 le « Radiogramme de la victoire » destiné à une unité située près de Noyon.

« Hâter l'approvisionnement en munitions, le faire même de jour tant qu'on n'est pas vu »

L'offensive Allemande vers Compiègne le 9 Juin 1918 fut stoppée net grâce à ce message...

Histoire de la cryptographie : La machine Enigma



Fut utilisé dans le crypt d'UNIX

- Utilisée par l'Allemagne Nazie : 3 rotors avec alphabet mélangé câblé changé à chaque caractère par rotation des rotors. Equivalent à un masque de période 26*26*26 = 17576. Fut cryptanalysée avec succès par les alliés…

Cryptographie moderne : OU exclusif (XOR)

- L'électronique numérique a rendu possible la réalisation de matériels et logiciels de cryptographie. Les plus simples utilisent un simple OU Exclusif bit à bit entre le texte à chiffrer et un mot clé répété autant de fois que nécessaire :

Chiffrement

Clair	**A (Code 65)**	⊕ **01000001**	$0 \oplus 0 = 0 \quad 1 \oplus 1 = 0$
Clé	**y (Code 121)**	**01111001**	$1 \oplus 0 = 1 \quad 0 \oplus 1 = 1$
Chiffré	**8 (Code 56)**	= **00111000**	

Déchiffrement

Chiffré	**8 (Code 56)**	**00111000**	
Clé	**y (Code 121)**	⊕ **01111001**	
Clair	**A (Code 65)**	= **01000001**	

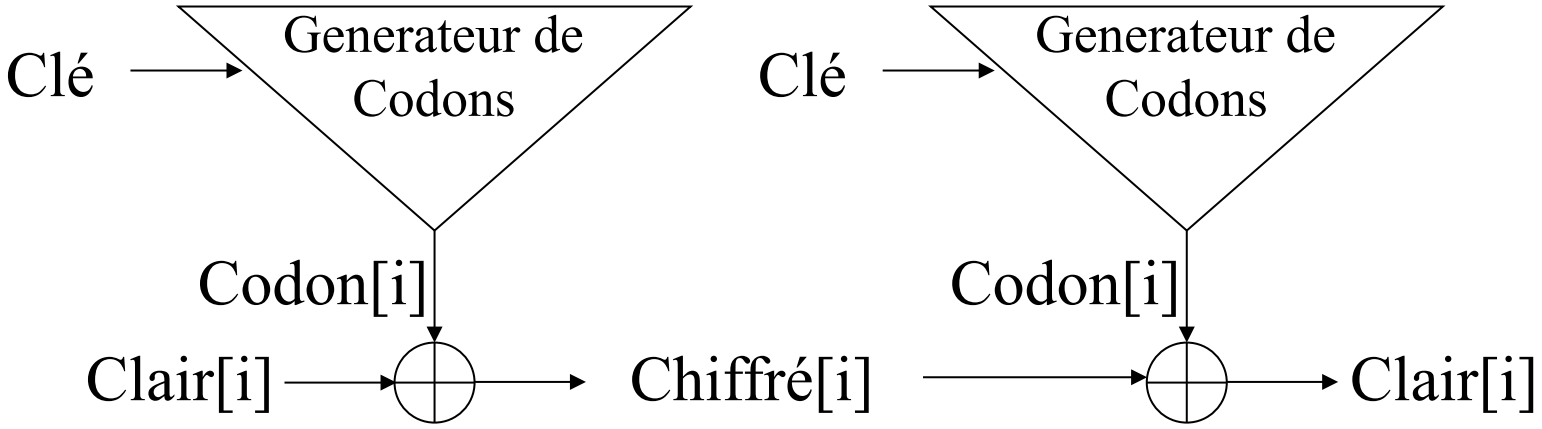
Avantage :
Symétrie du XOR :

$\text{Chiffré} = \text{Clair} \oplus \text{Clé}$

$\text{Clair} = \text{Chiffré} \oplus \text{Clé}$

Chiffrement en continu (Stream Cipher) (1/2)

- Le OU exclusif n'est rien d'autre qu'une substitution polyalphabétique type Vigenère => Aucune sécurité avec des clés courtes et/ou non aléatoires.
- Le OU exclusif avec un générateur de codons (Keystream generator « flot de clés ») : générateur de bits pseudo-aléatoires sûr d'un point de vue cryptographique est la base du chiffrement en continu (Stream Cipher) :



Chiffrement en continu (Stream Cipher) (2/2)

- Le chiffrement en continu est surtout utilisé dans les communications codées.
- Les générateurs de codons côté émetteur et récepteur doivent absolument être synchronisés (=> sensibilité à la perte de bits dans la communication). Certains générateurs sont à auto-synchronisation (leur état dépend d'un nombre certain nombre de bits du message chiffré, ce qui limite les conséquence de la perte d'un bit).
- Le générateur de codons peut être implémenté en matériel (Linear Feedback Shift Register : registre à décalages à rétroaction linéaire) ou en logiciel. On décrira ici l'algorithme le plus répandu à ce jour : RC4

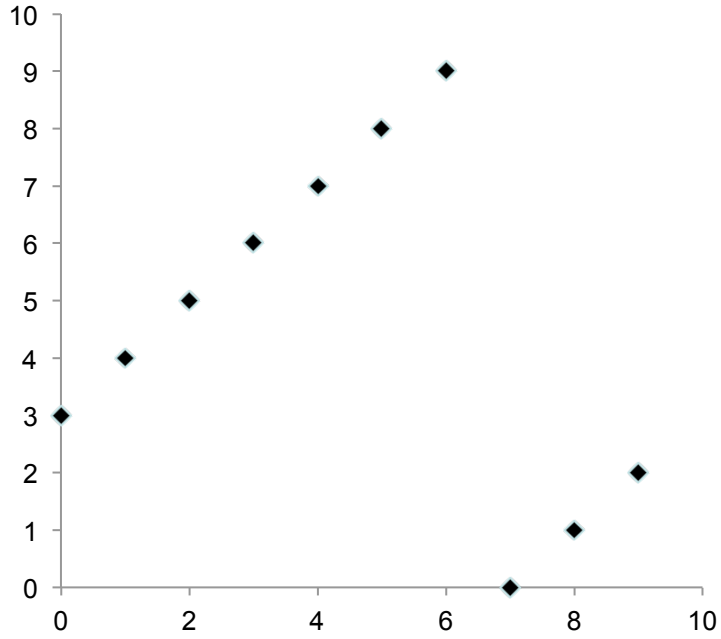
Chiffrement en continu : RC4 (1/4)

- Séquences pseudo-aléatoires générées par machines déterministes (ordinateurs…) forcément *périodiques*
- Contrainte supplémentaire en cryptographie : période *la plus longue* possible (pour se rapprocher du chiffre de Vernam)
- Générateurs usuels (type linéaires à congruence : $X_{n+1}=aX_n+b \bmod M$) inutilisables en cryptographie car de période maximale M (état entièrement décrit par leur valeur)
- Générateurs cryptographiques comme RC4 :
  => Nombre d'états possibles beaucoup plus important que nombre de valeurs
  => Fonction de passage à l'état suivant complexe
  => Etat initial dépendant de la clé

Illustration de la faiblesse des générateurs simples

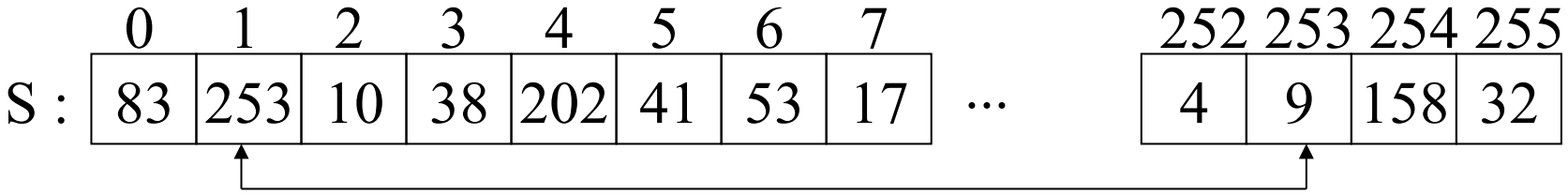
Exemple simplifié : $X_{N+1} = X_N + 3 \bmod 10$

Valeurs de 0 à 9 toutes couvertes car avec $X_0=0$ (par exemple) :
0->3->6->9->2->5->8->1->4->7->0 couverture homogène des 10 valeurs possibles
Mais : utilisation pour tirer au hasard un point du carré 10*10 met en évidence le caractère non aléatoire



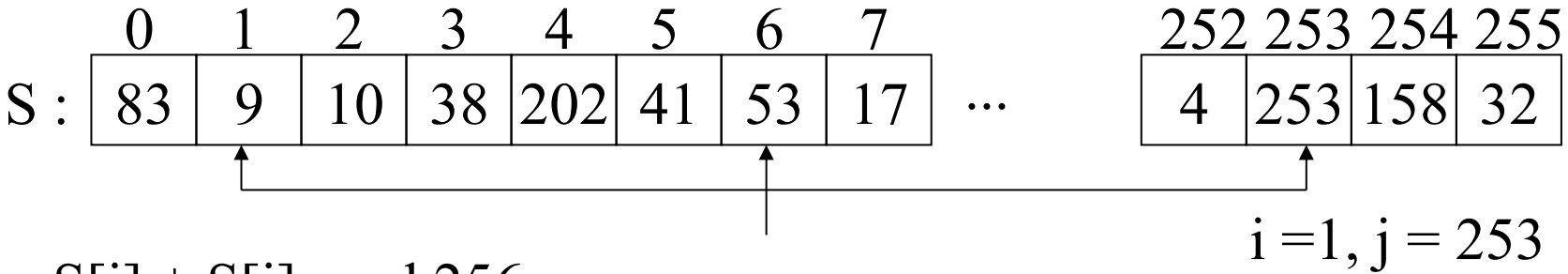
Chiffrement en continu : RC4 (2/4)

- RC4 génère des octets pseudo-aléatoires par permutations sur un tableau S de 256 entiers de 1 octet, initialement rempli par les entiers 0 à 255 dans un ordre aléatoire dépendant de la clé.

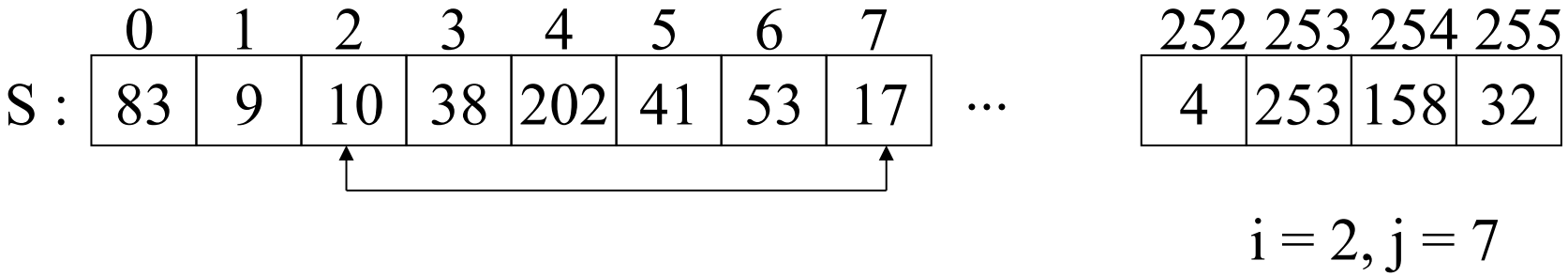


  - Ce tableau est alors permuté par pas élémentaires utilisant deux compteurs i et j (initialisés à 0). Un pas :
  - i = i + 1 mod 256 (assure que chaque case est changée)
  - j = j + S[i] mod 256 (échange avec une case pseudo-aléatoire)
  - Permuter S[i] et S[j]

Chiffrement en continu : RC4 (3/4)



- t = S[i] + S[j] mod 256
- Octet aléatoire n°i : S[t]
- Et ainsi de suite (=> i=2, j=253+10 mod 256=7, t=10+17=27…)



Chiffrement en continu : RC4 (4/4)

Initialisation du tableau (ordre aléatoire dépendant de la clé) :

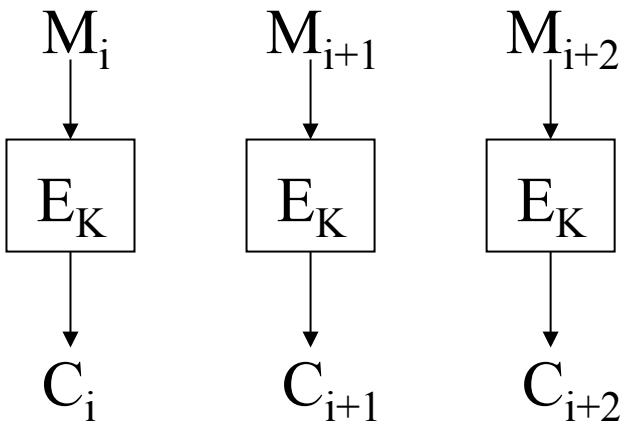
- Le remplir linéairement : S[i] = i
- Fabriquer un autre tableau de 256 octets avec la clé K[i] répétée autant que nécessaire (la clé RC4 a donc une longueur variable pouvant aller jusqu'à 256 octets) :
- Pour i de 0 à 255
- j = j + S[i] + K[i] mod 256 (j est initialisé à 0)
- permuter S[i] et S[j]

RC4 peut être dans $256!*256^2$ états (Nombre de l'ordre de 2^{1700} ou 10^{512} !). RC4 est encore utilisé pour la sécurisation du wifi (WEP, WPA…) mais le WPA2 utilise l'AES.

Chiffrement par blocs (Block Cipher) : modes (1/4)

Rappel : les algorithmes peuvent être symétriques (clé secrète) ou asymétriques (clé publique)

- Transforme un bloc de message clair en un bloc de texte chiffré de même taille (typiquement 64 ou 128 bits)
- Mode d'utilisation le plus simple : ECB : Electronic Code Book : blocs chiffrés indépendamment



- M_i : Bloc i du message clair
- C_i : Bloc i du texte chiffré
- E_K : Algorithme de chiffrement (K est la clé)

Chiffrement par blocs (Block Cipher) : modes (2/4)

Grave inconvénient du mode ECB : un attaquant peut enregistrer de nombreux messages ce qui permet de connaître la structure sans les déchiffrer :

BNP	Dupont	127873	Dépot	500€
BNP	Lepirate	132607	Retrait	5000€

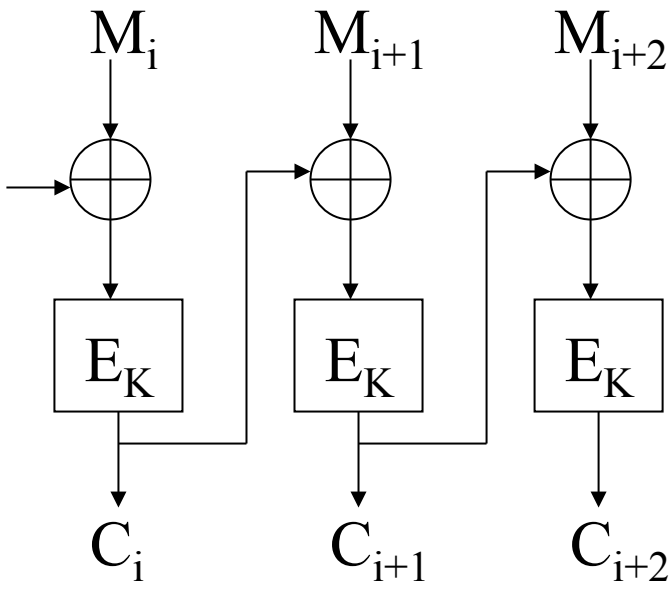
Puis fabriquer un message à sa convenance par assemblage :

BNP	Lepirate	132607	Dépot	5000€

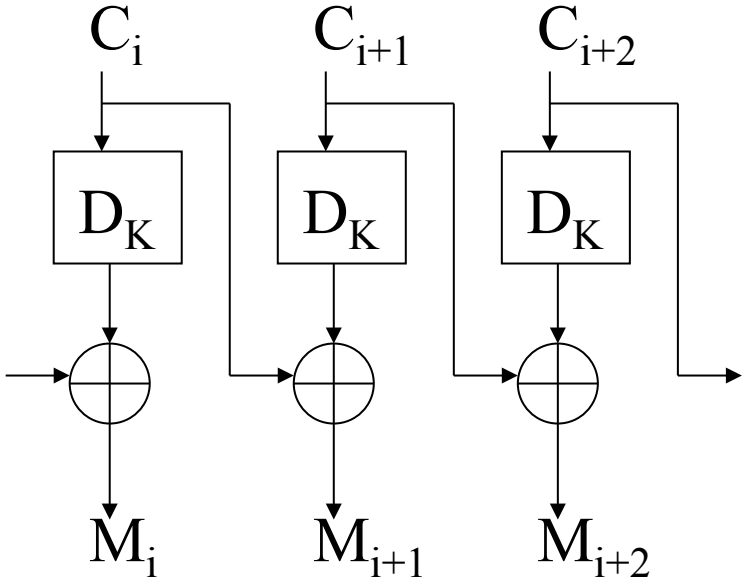
Cette technique dite « Block replay » rend le mode ECB fragile si utilisé seul (=> doit être combiné avec des authentifications)

Chiffrement par blocs (Block Cipher) : modes (3/4)

Pour pallier cet inconvénient différents types de rétroaction peuvent être mis en œuvre : Mode Cipher Block Chaining (CBC). Pour le premier bloc : un vecteur d'initialisation (Initialization Vector : IV) aléatoire est utilisé.



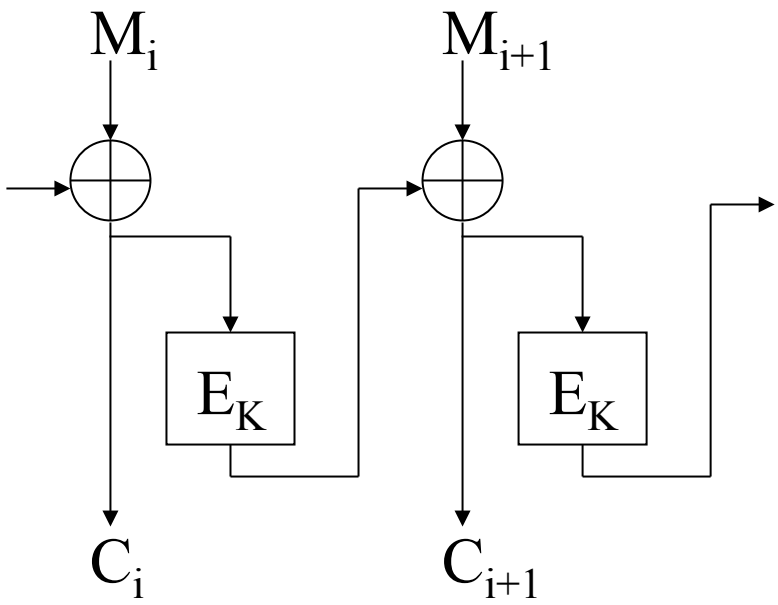
Chiffrement



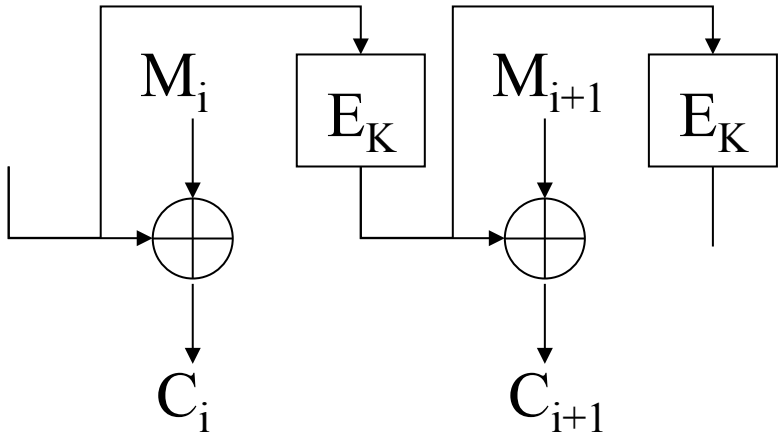
Déchiffrement

Chiffrement par blocs (Block Cipher) : modes (4/4)

Pour pallier cet inconvénient différents types de rétroaction peuvent être mis en œuvre



Mode Cipher Feedback (CFB)



Mode Output Feedback (OFB)

Chiffrement par blocs à clé secrète : DES (1/11)

- DES : Data Encryption Standard : L'ancien standard d'algorithme symétrique.
- Opère sur des blocs de 64 bits en utilisant une clé de 64 bits (en fait 56 seulement sont utilisés comme clé, les autres pouvant être utilisés comme contrôle d'intégrité de la clé)
- Combine *substitutions* et *permutations* (objectif : chaque bit de sortie devient une fonction pseudo-aléatoire de chaque bit d'entrée).
- Désormais abandonné (date de 1976). Fut utilisé (ainsi que MD5 décrit après) pour « hacher » les mots de passe UNIX.

Chiffrement par blocs à clé secrète : DES (2/11)

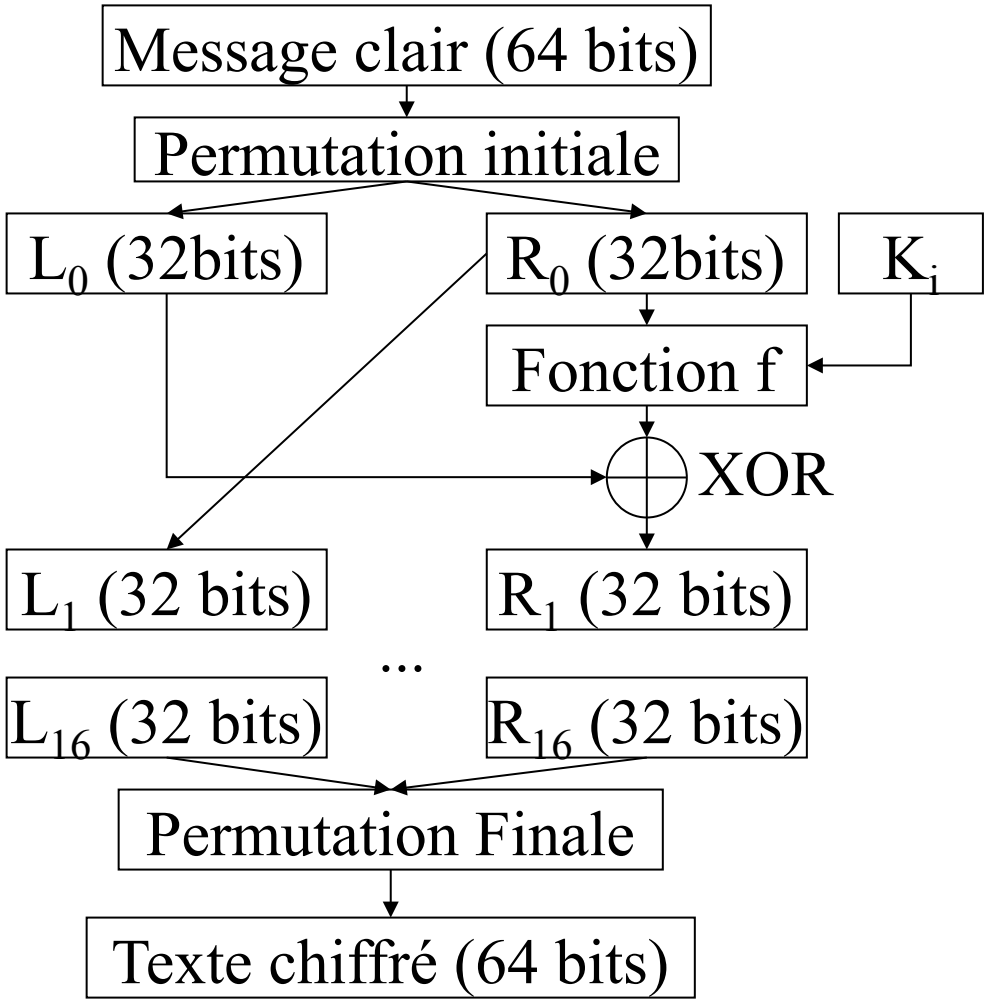
- 16 rondes

$L_i = R_{i-1}$

$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$

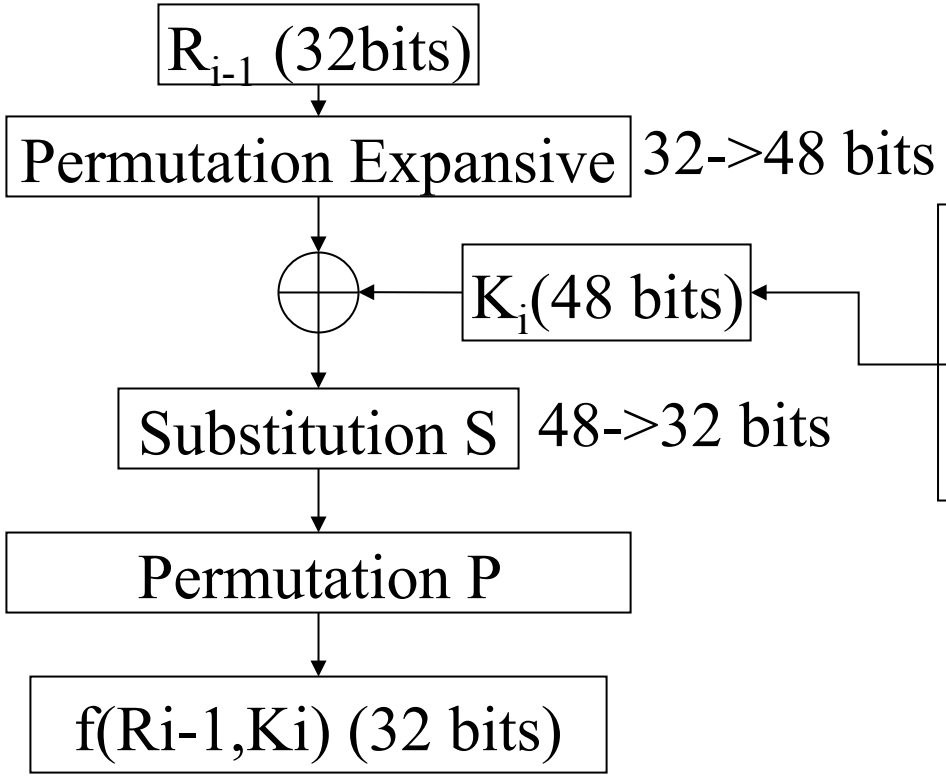
K_i : sous-clé de 48 bits générée pour chaque ronde à partir de la clé de 56 bits

Permutation finale : Echange L16<->R16 puis inverse de la permutation initiale

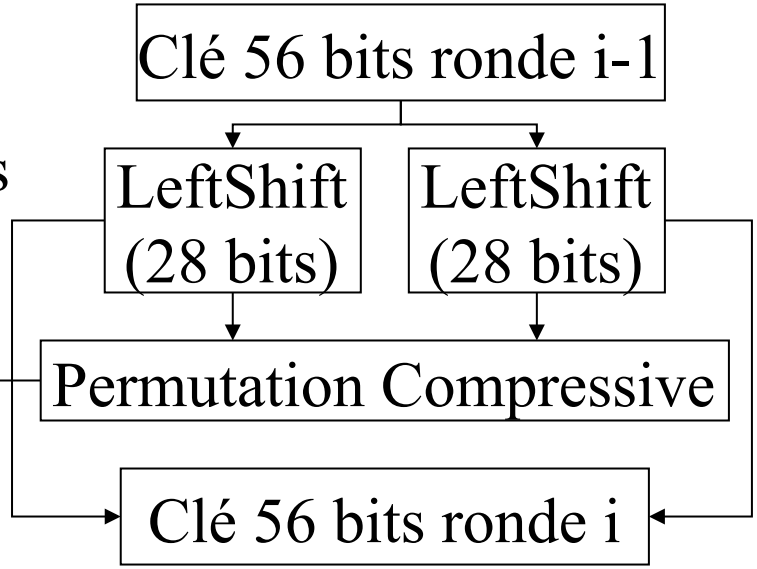


Chiffrement par blocs à clé secrète : DES (3/11)

- Détail de la fonction f :



- Elaboration des clés



Chiffrement par blocs à clé secrète : DES (4/11)

- Substitutions : données par des tables (mettre le bit 58 en premier, puis le 50 etc.) :
- Permutation initiale du message

  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,
- Permutation sur la clé (réduction à 56 bits)

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

Chiffrement par blocs à clé secrète : DES (5/11)

- Nombre de shifts de la clé pour chaque ronde (de 1 à 16)
  1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1
- Permutation compressive (clé 56->48 bits)
  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
- Permutation Expansive (bloc message 32->48 bits)
  32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9,
  8, 9, 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, 1

Chiffrement par blocs à clé secrète : DES (6/11)

- Permutation P

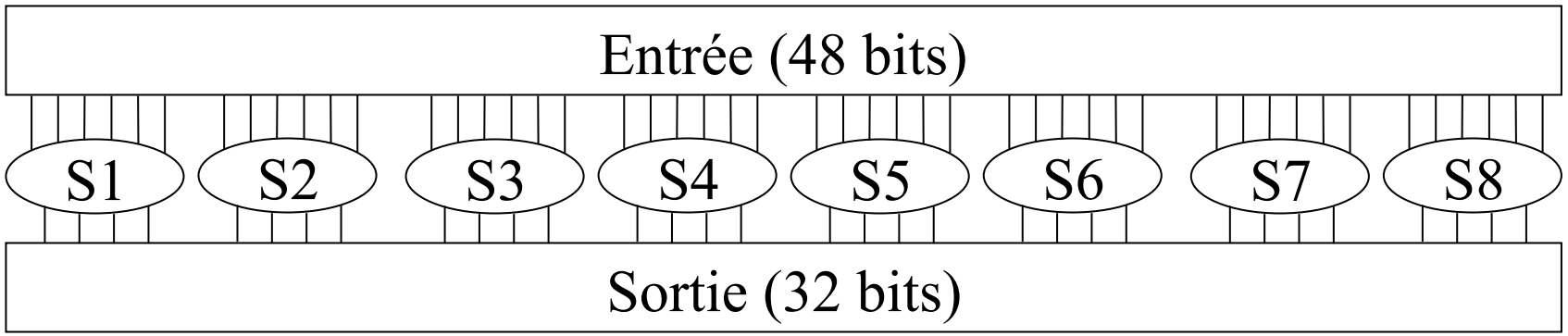
  16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23, 26, 5, 18, 31, 10,
  2, 8, 24, 14, 32, 27, 3, 9, 19, 13, 30, 6, 22, 11, 4, 25

- Permutation finale

  Echanger d'abord L16 et R16 puis effectuer
  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

Chiffrement par blocs à clé secrète : DES (7/11)

- Mais l'essentiel du secret de DES est dans la substitution S (48->32bits) effectuée par 8 « Boites S » prenant chacune 6 bits en entrée et donnant 4 bits en sortie :



Pour chaque entrée de 6 bits on combine les deux bits extrèmes pour former un nombre entre 0 et 3 (-> n° ligne), puis les 4 bits du milieu pour former un nombre entre 0 et 15 (-> n° colonne)

Chiffrement par blocs à clé secrète : DES (8/11)

- Résultat (nombre de 4 bits donc entre 0 et 15) dans la boite S à la (ligne, colonne) correspondante (indices à partir de 0) :
- Boite S1

  14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,
  0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,
  4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,
  15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13
- Boite S2

  15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,
  3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,
  0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,
  13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9

Chiffrement par blocs à clé secrète : DES (9/11)

- Boite S3

  10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,
  13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,
  13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,
  1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12

- Boite S4

  7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,
  13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,
  10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,
  3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14

Chiffrement par blocs à clé secrète : DES (10/11)

- Boite S5

  2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,
  14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,
  4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,
  11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3

- Boite S6

  12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,
  10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,
  9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,
  4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13

Chiffrement par blocs à clé secrète : DES (11/11)

- Boite S7

  4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,
  13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,
  1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,
  6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12

- Boite S8

  13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,
  1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,
  7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,
  2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11

DES : Déchiffrement et cryptanalyse (1/2)

- Algorithme de déchiffrement : **le même que l'algorithme de chiffrement en utilisant les clés dans l'ordre inverse** ! (Si vous en doutiez, DES n'a pas été conçu au hasard…)
- Cryptanalyse : la technique la plus efficace contre DES est la *cryptanalyse différentielle* (étudiant la propagation dans le texte chiffré de la différence entre deux textes clairs choisis)
- Cette technique (pas à la portée du premier pirate venu) est efficace contre les DES simplifiés (moins de 16 rondes)
- La longueur de la clé est désormais trop faible en regard des progrès de la technologie informatique.

DES : Déchiffrement et cryptanalyse (2/2)

- Il existe des variantes de DES (genre 3DES) qui augmentent la sécurité du DES et utilisent des clés plus longues,
- La NSA américaine a participé à la conception de boîtes S et est soupçonnée (à tort ?) d'y avoir introduit une faille secrète (« trap-door » ou porte d'évitement secrète),
- Quoi qu'il en soit cette organisation qui compte parmi les meilleurs cryptographes mondiaux (et qui a longuement travaillé sur DES) peut certainement casser un chiffre DES très rapidement.
- **Moralité** : Le DES ne soit plus désormais être utilisé, son successeur l'AES l'a maintenant totalement remplacé.

Authentification sur les systèmes UNIX

- L'authentification sur les systèmes UNIX basée sur crypt(3) utilisait une variante de DES utilisée comme *fonction à sens unique* (i.e. facile à calculer, très difficile à inverser) :
- /etc/passwd (ou /etc/shadow) ne contient pas le mot de passe chiffré, mais le résultat du chiffrement (multiple) par crypt(3) d'une chaîne de caractères donnée, avec comme clé le mot de passe.
- Authentification :
  Utilisateur : tape son mot de passe
  Système : calcule le résultat de la fonction à sens unique
  Système : compare ce résultat à celui enregistré dans /etc/passwd ou /etc/shadow pour l'utilisateur

Authentification par mots de passe : Attaque par dictionnaires

- Grande vulnérabilité à une *attaque par dictionnaires* (essai systématique avec fichier de mots de passe courants genre *monprénom*, *machérie*, *monherosdeBD*)
- Amélioration : ajout de « sel » (chaîne aléatoire différente pour chaque utilisateur et concaténée au mot de passe avant calcul de la fonction à sens unique)
  Mais : sel forcément stocké dans /etc/passwd ou /etc/shadow (pour que le système puisse recalculer la fonction),
- Limite juste le piratage à attaques sur un utilisateur donné
- **Moralité** : choisissez des mots de passe robustes cette attaque, mais faciles à retenir. (Exemple **TVAPAQSA**)
  Tout **V**ient **A** **P**oint **A** **Q**ui **S**ait **A**ttendre...

Le successeur du DES : AES

- Conscient de la faiblesse potentielle du DES le NIST Américain (National Institute for Standards and Technologies) lance en 1997 un appel d'offres pour élaborer un nouveau standard : l'Advanced Encryption System AES
- Le 2 Octobre 2000, l'algorithme retenu par le NIST est le Rijndael (prononcer «Rain Doll») conçu par deux belges, Joan Daemen et Vincent Rijmen.
- Le 26 Novembre 2001, le nouveau standard est publié en tant que Federal Information Processing Standard (FIPS)
- L'algorithme traite l'entrée par blocs de 128 bits avec une clé de longueur 128, 192 ou 256 bits

AES : Conventions (1/3)

- Le bloc de 128 bits est considéré comme un séquence de 16 octets avec les conventions usuelles (bit de poids fort en premier) puis recopié dans un tableau 4*4 («état») avec les conventions suivantes :

B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15

b7	b6	b5	b4	b3	b2	b1	b0

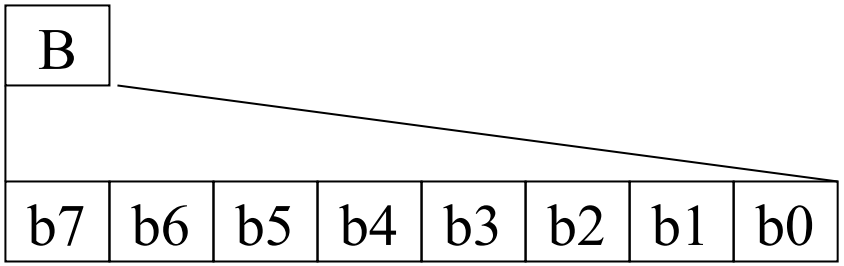
$$B0 = \sum_{i=0}^{7} bi * 2^i$$

Exemple : 10011101 = {9D}

B0	B4	B8	B12
B1	B5	B9	B13
B2	B6	B10	B14
B3	B7	B11	B15

AES : Conventions (2/3)

- AES manipule les octets comme des polynômes de degré <=7 dont les coefficients sont les bits



$$B \Leftrightarrow \sum_{i=0}^{7} bi.x^i$$

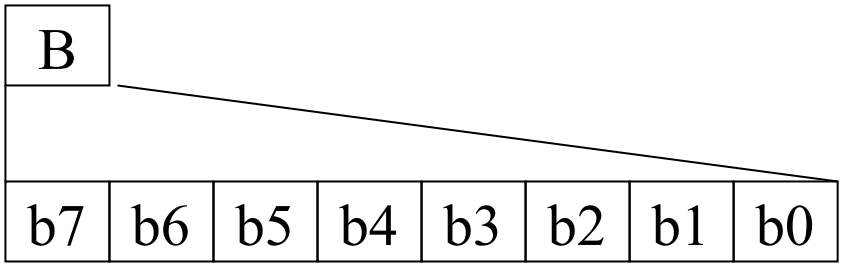
Exemple : {9D} <=> 10011101 <=> $x^7+x^4+x^3+x^2+1$

L'addition de deux octets est alors l'addition modulo 2 des coefficients (équivalent au XOR bit à bit) :

$$\{57\} \oplus \{83\} = ?$$

AES : Conventions (2/3)

- AES manipule les octets comme des polynômes de degré <=7 dont les coefficients sont les bits



$$B \Leftrightarrow \sum_{i=0}^{7} bi.x^i$$

Exemple : {9D} <=> 10011101 <=> $x^7+x^4+x^3+x^2+1$

L'addition de deux octets est alors l'addition modulo 2 des coefficients (équivalent au XOR bit à bit) :

$$\{57\} \oplus \{83\} = 01010111 \oplus 10000011 = 11010100 = \{D4\}$$

AES : Conventions (3/3)

- La multiplication de deux octets est la multiplication polynomiale de leurs polynômes représentatifs, modulo un polynôme *irréductible* de degré 8 :

  $m(x) = x^8 + x^4 + x^3 + x + 1$ ou {01}{1B} en hexa

- Ainsi le résultat reste un polynôme de degré <=7
- Contrairement à l'addition, il n'existe pas d'opération simple au niveau des octets correspondant à cette multiplication

AES : Le corps fini GF(2^8)

- Dûment muni de ces opérations, l'ensemble des octets a une structure de corps fini ou corps de Galois (Galois Field) à 2^8 éléments noté GF(2^8)
- En particulier tout élément non nul a un inverse ce qui est conséquence du fait que le polynôme m est irréductible c'est-à-dire « premier » (on montre ci-après : Algorithme d'Euclide, qu'un élément est inversible modulo un autre si et seulement si ils sont premiers entre eux)
- On démontre que tout corps fini est isomorphe au corps GF(p^m) des polynômes de degré m-1 à coefficients entiers modulo p (premier), la multiplication étant modulo un polynôme irréductible de degré m si m>1

AES : Propriétés des corps finis

- Dans GF(p^m), p est la caractéristique du corps, p^m est sa cardinalité
- GF(p) est le corps des entiers modulo p souvent noté Z/pZ
- Pour tout élément u de GF(p^m), on a : $\underbrace{u + u + \ldots + u}_{p} = 0$
- En particulier dans GF(2^8) on a pour tout élément u : u+u=0 ce qui signifie qu'additionner et soustraire sont équivalents (pas de risque de fautes de signe !)

AES : Exemples de calculs dans GF(2^8) : Produits

Exercice : Effectuer dans GF(2^8) la multiplication :

$$\{19\} \bullet \{3F\}$$

On rappelle que le polynôme modulo est m={01}{1B}

AES : Exemples de calculs dans GF(2^8) : Produits

Soit à effectuer la multiplication notée $\{19\} \bullet \{3F\}$

$$\{19\} \bullet \{3F\} = 00011001 \bullet 00111111 \Leftrightarrow$$

$$\left(x^4 + x^3 + 1\right)\left(x^5 + x^4 + x^3 + x^2 + x + 1\right) = x^9 + x^5 + x^4 + x^2 + x + 1$$

Pour prendre le modulo $m(x) = x^8 + x^4 + x^3 + x + 1$

Il faut soustraire $x.m(x) = x^9 + x^5 + x^4 + x^2 + x$

Conclusion $\{19\} \bullet \{3F\} = \{01\}$

AES : Exemples de calculs dans GF(2^8) : Division Euclidienne

Exercice : Effectuer la division Euclidienne de m={01}{1B} par {19} (trouver le quotient et le reste par division polynomiale)

AES : Exemples de calculs dans GF(2^8) : Division Euclidienne

Dividende
m={01}{1B}

$x^8 + x^4 + x^3 + x + 1$

$x^8 + x^7 + x^4$

$x^7 + x^3 + x + 1$

$x^7 + x^6 + x^3$

$x^6 + x + 1$

$x^6 + x^5 + x^2$

$x^5 + x^2 + x + 1$

$x^5 + x^4 + x$

$x^4 + x^2 + 1$

$x^4 + x^3 + 1$

$x^3 + x^2$ ←

Diviseur
{19}

$x^4 + x^3 + 1$

$x^4 + x^3 + x^2 + x + 1$

Quotient 00011111
soit {1F}

$m = \{1F\} \bullet \{19\} + \{0C\}$

Reste 00001100
soit {0C}

AES : Algorithme d'Euclide étendu dans un corps fini

Pour déterminer le pgcd de deux éléments a et b dans GF(p^m) :

- Tout diviseur de a et b divise a mod b = a - qb, par conséquent : g=pgcd(a,b)=pgcd(b,a mod b) => Algorithme d'Euclide
- Par divisions successives on construit une suite de restes strictement décroissante. Le dernier reste non nul est g=pgcd
- On a donc à l'avant-dernière étape $a_n=q_nb_n+g$ et à l'étape d'avant $a_{n-1}=q_{n-1}b_{n-1}+r_{n-1}$ (avec $a_n=b_{n-1}$ et $b_n=r_{n-1}=a_{n-1}-q_{n-1}b_{n-1}$)
  En reportant dans la première : $b_{n-1}=q_n(a_{n-1}-q_{n-1}b_{n-1})+g$
- Par récurrence, on peut donc à toute étape exprimer g comme combinaison linéaire de a_i et b_i, donc en particulier trouver u et v tels que g=au+bv (identité de Bezout) que l'on détermine donc en « remontant » toutes les divisions euclidiennes

AES : Recherche de l'inverse d'un élément d'un corps fini (1/2)

- Ce qui précède montre que si a et b sont premiers entre eux, alors on peut trouver u et v vérifiant au+bv=1,
  donc a possède un inverse modulo b qui est u mod b (et b possède un inverse modulo a qui est v mod a)
- Réciproquement si on peut inverser a modulo b il existe u tel que au=1 mod b soit au=1+kb.
  Un diviseur de a et b divise par conséquent a-kb=1, donc a et b sont premiers entre eux
- Par conséquent a possède un inverse modulo b (et b possède un inverse modulo a) <=> a et b premiers entre eux
- Inversion d'un élément a de GF(2^8) : par algorithme d'Euclide étendu entre a et le polynôme irréductible m(x)

AES : Recherche de l'inverse d'un élément d'un corps fini (2/2)

- Exercice : Utiliser l'algorithme d'Euclide étendu pour inverser l'élément {19} de GF(2^8)
- On rappelle que le polynôme modulo est m={01}{1B} et que la première division de l'algorithme est par conséquent {01}{1B} / {19} qui vient d'être effectué à titre d'exercice

AES : Recherche de l'inverse d'un élément d'un corps fini (2/2)

- Soit à inverser l'élément {19} de GF(2^8)
- La première étape ci-avant montre que $m = \{1F\} \bullet \{19\} + \{0C\}$
- Deuxième étape : Dividende {19}

$$\begin{array}{l|l} x^4 + x^3 + 1 & x^3 + x^2 \text{ Diviseur} \\ \hline x^4 + x^3 & \{0C\} \\ \hline & x \\ 1 & \boxed{\{19\} = \{02\} \bullet \{0C\} + \{01\}} \end{array}$$

- D'où :

$$\{19\} + \{02\} \bullet (m + \{1F\} \bullet \{19\}) = \{01\}$$

$$\{19\} \bullet (\{01\} + \{02\} \bullet \{1F\}) = \{01\}$$

$$\{02\} \bullet \{1F\} \Leftrightarrow x(x^4 + x^3 + x^2 + x + 1)$$

$$x^5 + x^4 + x^3 + x^2 + x \Leftrightarrow 00111110 = \{3E\}$$

$$\boxed{\{19\}^{-1} = \{01\} + \{3E\} = \{3F\}}$$ Identité vérifiée plus haut

Une astuce pour calculer dans GF(2^8) : logarithmes discrets

- On cherche un générateur du groupe multiplicatif GF(2^8)* : avec le polynôme modulo de Rijndael {03} est un tel générateur : les puissances successives de {03} parcourent les 255 éléments de GF(2^8)* de sorte que $\{03\}^{255} = \{01\}$
- On dresse alors la table des $\{03\}^i$, et son inverse qui représente le logarithme discret à base {03}
- Cela est possible du fait de que GF(2^8) est un corps de taille modeste (le calcul du logarithme dans un corps de grande taille est très difficile : cf. ci-après)

Une astuce pour calculer dans GF(2^8) : logarithmes discrets

- Table des {03}^CL :

	0	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**	**110**	**120**	**130**	**140**	**150**	**160**	**170**	**180**	**190**	**200**	**210**	**220**	**230**	**240**	**250**
0	01	72	D8	66	6A	04	D3	4D	83	B3	10	61	2F	3A	FA	40	9F	BC	E8	C5	1B	4A	C6	8D	39	6C
1	03	96	73	AA	BE	0C	6E	D7	9E	CE	30	A3	71	4E	15	C0	BA	DF	23	54	2D	DE	51	8C	4B	B4
2	05	A1	95	E5	D9	14	B2	62	B9	49	50	FE	93	D2	3F	5B	D5	7A	65	FC	77	79	F3	8F	DD	C7
3	0F	F8	A4	34	70	3C	CD	A6	D0	DB	F0	19	AE	6D	41	ED	64	8E	AF	1F	99	8B	0E	8A	7C	52
4	11	13	F7	5C	90	44	4C	F1	6B	76	0B	2B	E9	B7	C3	2C	AC	89	EA	21	B0	86	12	85	84	F6
5	33	35	02	E4	AB	CC	D4	08	BD	9A	1D	7D	20	C2	5E	74	EF	80	25	63	CB	91	36	94	97	01
6	55	5F	06	37	E6	4F	67	18	DC	B5	27	87	60	5D	E2	9C	2A	9B	6F	A5	46	A8	5A	A7	A2	03
7	FF	E1	0A	59	31	D1	A9	28	7F	C4	69	92	A0	E7	3D	BF	7E	B6	B1	F4	CA	E3	EE	F2	FD	05
8	1A	38	1E	EB	53	68	E0	78	81	57	BB	AD	FB	32	47	DA	82	C1	C8	07	45	3E	29	0D	1C	0F
9	2E	48	22	26	F5	B8	3B	88	98	F9	D6	EC	16	56	C9	75	9D	58	43	09	CF	42	7B	17	24	11

Une astuce pour calculer dans GF(2^8) : logarithmes discrets

- Table des $\log_{\{03\}}$(LC) (valeurs en décimal) :

0	**0**	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**A0**	**B0**	**C0**	**D0**	**E0**	**F0**
0		100	125	101	150	102	126	43	175	44	127	204	151	83	68	103
1	0	4	194	47	143	221	110	121	88	215	12	187	178	57	17	74
2	25	224	29	138	219	253	72	10	168	117	246	62	135	132	146	237
3	1	14	181	5	189	48	195	21	80	122	111	90	144	60	217	222
4	50	52	249	33	54	191	163	155	244	235	23	251	97	65	35	197
5	2	141	185	15	208	6	182	159	234	22	196	96	190	162	32	49
6	26	129	39	225	206	139	30	94	214	11	73	177	220	109	46	254
7	198	239	106	36	148	98	66	202	116	245	236	134	252	71	137	24
8	75	76	77	18	19	179	58	78	79	89	216	59	188	20	180	13
9	199	113	228	240	92	37	107	212	174	203	67	82	149	42	124	99
A	27	8	166	130	210	226	40	172	233	95	31	161	207	158	184	140
B	104	200	114	69	241	152	84	229	213	176	45	108	205	93	38	128
C	51	248	154	53	64	34	250	243	231	156	164	170	55	86	119	192
D	238	105	201	147	70	136	133	115	230	169	118	85	63	242	153	247
E	223	28	9	218	131	145	61	167	173	81	123	41	91	211	227	112
F	3	193	120	142	56	16	186	87	232	160	183	157	209	171	165	7

Une astuce pour calculer dans GF(2^8) : logarithmes discrets

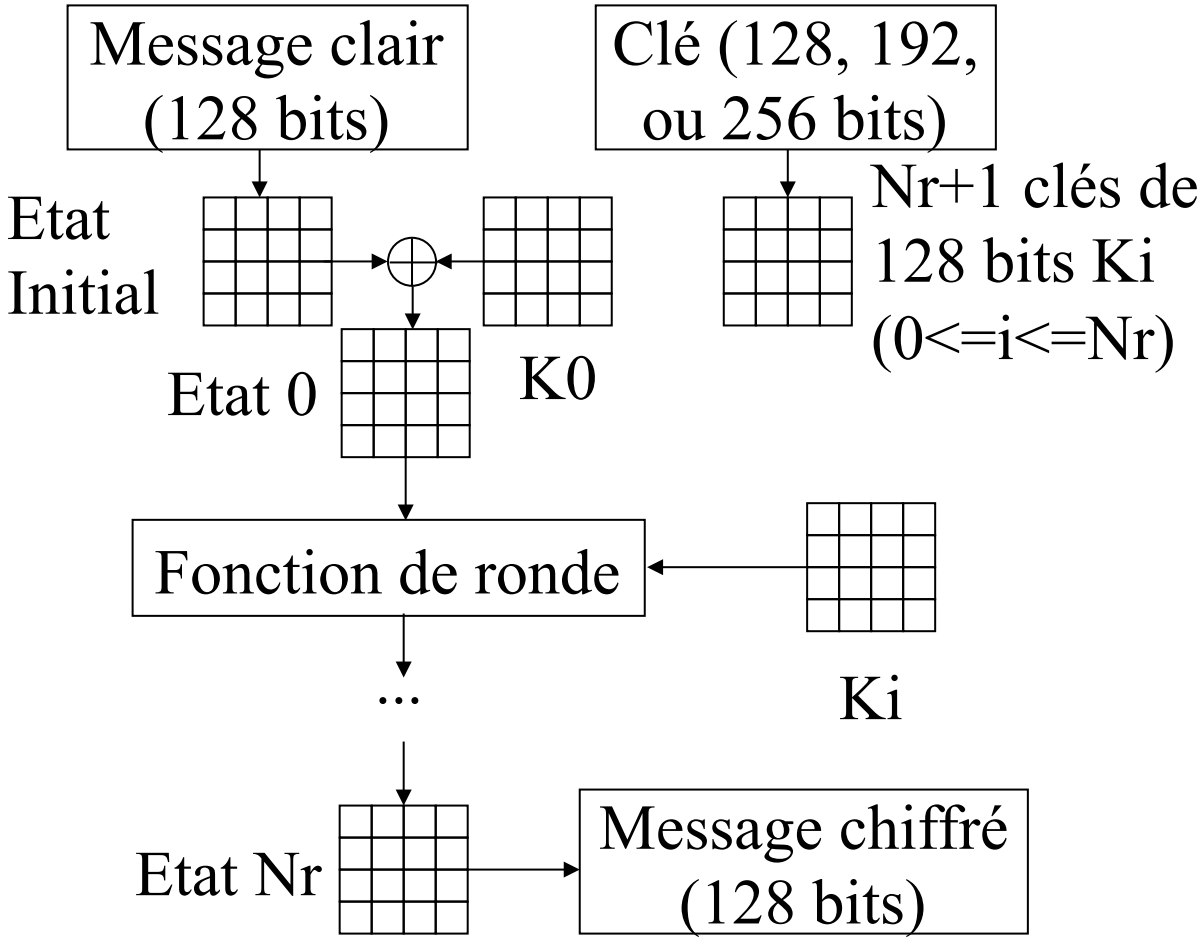
- Soit à effectuer $\{19\} \bullet \{3F\}$
- $\log_{\{03\}}(\{19\}) = 113$, $\log_{\{03\}}(\{3F\}) = 142$

  $\{19\} \bullet \{3F\} = \{03\}^{113} \bullet \{03\}^{142} = \{03\}^{255} = \{01\}$
- Soit à rechercher l'inverse de $\{19\}$
- Le $\log_{\{03\}}$ de l'inverse doit valoir 255-113=142
- On recherche $\{03\}^{142}$ dans la table (142 = 0x8E)
- L'inverse vaut $\{3F\}$

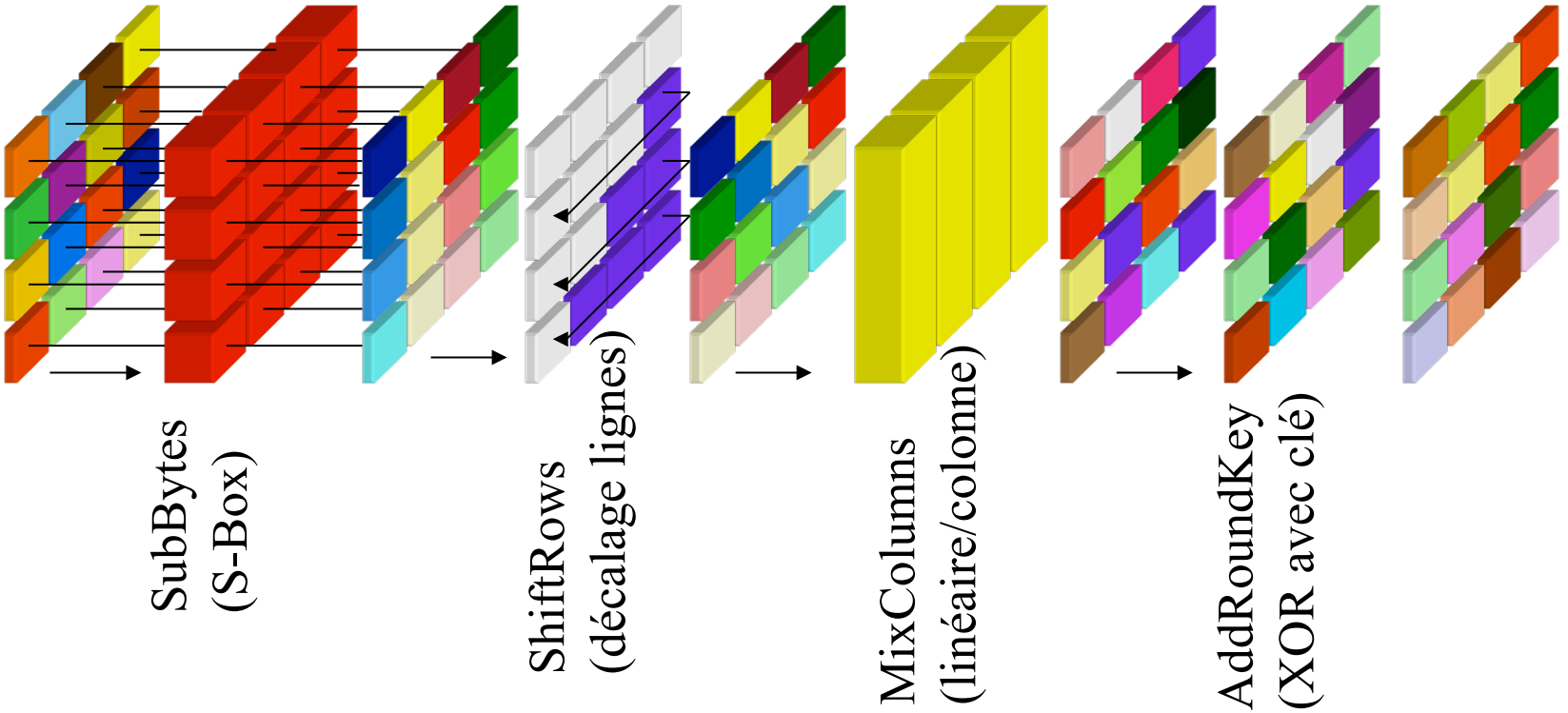
AES : Présentation de l'algorithme

- Nr Rondes

Clé	Nr
128 bits	10
192 bits	12
256 bits	14



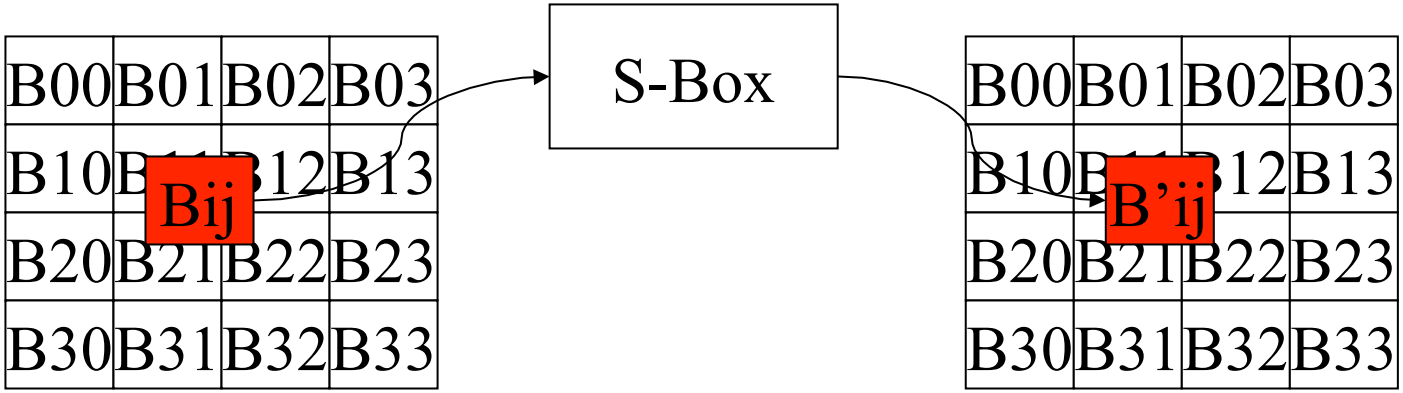
AES : Fonction de ronde



- Exception : la dernière ronde ne comporte pas de MixColumns

AES : SubBytes

- La transformation Subbytes opère sur la matrice d'états octet par octet en appliquant une S-Box :



- Prendre l'inverse multiplicatif (dans GF(2^8)) puis remplacer le bit i de l'octet par (c_i : bit i de {63}) :

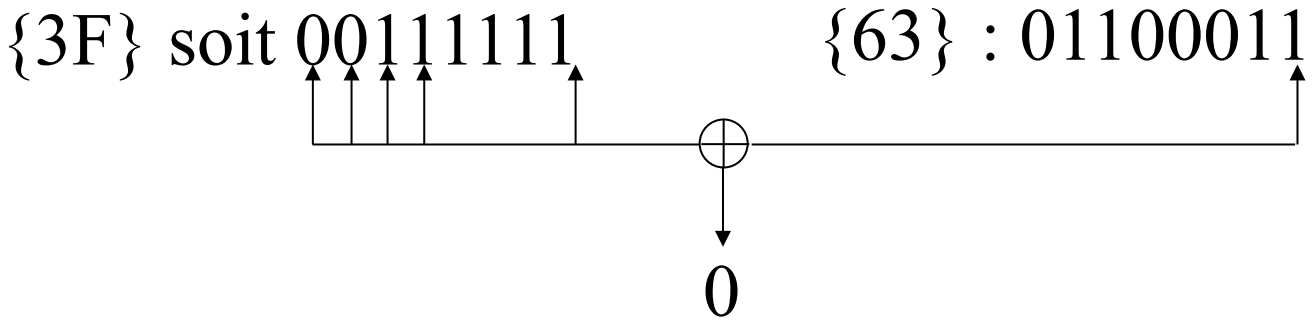
$$b_i' = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i$$

- Nota : pour l'octet 00 qui n'a pas d'inverse on substitue 00 et on applique la seconde étape : le substitué de 00 vaut donc 63

AES : SubBytes : Exemple

- Calcul du bit 0 de l'octet substitué à {19} :

Inverse de {19} :



- Idem pour chaque bit : résultat {D4}
- Le résultat de la S-Box sur chaque octet de {00} à {FF} peut être précalculé et placé dans une table.

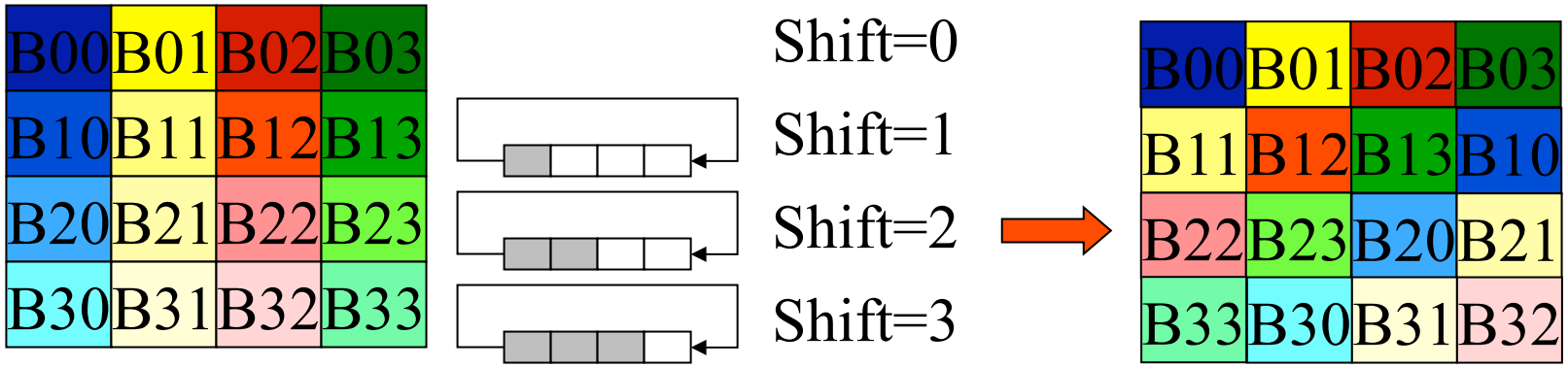
AES : SubBytes : La S-Box

- Transformée de l'octet {LC} :

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

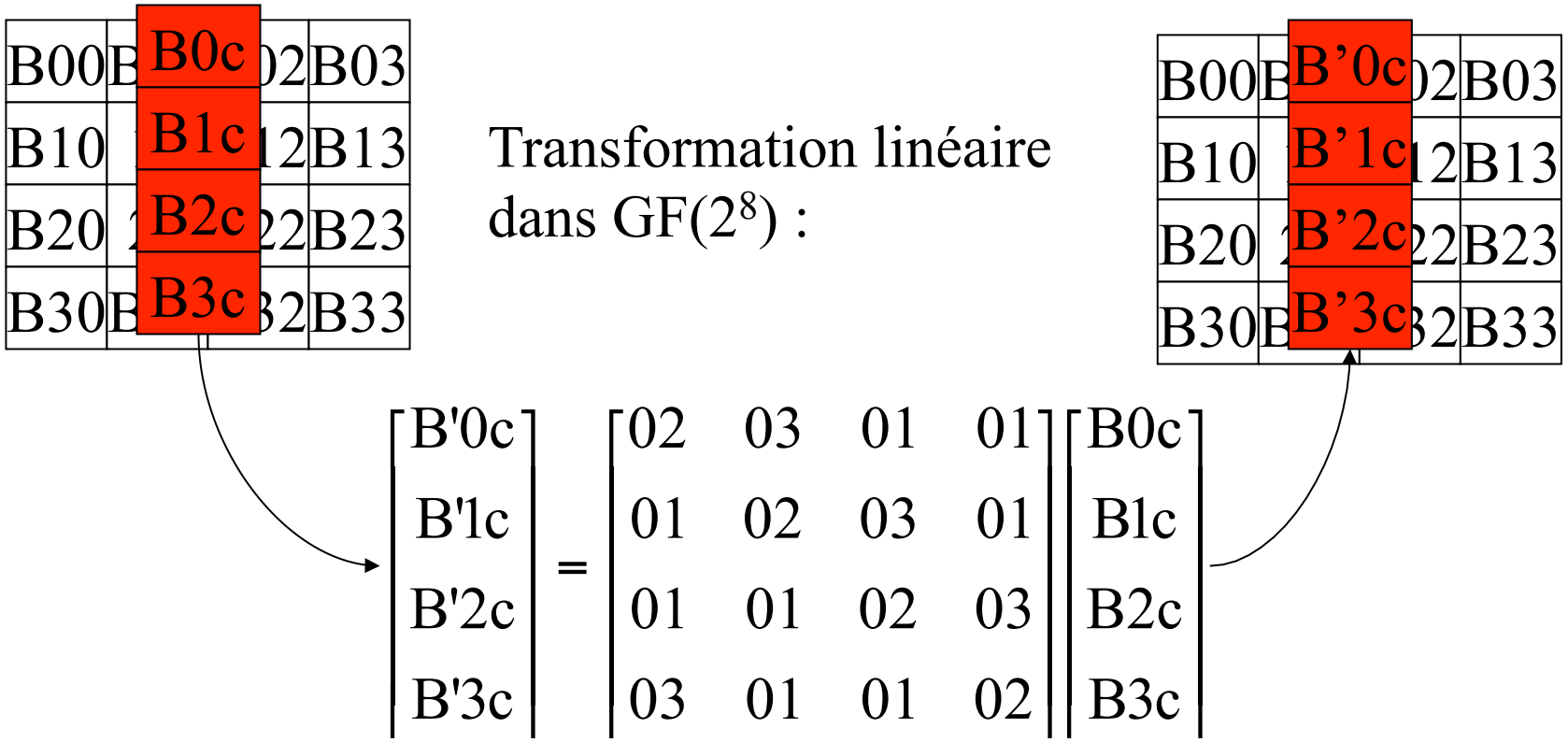
AES : ShiftRows

- La transformation ShiftRows opère sur la matrice d'états ligne à ligne en effectuant des shifts cycliques à gauche :



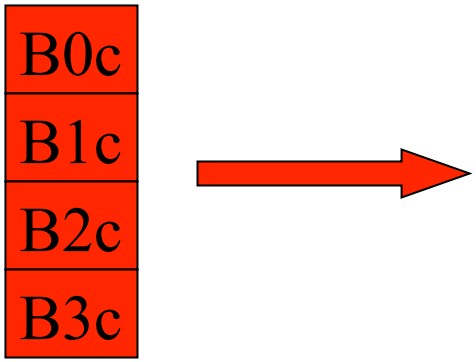
AES : MixColumns (1/3)

- MixColumns opère colonne par colonne :



AES : MixColumns (2/3)

- MixColumns peut également être décrite en considérant la colonne comme polynôme de degré 3 à coefficients dans GF(2^8) :



$$B3c.x^3 + B2c.x^2 + B1c.x + B0c$$

- MixColumns est alors équivalente à la multiplication modulo x^4+1 de ce polynôme par le polynôme :
  $a(x) = \{03\}.x^3 + \{01\}.x^2 + \{01\}.x + \{02\}$

AES : MixColumns (3/3)

- En effet, le terme de degré 0 du résultat est la somme des termes de degrés 0 et 4 du produit :

$$(B3c.x^3+B2c.x^2+B1c.x+B0c)\ (\{03\}.x^3+\{01\}.x^2+\{01\}.x+\{02\})$$

car l'élimination d'un terme $A.x^4$ par le modulo s'effectue en ajoutant $A.(x^4+1)$

D'où $B'0c = \{02\} \bullet B0c \oplus \{03\} \bullet B1c \oplus \{01\} \bullet B2c \oplus \{01\} \bullet B3c$

Idem pour les autres degrés. Ainsi l'inverse de MixColumns sera la multiplication par l'inverse modulo x^4+1 de a(x), s'il existe (x^4+1 non premier)

AES : MixColumns : Exemple

- La colonne à mixer est représentée par le polynôme $(\{30\}.x^3+ \{5D\}.x^2+ \{BF\}.x+ \{D4\})$
- $\log_{\{03\}}(\{D4\})=65$, $\log_{\{03\}}(\{02\})=25$ d'où :

$$\{02\} \bullet \{D4\} = \{03\}^{90} = \{03\}^{0x5A} = \{B3\}$$

- $\log_{\{03\}}(\{BF\})=157$, $\log_{\{03\}}(\{03\})=1$ d'où :

$$\{03\} \bullet \{BF\} = \{03\}^{158} = \{03\}^{0x9E} = \{DA\}$$

- Soit finalement :

$$B'0c = \{B3\} \oplus \{DA\} \oplus \{5D\} \oplus \{30\} = \{04\}$$

- Idem pour les autres éléments de la colonne

AES : AddRoundKey

- AddRoundKey est le simple XOR bit à bit de la matrice d'état avec une matrice de 4*4 Octets fabriquée à partir de la clé de chiffrement pour la ronde en cours.
- Si le nombre de rondes est Nr il faut donc Nr+1 telles matrices, l'algorithme commençant par une addition de clé initiale (Rappel : Nr dépend de la taille de la clé : 128, 192 ou 256)
- La fabrication de ces Nr+1 clés est l'algorithme d'expansion des clés décrit ci-après

AES : Expansion des clés (1/3)

- Manipule des mots de 32 bits.
- Clé de l'addition initiale : 4 (premiers) mots de la clé de chiffrement notés w[0] à w[3] rangés comme le message (en colonne du haut vers le bas).
- Selon la clé (128,192 ou 256), Nr = 10, 12 ou 14 => on doit obtenir au total 44, 52 ou 60 mots de 32 bits
- Clé de la ronde i : mots numérotés de w[4*i] à w[4*i+3] rangés dans une matrice de 4*4 octets suivant les même conventions

AES : Expansion des clés (2/3)

- La clé de chiffrement est constituée des Nk premiers mots (Nk=4, 6 ou 8 pour AES 128, 192 ou 256)
- En général pour i >= Nk $w[i] = w[i-1] \oplus w[i-Nk]$
- sauf si i est multiple de Nk où w[i-1] subit préalablement les opérations RotWord, SubWord et XOR avec une constante notée Rcon[i/Nk]
- Autre exception pour l'AES 256 seulement (Nk=8) : si i-4 est multiple de 8, w[i-1] subit un Subword (les opérations précédentes restant applicables si i est multiple de 8)

AES : Expansion des clés (3/3)

- Rotword : Shift cyclique gauche : Transforme un mot de 32 bits [B0,B1,B2,B3] en [B1,B2,B3,B0]
- Subword : Opère octet par octet en appliquant la même S-Box que SubBytes
- Rcon[i] est le mot de 32 bits représenté par $[x^{i-1},\{00\},\{00\},\{00\}]$
- Par exemple, pour Rcon[7] utilisé pour w[7Nk] l'octet représenté par x^6 est 01000000 soit {40} et Rcon[7]=40000000
- Si i-1>7, il faut faire un recalage modulo m(x)

AES : Expansion des clés : Exemple (1/2)

- En AES 128 si la clé vaut
  2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C

La clé utilisée pour l'addition initiale est :

2B	28	AB	09
7E	AE	F7	CF
15	D2	15	4F
16	A6	88	3C

- Les mots w[0] à w[3] sont ceux que l'on lit en colonnes de haut en bas et de gauche à droite

AES : Expansion des clés : Exemple (2/2)

- Pour fabriquer w[4] qui sera la première colonne de la clé de la première ronde, il faut appliquer RotWord, SubWord et XOR avec Rcon[1] sur 09CF4F3C puis faire un XOR du résultat avec 2B7E1516
- Après RotWord on a CF4F3C09
- Après SubWord on a 8A84EB01 (voir la S-Box)
- Rcon[1] vaut 01000000 donc après XOR avec Rcon[1] on a 8B84EB01
- Finalement

$$w[4] = 8B84EB01 \oplus 2B7E1516 = A0FAFE17$$

AES : Inversion

- Les opérations SubBytes, ShiftRows, MixColumns et AddRoundKey sont inversibles => il suffit de les appliquer dans l'ordre inverse
- SubBytes : il faut construire la S-Box inverse
- ShiftRows : l'inverse est obtenue par des shifts cyclique à droite
- AddRoundKey : est sa propre inverse
- Pour prouver l'inversibilité de MixColumns on montre que l'inverse modulo x^4+1 de $\{03\}x^3+\{01\}x^2+\{01\}x+\{02\}$ existe et vaut $\{0B\}x^3+\{0D\}x^2+\{09\}x+\{0E\}$

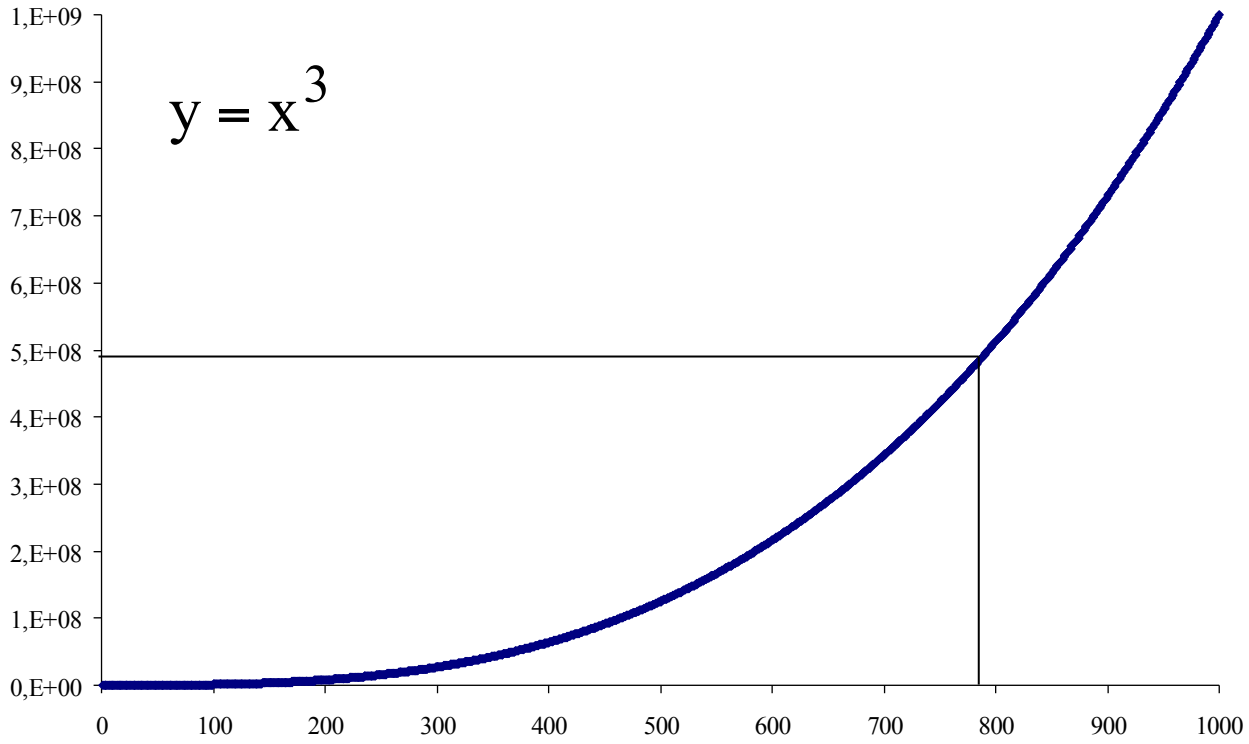
Chiffrement par blocs à clé publique

- Tous les algorithmes à clé publique (algorithmes asymétriques) reposent sur la difficulté *supposée* de certains problèmes mathématiques pour lesquels l'opération inverse est simple :

Il est facile de	Il est difficile de
•Calculer le produit pq (2 grands entiers premiers)	•Trouver les facteurs pq, en ne connaissant que n=pq (problème de la factorisation)
•Calculer a^x mod n (a, x et n grands entiers)	•Trouver x / a^x mod n = b en ne connaissant que b, a et n (problème du logarithme discret)

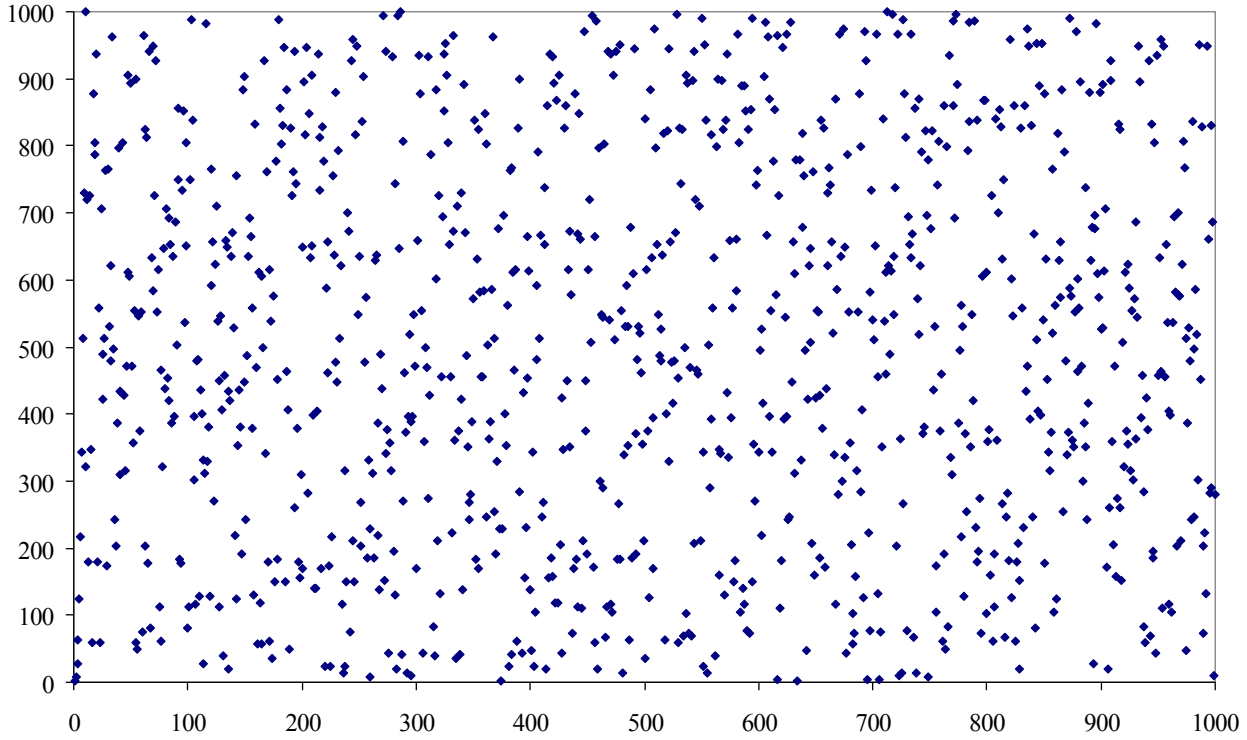
Un exemple de problème facile dans Z...

- Recherche de la racine cubique d'un entier



Et son équivalent dans Z/pZ...

- Très compliqué si p est grand (ici p=1009)



Fonctions à sens unique à porte d'évitement secrète

- De telles fonctions sont supposées être des fonctions à sens unique à porte d'évitement secrète (trapdoor one-way functions) :
- La fonction est facile à calculer (calcul d'un produit),
- L'inversion de la fonction (factorisation) est un problème difficile qui prendrait un temps prohibitif, même compte tenu des progrès supposés des algorithmes et de la puissance informatique (aspect sens unique),
- **Mais** : si l'on connaît un secret (l'un des deux facteurs par exemple) l'inversion redevient facile (aspect porte d'évitement secrète)

Cryptosystèmes basés sur les courbes elliptiques

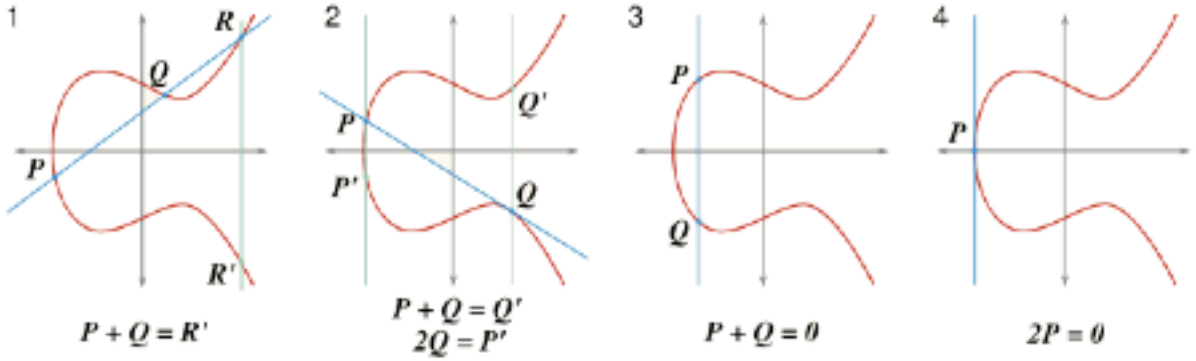
- Sont probablement l'avenir des cryptosystèmes à clé publiques
- Sont basés sur les courbes elliptiques (cubiques non singulières) sur les corps finis dans lesquels est définie une loi de groupe (« addition » de points)
- Les éléments du groupe sont les points du corps fini sous-jacent vérifiant une équation du troisième degré dite Equation de Weierstrass :

$$y^2+a_1xy+a_3y=x^3+a_2x^2+a_4x+a_6$$

- Auxquels est ajouté un point à l'infini qui est l'élément neutre de la loi de groupe
- Les courbes elliptiques ont été utilisés par Andrew Wiles dans la démonstration du théorème de Fermat en 1994

Cryptosystèmes basés sur les courbes elliptiques

- La définition de la loi de groupe « addition » des points peut être vue graphiquement de la manière suivante :



- Il est alors facile de « multiplier » un point par un entier n (l'additionner n fois avec lui même)
- Il est très difficile d'effectuer l'opération inverse (problème analogue au logarithme discret).

Le premier algorithme a clé publique : Diffie-Hellmann (1/2)

- Diffie-Hellmann permet d'échanger une clé secrète sur un canal non sûr. Il fut à l'origine des concepts de la cryptographie à clé publique. Il est basé sur la difficulté supposée du calcul de logarithmes discrets sur un corps fini :
- Les deux protagonistes choisissent deux entiers p et g : p doit être grand et premier, et g un générateur du groupe multiplicatif $(Z/pZ)^*$: la suite des g^k $1<=k<=n-1$ doit parcourir tous les éléments de $(Z/pZ)^*$
- Le premier choisit un grand nombre entier x et calcule $X=g^x \bmod p$ qu'il transmet au second
- Le second choisit un grand nombre entier y et calcule $Y=g^y \bmod p$ qu'il transmet au premier

Le premier algorithme a clé publique : Diffie-Hellmann (2/2)

- Celui qui a choisi x et reçu Y calcule $Y^x \bmod p$
- Celui qui a choisi y et reçu X calcule $X^y \bmod p$
- Les deux résultats sont égaux à $g^{xy} \bmod p$ qui devient la clé secrète partagée par les deux
- Un indiscret éventuel ne connaît que g, p, X et Y
- Ignorant x et y, il n'a aucun moyen simple de calculer $g^{xy} \bmod p$, à moins de les retrouver, ce qui revient à calculer le logarithme discret à base g dans GF(p)

Chiffrement par blocs à clé publique : RSA

- Pour l'algorithme RSA (Rivest, Shamir, Adleman), le problème posé par la cryptanalyse est *supposé* de même difficulté que la factorisation
- Dans l'état de l'art de actuel, on parvient à factoriser des nombres de 200 digits composés d'exactement deux grands facteurs premiers (nombres RSA) en quelques mois avec des équipes internationales qui se partagent le travail pour assurer un effort global de plusieurs années de CPU d'un processeur des technologies actuelles.

Le record actuel (12/12/2009)

- RSA768 (768 bits 232 digits) =
  123018668453011775513049495838496272077285356959533479219732245215172640050726365751874520219978646938995647494277406384592519255732630345373154826850791702612214291346167042921431160222124047927473779408066535141959745985690214341 3.

- Facteurs =
  - 33478071698956898786044169848212690817704794983713768568912431388982883793878002287614711652531743087737814467999489
  - 36746043666799590428244633799627952632279158164343087642676032283815739666511279233373417143396810270092798736308917

A nécessité environ 2000 ans de calculs avec un AMD Opteron mono-cœur à 2,2 GHz

Le challenge actuel

- RSA-240
- Nombre de digits : 240 (795 bits)
- Digits :
  124620366781718784065835044608106590434820374651678805754818788883289666801188210855036039570272508747509864768438458621054865537970253930571891217684318286362846948405301614416430468066875699415246993185704183030512549594371372159029236099
- Nota : des nombres RSA plus petits que RSA-768 résistent encore toutefois…
- Nota : des nombres plus grands mais qui ne sont pas des nombres RSA ont été factorisés depuis (actuellement 2^{1061}-1, nombre de 320 digits factorisé en Août 2012 à l'Université de Californie Fullerton)

RSA : Rappels mathématiques

- Un entier a est inversible modulo un autre entier b <=> a et b premiers entre eux.
- Inverse efficacement déterminé par l'algorithme d'Euclide étendu (qui donne pgcd(a,b) et deux entiers u et v tels que au +bv=pgcd). L'inverse de a modulo b et donc u mod b
- (Petit) théorème de Fermat : Si p est premier, alors pour tout entier a non multiple de p : $a^{p-1} \bmod p = 1$
- Algorithme d'exponentiation rapide : Calcul de $a^p \bmod m$ pour des grands entiers : calculer de proche en proche la suite des $a^{2^i} \bmod m$, représenter p en binaire, p : $c_n c_{n-1} \ldots c_0$, calculer enfin de proche en proche : $\prod_{i=0}^{n} (a^{2^i} \bmod m)^{c_i}$

RSA : Rappels mathématiques
Petit théorème de Fermat

- Soit p premier et a non multiple de p,
- Les p-1 premiers multiples de a : a, 2a, …, (p-1)a sont tous distincts et non nuls modulo p.
- En effet si na=ma mod p alors n=m mod p (car p ne divise pas a). D'autre part si p divisait ka (1<k<p-1) il diviserait a
- Ces p-1 entiers valent donc 1, 2, …, p-1 dans un certain ordre.
- Par multiplication, on obtient $a^{p-1}(p-1)! = (p-1)! \bmod p$
- Par simplification par (p-1)! on obtient

$$a^{p-1} \bmod p = 1$$

Algorithme RSA : génération des clés

- Le destinataire choisit deux grands entiers premiers p et q de même taille, calcule leur produit n=pq ainsi que (p-1)(q-1)
- Il choisit alors au hasard un entier e premier avec (p-1)(q-1)
- Il utilise l'algorithme d'Euclide étendu pour calculer l'inverse de e modulo (p-1)(q-1) : ed mod (p-1)(q-1) = 1
- Il place n et e dans le domaine public (constituent sa clé publique), conserve secrètement d (sa clé privée) et éventuellement détruit toute trace de p et q

Algorithme RSA : chiffrement / déchiffrement

Chiffrement :

- L'émetteur récupère la clé publique (n,e) du destinataire
- Le message clair (chaîne de bits) est tronçonné en paquets représentables par des nombres compris entre 0 et n-1
- Pour chiffrer un paquet m du message, il calcule $c=m^e \bmod n$ qui est le paquet chiffré

Déchiffrement :

- Le destinataire récupère c et pour déchiffrer calcule simplement $c^d \bmod n$ qui vaut m (preuve ci-après)

Note : les calculs de $m^e \bmod n$ et $c^d \bmod n$ nécessitent l'utilisation de l'algorithme d'exponentiation rapide

Algorithme RSA : Preuve que le déchiffrement fonctionne

- Comme $c=m^e \bmod n$, $c^d \bmod n = m^{ed} \bmod n$
- Comme $ed = 1 \bmod (p-1)(q-1)$ il existe un entier k tel que $ed = 1 + k(p-1)(q-1)$
- Par conséquent $c^d \bmod n = m^{1+k(p-1)(q-1)} \bmod n$
- Or (théorème de Fermat) $m^{(p-1)} \bmod p = 1$ si m n'est pas multiple de p. Par élévation à la puissance k(q-1) puis multiplication par m on obtient : $m^{1+k(p-1)(q-1)} \bmod p = m$ égalité qui reste vraie (2 membres=0) si m est multiple de p
- Par symétrie $m^{1+k(p-1)(q-1)} \bmod q = m$ donc $m^{1+k(p-1)(q-1)} - m$ est divisible par p et q donc par pq (p et q premiers et différents)
- Par conséquent $c^d \bmod n = m^{1+k(p-1)(q-1)} \bmod pq = m$

RSA : Exemple très simplifié (calculs réalisables à la main)

- Prenons p=5 et q=11 donc n=pq=55 et (p-1)(q-1)=40
- Prenons e=7, on s'assure (Euclide) que e est premier avec 40

40=5*7+5	On détermine	5-2*2=1
7=1*5+2	alors l'inverse	5-2*(7-1*5)=3*5-2*7=1
5=2*2+**1**	mod (p-1)(q-1)	3*(40-5*7)-2*7=3*40-17*7=1
	(Euclide étendu)	**7^{-1} mod 40 = -17 mod 40 = 23**

- La clé publique vaut (e=7, n=55), la clé privée vaut d=23, les nombres p=5 et q=11 sont détruits
- Soit à coder un fragment de message représenté par la valeur m=2, le calcul de c est simplement
  $c=2^7 \bmod \mathbf{55} = 128 \bmod 55 = 18$

RSA : Exemple très simplifié (calculs réalisables à la main)

- Déchiffrement : le destinataire calcule de c^d mod n $=18^{23}$ mod 55 (on utilise ici l'exponentiation rapide qui permet de ne manipuler que des nombres relativement petits alors que 18^{23} est de l'ordre de 10^{29})

$23_2=10111$

18^1 mod 55 = 18	1	18
18^2 mod 55 = 324 mod 55 = 49	1	18*49 mod 55=2
18^4 mod 55 = 49^2 mod 55 = 2401 mod 55 = 36	1	2*36 mod 55=17
18^8 mod 55 = 36^2 mod 55 = 1296 mod 55 = 31	0	17
18^{16} mod 55 = 31^2 mod 55 = 961 mod 55 = 26	1	17*26 mod 55=2

18^{23} mod 55 = 2

Avantages et inconvénients des algorithmes à clé publique (1/2)

- Résolvent le problème de la transmission des clés (clé de chiffrement publique => on peut envoyer un message chiffré à quelqu'un que l'on n'a jamais rencontré)
- Sont beaucoup plus lents que les algorithmes symétriques
- Sont vulnérables à une attaque « à texte clair connu » (la clé de chiffrement étant publique un pirate peut essayer de nombreux textes clairs et voir si le texte chiffré correspond)
- Sont vulnérables à une attaque « de l'intermédiaire » (un pirate peut substituer une clé publique par la sienne, intercepter et déchiffrer les messages, puis les rechiffrer avec la correcte clé publique et les renvoyer). *Attaque indécelable simplement* => nécessité de la *certification* des clés publiques

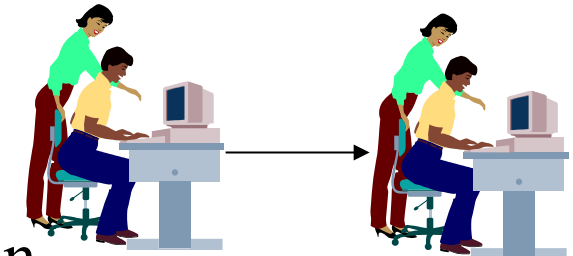
Avantages et inconvénients des algorithmes à clé publique (2/2)

- Il n'est pas prouvé que les problèmes mathématiques sous-jacents ne puissent être résolus par de meilleurs algorithmes
- Il n'est pas prouvé que la cryptanalyse est de même difficulté que les problèmes mathématiques sous-jacents (pour RSA il est juste prouvé que trouver d ou (p-1)(q-1) est de même difficulté que la factorisation)
- Les progrès des algorithmes et de la technologie informatique imposent des clés de taille élevée (1024 bits est conseillé)
- Dans la pratique, les algorithmes à clé publique sont surtout utilisés pour chiffrer et transmettre une *clé de session* (le message est ensuite chiffré en utilisant un algorithme symétrique et cette clé de session) : cryptosystèmes *hybrides*.

Signature numérique

- Problématique : obtenir pour des documents électroniques les propriétés souhaitées d'une signature (authentique, non falsifiable, non réutilisable, inaltérable, non répudiable).
  Protocole très simple avec RSA :

Emetteur :
chiffre M avec
sa clé *privée*
$M^{d_E} \bmod n_E$



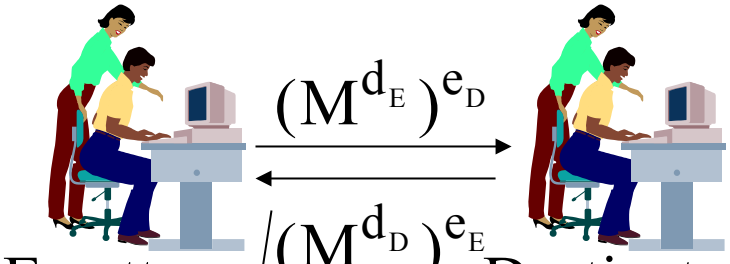
Destinataire : déchiffre
avec clé *publique* de
l'émetteur
$(M^{d_E})^{e_E} \bmod n_E = M$

- Tout le monde peut ici déchiffrer le document ce qui revient à vérifier la signature

- Signatures volumineuses et attaques possibles si même algorithme et mêmes clés sont utilisées pour chiffrer et signer.

Chiffrement + Signature à clé publique (1/2)

- M est signé (clé privée émetteur) avant d'être chiffré (clé publique destinataire). Opérations inverses en ordre inverse par le destinataire (mod n omis pour alléger les notations).



$$\left[(M^{d_E})^{e_D}\right]^{d_D} = (M^{d_E})$$

(Déchiffrement)

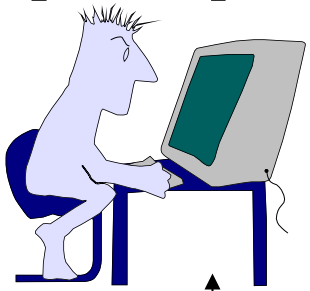
$(M^{d_E})^{e_E} = M$ (Vérif. Signature)

En retour, le destinataire peut envoyer un accusé de réception « j'ai bien reçu le message suivant » (M signé avec sa clé privée puis chiffré avec la clé publique de l'émetteur)

Mais un pirate a enregistré les messages réseau...

Chiffrement + Signature à clé publique (2/2)

- Un peu plus tard le pirate, utilisateur légitime du système renvoie au destinataire le message enregistré. Le destinataire pense qu'il s'agit d'un message normal d'un collègue.



$(M^{d_E})^{e_D}$

Destinataire

$\left[(M^{d_E})^{e_D}\right]^{d_D} = (M^{d_E})$

(Déchiffrement)

$(M^{d_E})^{e_P}$ = Charabia

(verif. signature *pirate*)

Pirate

$(M^{d_E})^{e_P}$ Scénario 1 : renvoi du message charabia (non modifié) avec une insulte (c'est quoi ce charabia)

$\left(\left((M^{d_E})^{e_P}\right)^{d_D}\right)^{e_P}$ Scénario 2 : envoi d'un accusé de réception

Dans les deux cas, le pirate a le message !

Chiffrement + Signature à clé publique : moralités

- Ne jamais envoyer de résultats d'opérations où l'on a utilisé sa clé privée sur des fichiers incompréhensibles (=> pas d'accusés de réception automatiques)
- Utiliser des paires de clés différentes pour la signature et le chiffrement
- Ne pas signer le message lui-même, mais une *empreinte* du message, générée par une *fonction de hachage à sens unique* (voir ci-après)
- Moyennant ces précautions, la signature numérique à clé publique est sûre dans l'état actuel des connaissances. L'algorithme le plus utilisé combine RSA avec (SHA, fonction de hachage à sens unique décrite ci-après)

Signature avec RSA : Exemple réel : cartes bancaires (1/3)

La puce d'une carte bancaire est véritable un petit ordinateur muni d'un CPU, de RAM de ROM et d'EEPROM. Pour authentifier une carte (garantir qu'elle a bien été émise par le GIE cartes bancaires), y sont stockées entre autres :

- Une Valeur d'Identification (VI) composée à partir des informations concernant la carte (porteur, numéro, date de validité, devise)
- Une Valeur d'Authentification (VA) qui est le chiffré RSA de la VI avec la clé privée du GIE

Quand une carte est introduite dans un terminal, la puce vérifie que le déchiffré de la VA avec la clé publique du GIE est égal à la VI

Signature avec RSA : Exemple réel : cartes bancaires (2/3)

- La clé publique du GIE était jusqu'à récemment e=3 et n=213598703592091008239502270499962879705109534182641740644252416500858395774644508840500943086599 9 (nombre sur 320 bits soit 96 digits)
- n fut factorisé et le résultat publié sur un forum : p=1113954325148827987925490175477024844070922844843 q=191748170252450443937578626823086218069693418929 3
- Il ne restait plus qu'à calculer (p-1)(q-1) puis l'inverse de e modulo (p-1)(q-1) pour trouver d, la clé privée
- Cette faille fut l'un des ingrédients de la fabrication des Yescards dites « méthode Humpich »

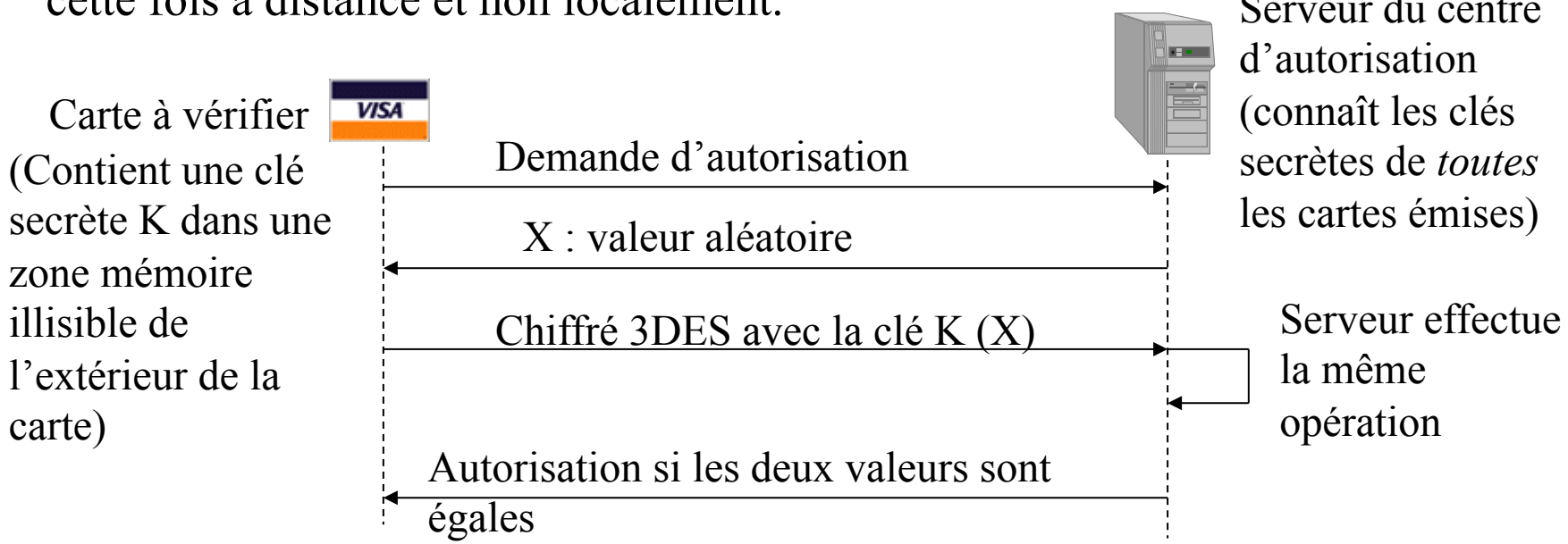
Signature avec RSA : Exemple réel : cartes bancaires (3/3)

- La clé publique du GIE est désormais e=3 et n=1550880802783769298423921500751307878471020215206711102793111990113875394553459999757605304671735856091597555389797408938173344043674704780986390069906679096728933081405044935969514508676239942493440750589270015739962374529363251827 (nombre sur 768 bits soit 231 digits).
- Bien que non cassée à ce jour, il n'est pas à exclure que cette clé puisse l'être dans un futur relativement proche, au vu des progrès technologiques.

Sécurité des cartes bancaires (suite)

- Le code à 4 chiffres (PIN) à saisir pour certains terminaux, vérification locale : pas efficace contre les faussaires experts
- La signature par VA/VI évoquée plus haut
- Au-delà d'un certain montant et périodiquement de manière aléatoire, vérification distante : Appel… Demande d'autorisation.

Quelle est cette autorisation ? Vérification que votre compte est approvisionné ? Pas du tout… C'est une nouvelle vérification de l'authenticité de la carte, effectuée cette fois à distance et non localement.



Fonctions de hachage à sens unique

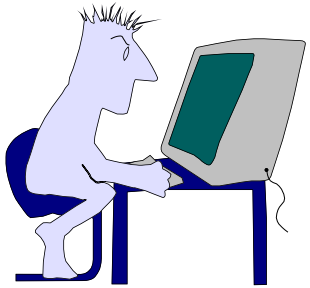
- Fonction de hachage : fonction prenant en entrée un message de longueur variable et générant une empreinte de taille fixe.
- Empreinte : permet de s'assurer avec un certain niveau de vraisemblance qu'un message n'a pas été altéré accidentellement (bits de parité, codes détecteurs d'erreurs)
- Fonction de hachage *à sens unique* (One-way hash function) : permet de s'assurer qu'un message n'a pas été altéré *intentionnellement*. Donc :
- Calcul de l'empreinte : facile, algorithme connu de tous,
- Recherche d'un message d'empreinte donnée : très difficile
- Pour les fonctions *résistantes aux collisions* : recherche de deux messages de même empreinte : très difficile

Fonctions de hachage à sens unique et signature

L'ensemble peut si besoin être chiffré (clé publique du destinataire)

- Emetteur :
  Calcule H(M)
  Chiffre H(M) avec sa clé privée d_E
  Envoie M et $(H(M))^{d_E}$

- Destinataire
  Calcule H(M)
  Déchiffre $(H(M))^{d_E}$ avec clé publique émetteur
  Vérifie $\left((H(M))^{d_E}\right)^{e_E} = H(M)$



- Pirate (cherchant à substituer le message par un autre) doit trouver M' tel que H(M')=H(M)

Attaque « des anniversaires » (1/2)

- Problème1 : Combien doit on réunir de personnes pour avoir plus d'une chance sur 2 d'en avoir une ayant une date anniversaire donnée ? Réponse : **253**
- Problème2 : Combien doit on réunir de personnes pour avoir plus d'une chance sur 2 d'en avoir deux ayant la même date anniversaire ? Réponse : **23 (!)**
- Argument: s'il y a 23 personnes, il y a 23*22/2=253 *couples*
- Démonstration complète : Pour *ne pas avoir* même anniversaire les probabilités valent :

$$\left(\frac{364}{365}\right)^{N} \text{ (Problème 1)} \qquad \prod_{i=1}^{N}\left(\frac{365-i+1}{365}\right) \text{ (Problème 2)}$$

Attaque « des anniversaires » (2/2)

Attaque (redoutable !) basée sur ce principe :

- Préparer deux contrats, l'un favorable l'autre défavorable,
- Générer automatiquement des variantes de chacun avec des changements cosmétiques (genre espace/retour arrière), calculer l'empreinte de chaque en recherchant des paires de même empreinte (avec modif/non modif à chaque ligne d'un document de 32 lignes crée aisément 2^{32} versions),
- Dès qu'une collision a été trouvée, faire signer l'élément de la paire qui lui est favorable à la future victime,
- Un peu plus tard affirmer preuve à l'appui qu'elle a signé l'autre...

Attaque « des anniversaires »: Moralités

- Pour une taille d'empreinte donnée N : Nombre de messages nécessaires pour avoir plus d'une chance sur 2 d'en trouver un d'empreinte donnée : 2^{N-1}
- Nombre de messages nécessaires pour avoir plus d'une chance sur 2 d'en trouver deux de même empreinte $2^{N/2}$
- La taille N de l'empreinte doit donc être choisie pour que $\mathbf{2^{N/2}}$ soit grand (au moins 128 bits)
- La fonction de hachage doit être choisie résistante aux collisions
- **Précaution** : Faites toujours des changements cosmétiques dans les documents avant de les signer, vous mettrez ainsi en échec l'attaque des anniversaires

Fonction de hachage à sens unique : MD5 (1/8)

- MD5 est la fonction de hachage utilisée en standard avec RSA pour les signatures numériques. Elle génère une empreinte de 128 bits à partir du message traités par blocs de 16 mots de 32 bits (512 bits)
- La convention adoptée est globalement Little Endian : les bits d'un octet sont donnés avec le bit de poids fort en tête

$$b_7b_6b_5b_4b_3b_2b_1b_0 = \sum_{i=0}^{7} 2^i b_i$$

- Par contre les octets d'un mot sont donnés avec l'octet de poids faible en tête

$$B_0B_1B_2B_3 = \sum_{i=0}^{3} 256^i B_i$$

Fonction de hachage à sens unique : MD5 (2/8)

- Le message est dans un premier temps complété pour que sa longueur devienne un multiple de 512 en procédant ainsi :
- Ajouter un bit 1 (toujours) et autant de 0 qu'il faut pour amener la longueur du message congruente à 448 mod 512
- Rajouter une représentation sur 64 bits de la longueur avant l'étape précédente (si $L>2^{64}$ soit 2 millions de To… prendre les bits de poids faible), toujours en convention Little Endian sur les 8 octets
- Le message de longueur multiple de 512 bits sera traité par blocs de 16 mots de 32 bits (notés M[0] à M[15])
- Initialiser 4 variables sur 32 bits : A=0x01234567, B=0x89abcdef, C=0xfedcba98, D=0x76543210

Fonction de hachage à sens unique : MD5 (3/8)

- L'algorithme a 4 rondes de 16 étapes, une des 4 fonctions suivantes est utilisée à chaque ronde :

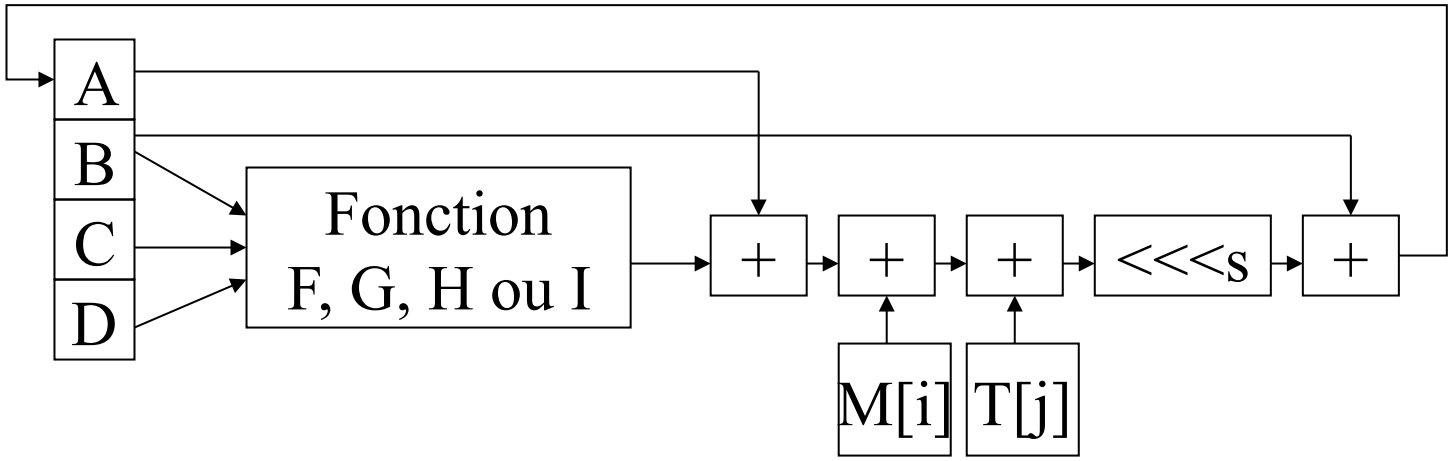
$$F(X,Y,Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$$
$$G(X,Y,Z) = (Z \wedge X) \vee ((\neg Z) \wedge Y)$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$
$$I(X,Y,Z) = Y \oplus ((\neg Z) \vee X)$$

X, Y, Z : sur 32 bits
Les opérations sont respectivement les ET, OU, NOT et XOR bit à bit avec les notations usuelles

- L'opération shift circulaire à gauche de s bits est notée <<<s
- L'algorithme utilise également un tableau de 64 entiers sur 32 bits T[1..64] généré par :
- T[i] = Int(2^{32}*abs(sin(i)) avec i en radians

Fonction de hachage à sens unique : MD5 (4/8)

- A chaque étape de chaque ronde une fonction non linéaire est effectuée sur 3 des variables A, B, C, D (ex. B, C, D), le résultat est ajouté à (ex. A) à un des 16 mots du bloc de message et à un élément du tableau T. Le résultat est shifté de s bits, ajouté à (ex. B) puis substitué à (ex. A)

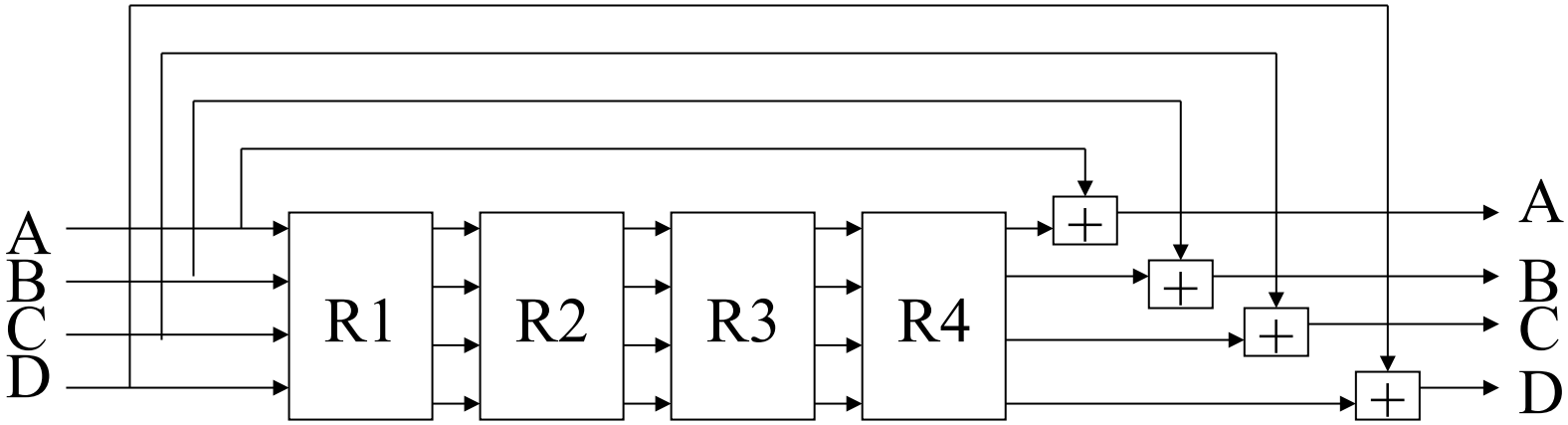


Fonction de hachage à sens unique : MD5 (5/8)

- Les additions sont effectuées modulo 2^{32}
- Veiller à respecter la convention Little Endian pour toutes les conversions de mots de 32 bits (valeurs de A, B, C, D des mots M[i] du message, et de ceux de T[i])
- Mais attention, il y a une exception : pour l'opération shift à gauche, il faut écrire le mot en convention Big Endian (Octet de poids fort en tête) effectuer le shift, puis évaluer le mot résultant en convention Big Endian
- Cela a une certaine logique lorsqu'il s'agit d'un shift, mais est source de confusions très fréquentes jusque dans certains ouvrages...

Fonction de hachage à sens unique : MD5 (6/8)

- A l'issue des 4 rondes (4*16 étapes) les nouvelles valeurs de A, B, C et D sont ajoutées aux anciennes (qui avaient été conservées dans des variables AA, BB, CC et DD). Ce nouveau jeu de A, B, C et D qui a « digéré » un bloc de message sert d'entrée à la « digestion » du bloc suivant



L'empreinte finale est la concaténation du dernier jeu ABCD

Fonction de hachage à sens unique : MD5 (7/8)

- Spécification ronde 1 : on note [ABCD k s i] l'opération :
  A = B + ((A + F(B,C,D) + M[k] + T[i])<<<s)

[ABCD 0 7 1] [DABC 1 12 2] [CDAB 2 17 3] [BCDA 3 22 4]
[ABCD 4 7 5] [DABC 5 12 6] [CDAB 6 17 7] [BCDA 7 22 8]
[ABCD 8 7 9] [DABC 9 12 10] [CDAB 10 17 11] [BCDA 11 22 12]
[ABCD 12 7 13] [DABC 13 12 14] [CDAB 14 17 15] [BCDA 15 22 16]

- Spécification ronde 2 : on note [ABCD k s i] l'opération :
  A = B + ((A + G(B,C,D) + M[k] + T[i])<<<s)

[ABCD 1 5 17] [DABC 6 9 18] [CDAB 11 14 19] [BCDA 0 20 20]
[ABCD 5 5 21] [DABC 10 9 22] [CDAB 15 14 23] [BCDA 4 20 24]
[ABCD 9 5 25] [DABC 14 9 26] [CDAB 3 14 27] [BCDA 8 20 28]
[ABCD 13 5 29] [DABC 2 9 30] [CDAB 7 14 31] [BCDA 12 20 32]

Fonction de hachage à sens unique : MD5 (8/8)

- Spécification ronde 3 : on note [ABCD k s i] l'opération :
  A = B + ((A + H(B,C,D) + M[k] + T[i])<<<s)

[ABCD 5 4 33] [DABC 8 11 34] [CDAB 11 16 35] [BCDA 14 23 36]
[ABCD 1 4 37] [DABC 4 11 38] [CDAB 7 16 39] [BCDA 10 23 40]
[ABCD 13 4 41] [DABC 0 11 42] [CDAB 3 16 43] [BCDA 6 23 44]
[ABCD 9 4 45] [DABC 12 11 46] [CDAB 15 16 47] [BCDA 2 23 48]

- Spécification ronde 4 : on note [ABCD k s i] l'opération :
  A = B + ((A + I(B,C,D) + M[k] + T[i])<<<s)

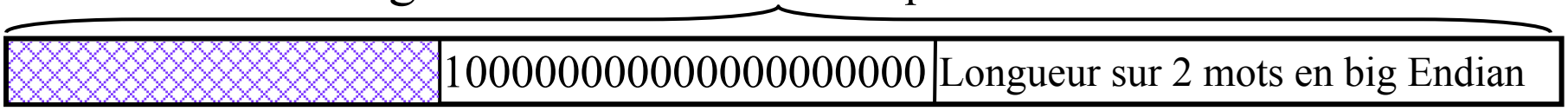
[ABCD 0 6 49] [DABC 7 10 50] [CDAB 14 15 51] [BCDA 5 21 52]
[ABCD 12 6 53] [DABC 3 10 54] [CDAB 10 15 55] [BCDA 1 21 56]
[ABCD 8 6 57] [DABC 15 10 58] [CDAB 6 15 59] [BCDA 13 21 60]
[ABCD 4 6 61] [DABC 11 10 62] [CDAB 2 15 63] [BCDA 9 21 64]

Fonctions de hachage : SHA

- Des faiblesses (collisions d'empreinte) ayant été mises en évidence dans MD5 (lors du congrès Crypto 2004 par une équipe Chinoise) son usage n'est désormais plus recommandé.
- Il faut désormais lui préférer les SHA qui se déclinent en plusieurs versions : SHA0 (obsolète), SHA1 (pour lequel des faiblesses ont également été trouvées) et SHA256, SHA384 et SHA 512 décrits ci-après.

SHA256, SHA384, SHA512

- Génèrent une empreinte de 256, 384 ou 512 bits en manipulant un message par blocs de :
  - 512 bits traités comme 16 mots de 32 bits (SHA256)
  - 1024 bits traités comme 16 mots de 64 bits (SHA384 et SHA512)
- Convention « big Endian » (toujours octets de poids fort à gauche)
- Le message est préablement complété (remplissage ou « padding ») avant le hachage :



Rajouter 1 (toujours) et autant de 0 qu'il faut pour qu'il reste juste deux mots (rajouter au besoin un bloc) : longueur du bloc à ce stade : 448 (SHA256) ou 896 (SHA384 et 512)

Rajouter enfin sur les deux derniers mots (64 ou 128 bits) la longueur du message

Longueur de message donc limitée à 2^{64} ou 2^{128} mais cela fait 2 ou 4 Millions de To…

SHA256, 384 et 512

L'empreinte ou « hash » est alors initialisée à une valeur $H^{(0)}$ décomposée en 8 mots $H_0^{(0)} H_1^{(0)} H_2^{(0)} H_3^{(0)} H_4^{(0)} H_5^{(0)} H_6^{(0)} H_7^{(0)}$

Ces 8 mots sont :

•Les 32 premiers bits de la partie fractionnaire de la racine carrée des 8 premiers nombres premiers pour SHA256 (première valeur pour le nombre 2 : $H_0^{(0)}$=6A09E667)

•Les 64 premiers bits de la partie fractionnaire de la racine carrée des 9ème au 16ème nombres premiers pour SHA384

•Les 64 premiers bits de la partie fractionnaire de la racine carrée des 8 premiers nombres premiers pour SHA512

Nota : SHA384 est identique dans son algorithme à SHA512 : la seule différence est la valeur initiale et la troncature du hash final à ses 384 bits de poids fort

SHA256, 384 et 512

- Chaque bloc du message est alors successivement « haché » produisant une nouvelle valeur du hash (fonctions non linéaires de la valeur précédente, du bloc de message et de constantes) qui est utilisée pour le hachage du bloc suivant :

Pour préparer le hash numéro i, hacher finement :

Le hash précédent : $H^{(i-1)}$

Le bloc de message i : $M^{(i)}$

Les constantes de la recette K_t

La dernière valeur du hash (tronquée éventuellement pour SHA384) est l'empreinte finale.

SHA256, 384 et 512

- Chaque hachage de bloc est décomposée en 64 (SHA256) ou 80 (SHA384 et 512) étapes traitant chacun un mot W_t (t de 0 à 63 ou 79)
- Les 16 premiers mots sont ceux du bloc de message, $M^{(i)} = M_0^{(i)} M_1^{(i)} ... M_{15}^{(i)}$
- Les autres sont fabriqués à partir de ces mots à l'aide de fonctions σ_0 et σ_1 qui diffèrent entre SHA256 et 512 (ou 384) : XOR de 3 shifts à droite dont 2 circulaires de leur mot argument

$$W_t = M_t^{(i)} \quad 0 \le t \le 15$$

$$W_t = \sigma_1^{\{256 ou 512\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256 ou 512\}}(W_{t-15}) + W_{t-16} \quad 16 \le t \le 63 (ou 79)$$

$$\sigma_0^{\{256\}}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$$

$$\sigma_1^{\{256\}}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$$

$$\sigma_0^{\{512\}}(x) = ROTR^1(x) \oplus ROTR^8(x) \oplus SHR^7(x)$$

$$\sigma_1^{\{512\}}(x) = ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^6(x)$$

+ : addition modulo 2^{32} ou 2^{64}

SHR^n : shift à droite de n bits (=division par 2^n)

$ROTR^n$: shift circulaire à droite de n bits

SHA256, 384 et 512

Hachage d'un bloc :

Initialiser (à la valeur du hash précédent) 8 variables (vont évoluer lors du traitement de chaque mot) :

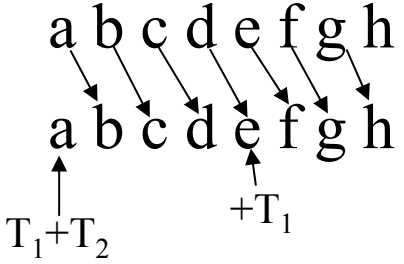
$$a = H_0^{(i-1)} \quad b = H_1^{(i-1)} \quad c = H_2^{(i-1)} \quad d = H_3^{(i-1)} \quad e = H_4^{(i-1)} \quad f = H_5^{(i-1)} \quad g = H_6^{(i-1)} \quad h = H_7^{(i-1)}$$

Pour chaque mot W_t construit comme indiqué ci-avant calculer (Ch, Maj, Σ_0 et Σ_1 sont des fonctions K_t un ensemble de constantes : Σ_0, Σ_1 et K_t diffèrent entre SHA256 et 512) :

$$T_1 = h + \Sigma_1^{\{256ou512\}}(e) + Ch(e,f,g) + K_t^{\{256ou512\}} + W_t$$

$$T_2 = \Sigma_0^{\{256ou512\}}(a) + Maj(a,b,c)$$ puis faire : h=g; g=f; f=e; e=d+T_1;d=c; c=b; b=a; a=T_1+T_2

Une fois traité le dernier mot de la séquence (64ème ou 80ème), incrémenter les différents mots du hash des mots a,b,c,d,e,f,g,h



$$H_0^{(i)} = a + H_0^{(i-1)} \quad H_1^{(i)} = b + H_1^{(i-1)} \quad H_2^{(i)} = c + H_2^{(i-1)} \quad H_3^{(i)} = d + H_3^{(i-1)}$$

$$H_4^{(i)} = e + H_4^{(i-1)} \quad H_5^{(i)} = f + H_5^{(i-1)} \quad H_6^{(i)} = g + H_6^{(i-1)} \quad H_7^{(i)} = h + H_7^{(i-1)}$$

SHA256, 384 et 512 : Fonctions et constantes

$$Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z)$$
$$Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$
$$\Sigma_0^{\{256\}}(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x)$$
$$\Sigma_1^{\{256\}}(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x)$$
$$\Sigma_0^{\{512\}}(x) = ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x)$$
$$\Sigma_1^{\{512\}}(x) = ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x)$$

Constantes K_t :

•SHA256 : Le début (sur 32 bits) de la partie fractionnaire des racines cubiques des 64 premiers nombres premiers

•SHA512 : Le début (sur 64 bits) de la partie fractionnaire des racines cubiques des 80 premiers nombres premiers

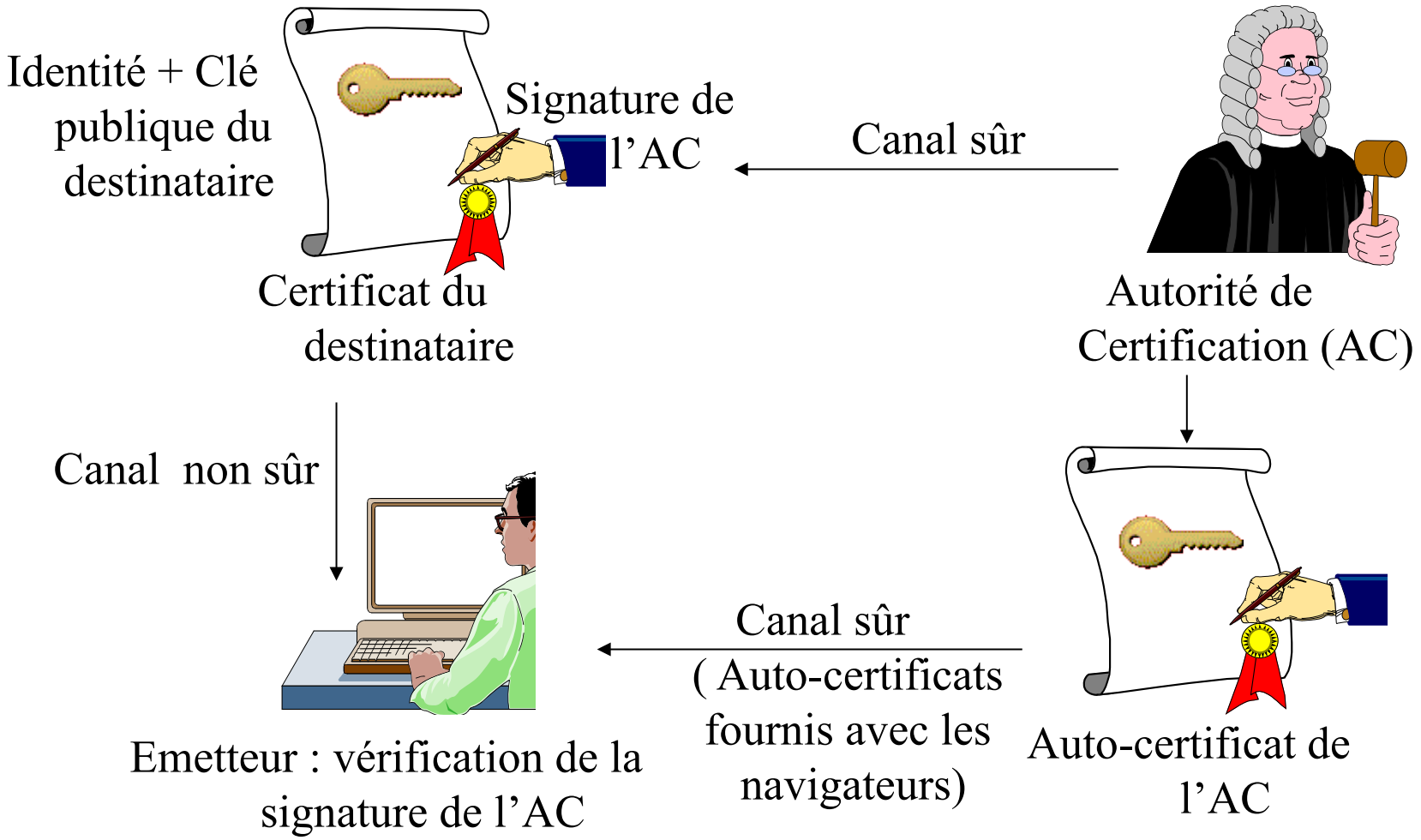
Certificats (1/3)

- Alice : Comment pouvons nous échanger des messages de manière confidentielle ?
- Bob : On se met d'accord sur une clé secrète et on chiffre nos messages avec AES.
- Alice : Comment échanger cette clé sans qu'elle soit interceptée par un indiscret ?
- Bob : Tu m'envoie ta clé publique RSA et je t'envoie la clé secrète AES.
- Alice : Comment vas tu être sûr que la clé publique que tu vas recevoir est bien la mienne et pas celle d'un pirate qui l'a substituée lors de l'envoi ?

Certificats (2/3)

- Le scénario qui précède est un point faible de la cryptographie à clé publique : l'attaque de l'intermédiaire (Man in the middle).
- Si les clés publiques sont échangées de façon non sûre (envoyées par mail par exemple) elles peuvent être substituées.
- L'intermédiaire peut alors intercepter les messages destinés à sa victime les déchiffrer, les rechiffrer avec la véritable clé publique du destinataire, et les ré-envoyer.
- Le certificat est un moyen de garantir qu'une clé publique appartient bien à la bonne personne.

Certificats (3/3)



Utilisation des certificats : Exemple de SSL

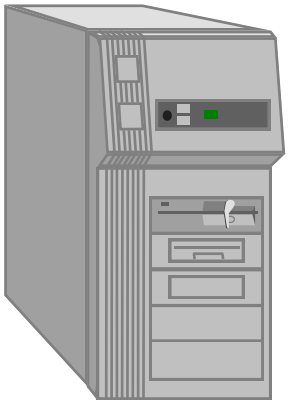
- Les navigateurs web intègrent des modules de cryptographie ainsi que des auto-certificats d'AC (voir Préférences/Avancé/Certificats/ sous Mozilla Firefox par exemple).
- Lorsque cela est nécessaire (transactions commerciales…), il est possible de chiffrer les communications (pages https://… et pictogramme cadenas fermé).
- Le mécanisme de transaction (totalement transparent pour l'utilisateur) est décrit ci-après.

SSL : mécanisme de transaction

Client

Vérif Certificat

Serveur du site marchand



Demande d'authentification + cryptosystèmes supportés

Certificat + cryptosystème compatible

[Si Certificat=OK]Clé de session chiffrée avec clé publique du serveur

Déchiffrement clé de session

Échanges chiffrés avec la clé de session (utilisation d'un algorithme symétrique)

Génération des grands nombres premiers

- Pour des grands nombres premiers les méthodes classiques (type crible d'Eratosthène) ne sont pas envisageables
- On utilise alors des tests de primalité qui rendent un résultat type : «p n'est certainement pas premier» ou «p est probablement premier»
- En répétant le test plusieurs fois, on parvient à générer des nombres pour lesquels les chances qu'ils ne soient pas premiers sont très faibles
- Le plus utilisé est le test de Rabin-Miller décrit ci-après

Génération des grands nombres premiers : Rabin-Miller (1/5)

Rabin-Miller est basé sur deux propriétés des nombres premiers :

- Pour tout a non nul et < p : $a^{p-1} \bmod p = 1$ (théorème de Fermat)
- Dans Z/pZ l'équation $x^2=1$ n'a que les deux solutions triviales x=1 et x=-1 (autre écriture de l'élément p-1)

En effet $x^2=1 <=> (x-1)(x+1)=0$ et comme Z/pZ est un corps pour p premier, si l'un des éléments x-1 ou x+1 est non nul il suffit de multiplier par son inverse pour conclure que l'autre est nul

Génération des grands nombres premiers : Rabin-Miller (2/5)

Étant donné un candidat nombre premier p et un nombre test non nul a<p on effectue les actions suivantes :

- Trouver b : nombre de fois que 2 divise p-1 (p-1 est pair donc b vaut au moins 1).
- Déterminer m tel que $p-1=2^b.m$
- Envisager la suite des $\left(a^m\right)^{2^j}$ (modulo p) pour 0<=j<=b (obtenue par élévations au carré successives à partir d'un élément initial a^m). Le dernier élément de la suite vaut $\left(a^m\right)^{2^b} = a^{2^b.m} = a^{p-1} = 1$ (modulo p)

d'après le théorème de Fermat

Génération des grands nombres premiers : Rabin-Miller (3/5)

Si p est effectivement premier, deux cas sont alors possibles :

- Soit le premier terme a^m vaut 1 ou -1 (modulo p) et auquel cas tous les autres termes valent 1
- Sinon les élévations au carré successives doivent obligatoirement faire tomber sur -1 pour un j<b

Il est en effet impossible de tomber sur 1 sans que la valeur précédente soit -1 car sinon on aurait trouvé une racine carrée de l'unité différente de 1 ou -1

On peut donc coder l'algorithme sous la forme :

RabinMiller(p : Candidat premier, a : Entier test<p) :
(NonPremier, ProbablementPremier)

Génération des grands nombres premiers : Rabin-Miller (4/5)

RabinMiller(p,a) : NonPremier, ProbablementPpremier

Diviser p-1 par 2 autant de fois que possible
b=Nombre de fois que 2 divise p-1
$m=(p-1)/2^b$
$z=a^m \bmod p$ (algorithme d'exponentiation rapide)
Si z=1 ou p-1 retourner ProbablementPremier
Pour 1=<j<b Boucle
    z=z*z mod p
    Si z=p-1 retourner ProbablementPremier
    Si z=1 retourner NonPremier
    j=j+1
FinBoucle
Retourner NonPremier

Fin RabinMiller

Génération des grands nombres premiers : Rabin-Miller (5/5)

On peut ainsi générer des nombres sur N bits très probablement premiers en faisant :

Générer un nombre aléatoire sur N bits

Mettre le premier bit à 1 (pour avoir la bonne taille)

Mettre le dernier bit à 1 (pour avoir un nombre impair)

Tester par prudence la divisibilité par tous les premiers <256

Faire 5 tests de RabinMiller pour 5 valeurs tests a aléatoires mais relativement petites (pour des raisons de rapidité des calculs)

Si le nombre candidat passe avec succès toutes ces étapes, la probabilité qu'il ne soit pas premier est très faible

Le plus grand nombre premier connu (23/8/2008)

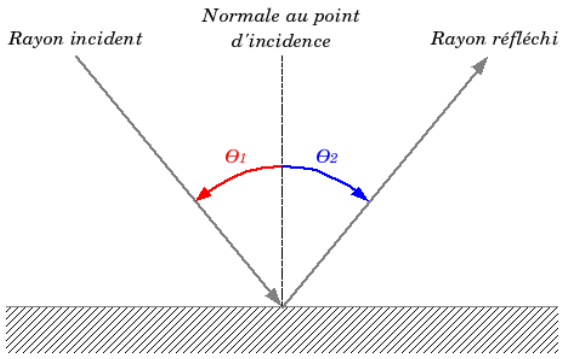
- Un projet informatique de longue haleine appelé Great Internet Mersenne Prime Search (GIMPS : http://www.mersenne.org/) permet de déterminer des grands nombre premier dits de Mersenne, du nom d'un mathématicien français du XVIIe siècle. Ce type de nombre, très rare, s'exprime sous la forme 2 puissance p moins 1, avec p également nombre premier. Ces nombres qui nécessiteraient des millions de digits (17 425 170 dans ce cas) pour être écrits sous forme développée (il faudrait un livre d'au moins 6000 pages), peuvent cependant être désignés par leur forme plus condensée de nombre de Mersenne :

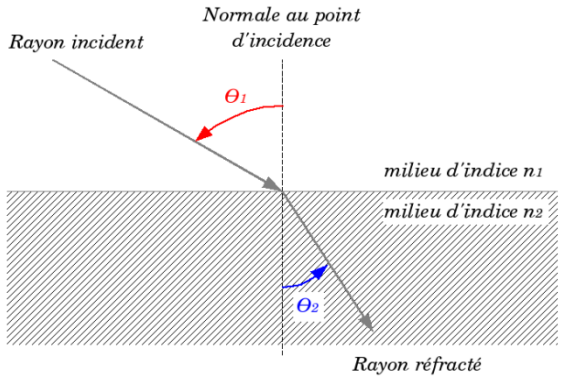
$$2^{57885161} - 1$$

Nota : il s'agit bien ici d'un nombre *certainement* premier

Cryptographie Quantique : principaux concepts de physique quantique

- Newton considérait la lumière comme un flux de particules matérielles et s'opposait ainsi à Huyghens qui y voyait une onde. Les lois de l'optique géométrique ont en effet des interprétations mécanistes ou ondulatoires.
- Les équations de Maxwell (1865) posant les bases de l'électromagnétisme semblaient conclure la question, mais elles posaient aussi les questions à l'origine de la théorie de la relativité…



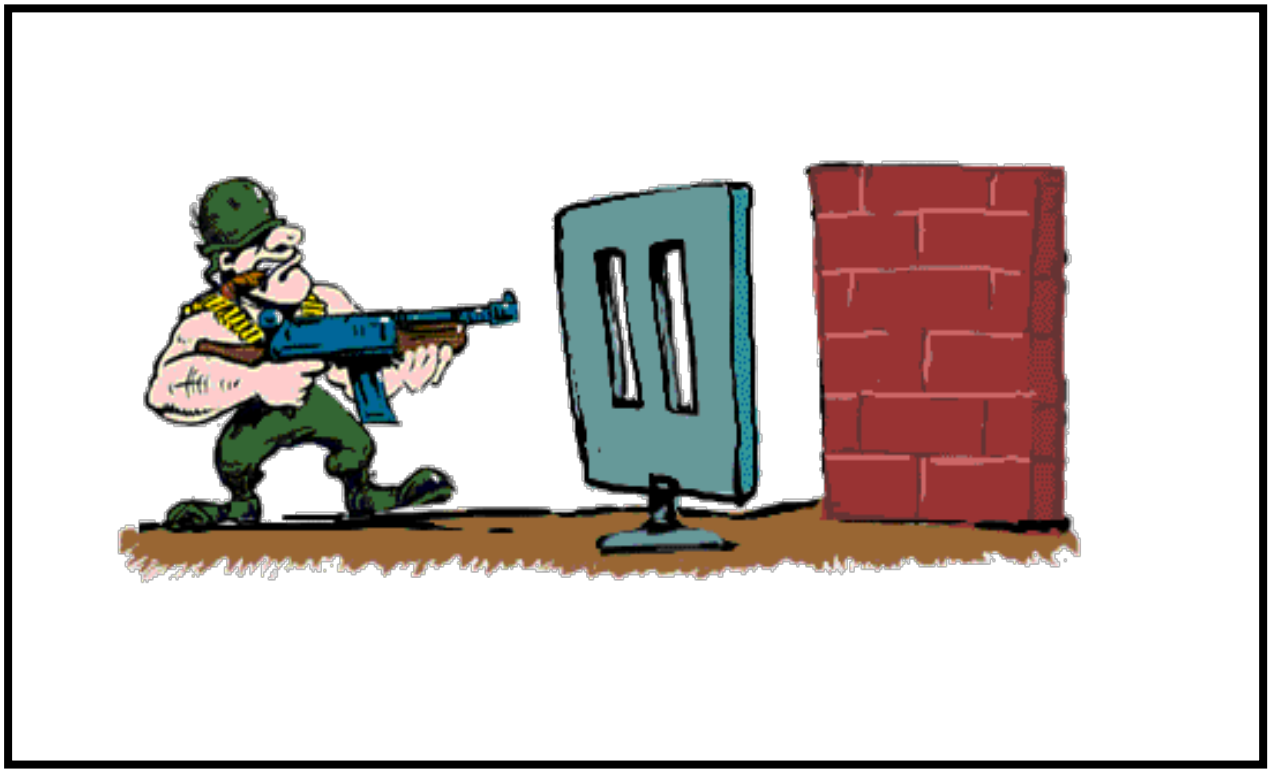


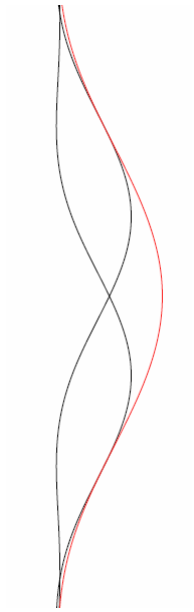
Cryptographie Quantique : principaux concepts de physique quantique

- L'étude du rayonnement du corps noir et celui de l'effet photo-électrique conduisirent Planck et Einstein (1905) à conclure à des comportements corpusculaires de la lumière.
- En effet dans un cas comme dans l'autre les résultats expérimentaux conduisaient à introduire des quantités élémentaires « quantas » d'énergie lumineuse : les photons.
- Alors lumière = onde ou corpuscule ?

L'expérience des fentes d'Young

- L'expérience avec des balles de fusil :



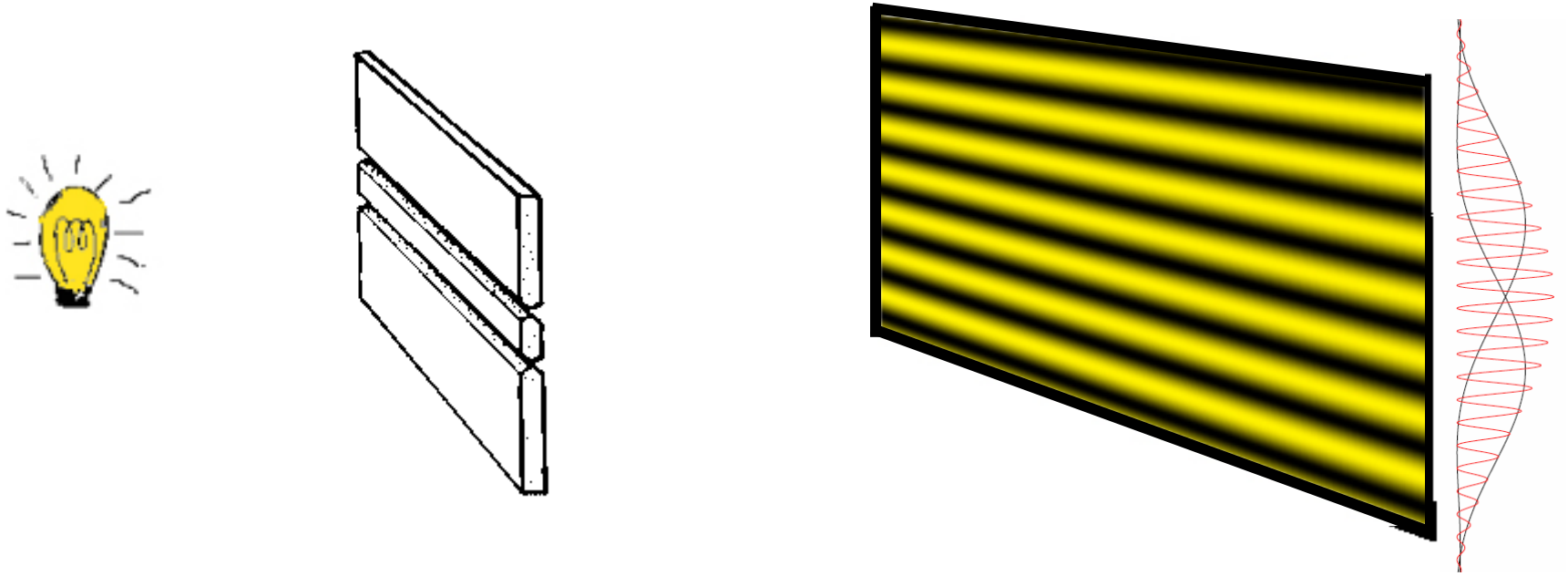


Additivité des probabilités d'atteindre un point donné de la cible

P(cible)=P(cible | Fente 1) + P(cible | Fente 2)

L'expérience des fentes d'Young

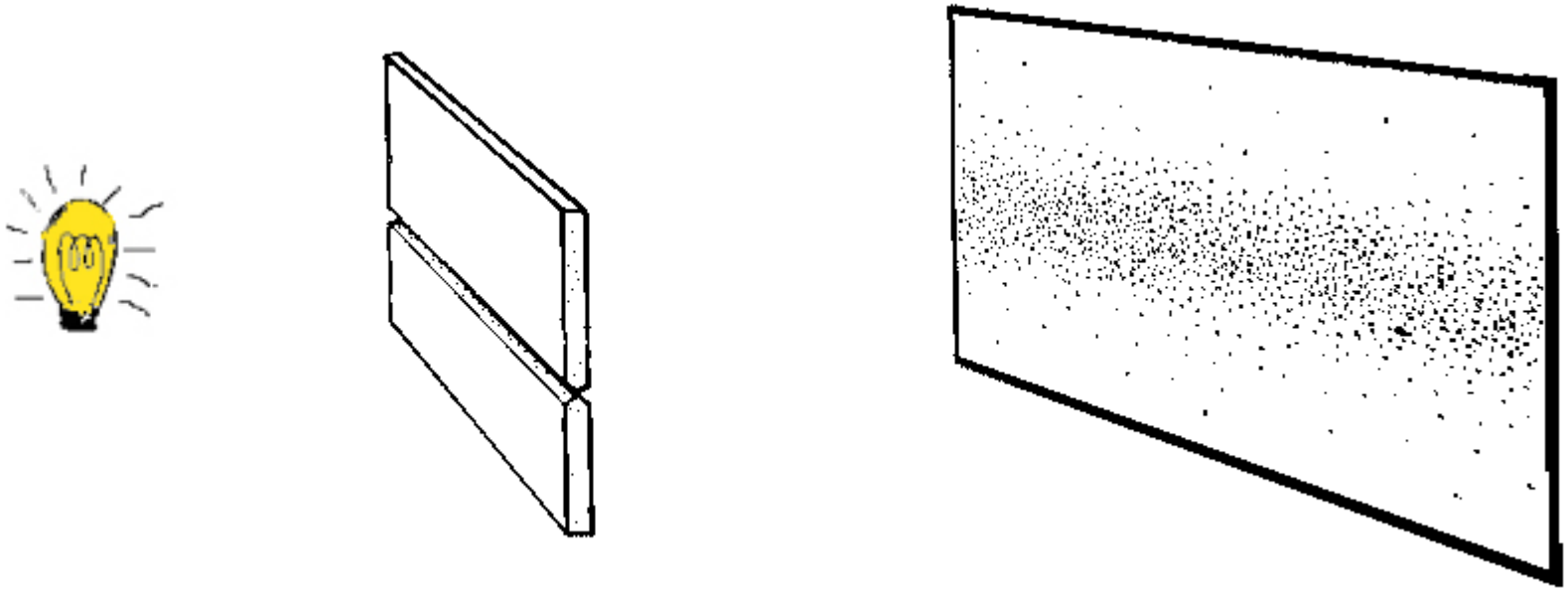
- L'expérience avec la lumière : fut l'un des arguments majeurs en faveur de la théorie ondulatoire :



- Franges d'interférence dues à l'addition constructive (en phase) ou destructive (en opposition de phase) des ondes provenant des fentes 1 et 2

L'expérience des fentes d'Young

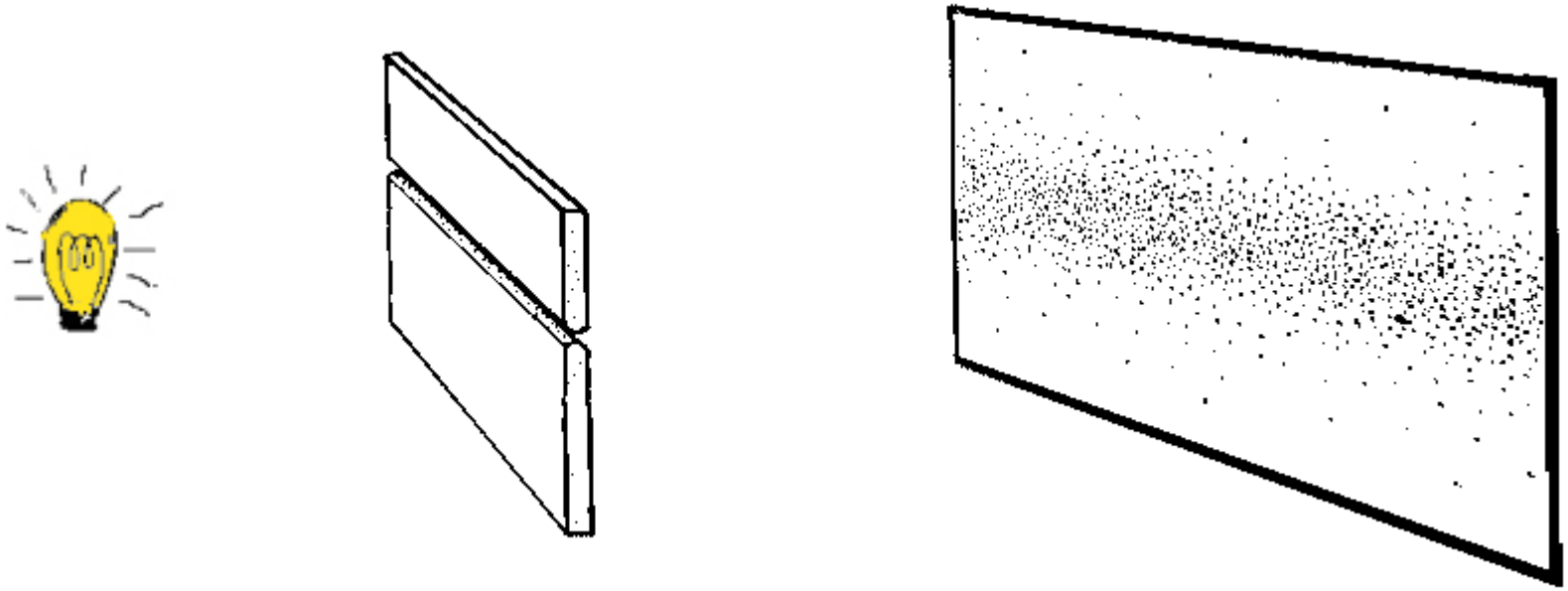
- Que se passe-t-il si la source lumineuse envoie les photons un par un ?



- Photons diffractés par la seule fente 1…

L'expérience des fentes d'Young

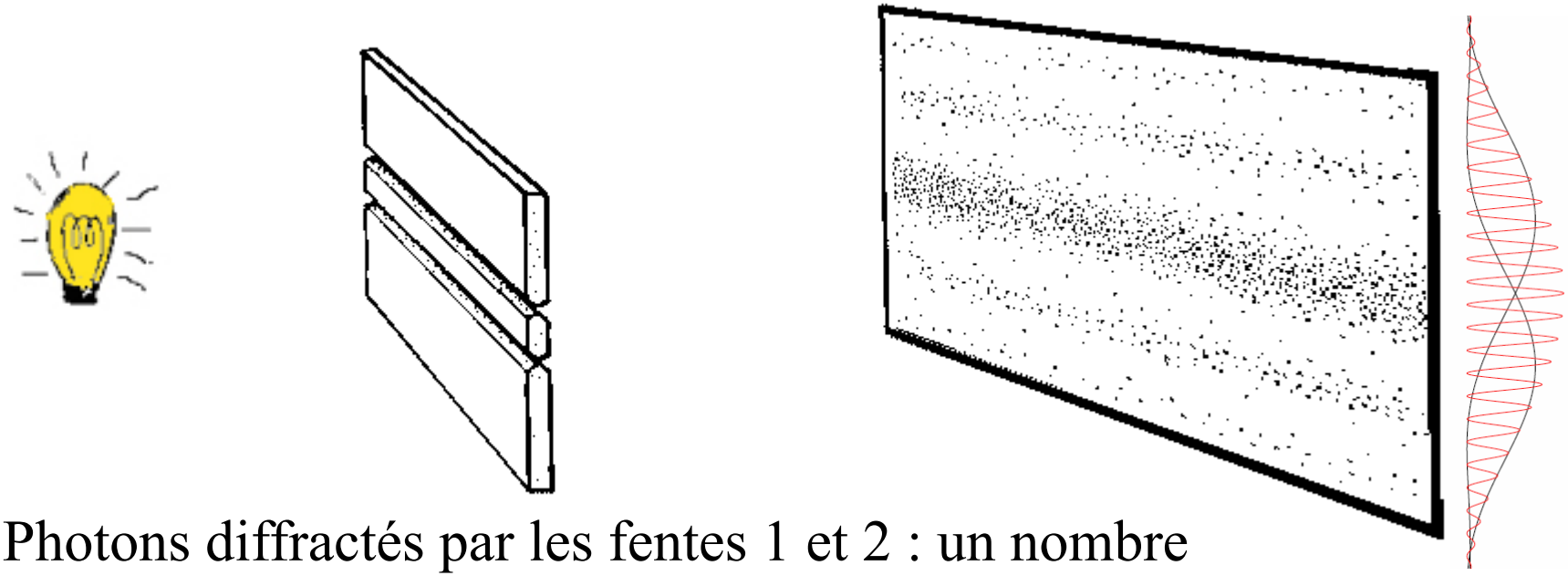
- Que se passe-t-il si la source lumineuse envoie les photons un par un ?



- Photons diffractés par la seule fente 2…

L'expérience des fentes d'Young

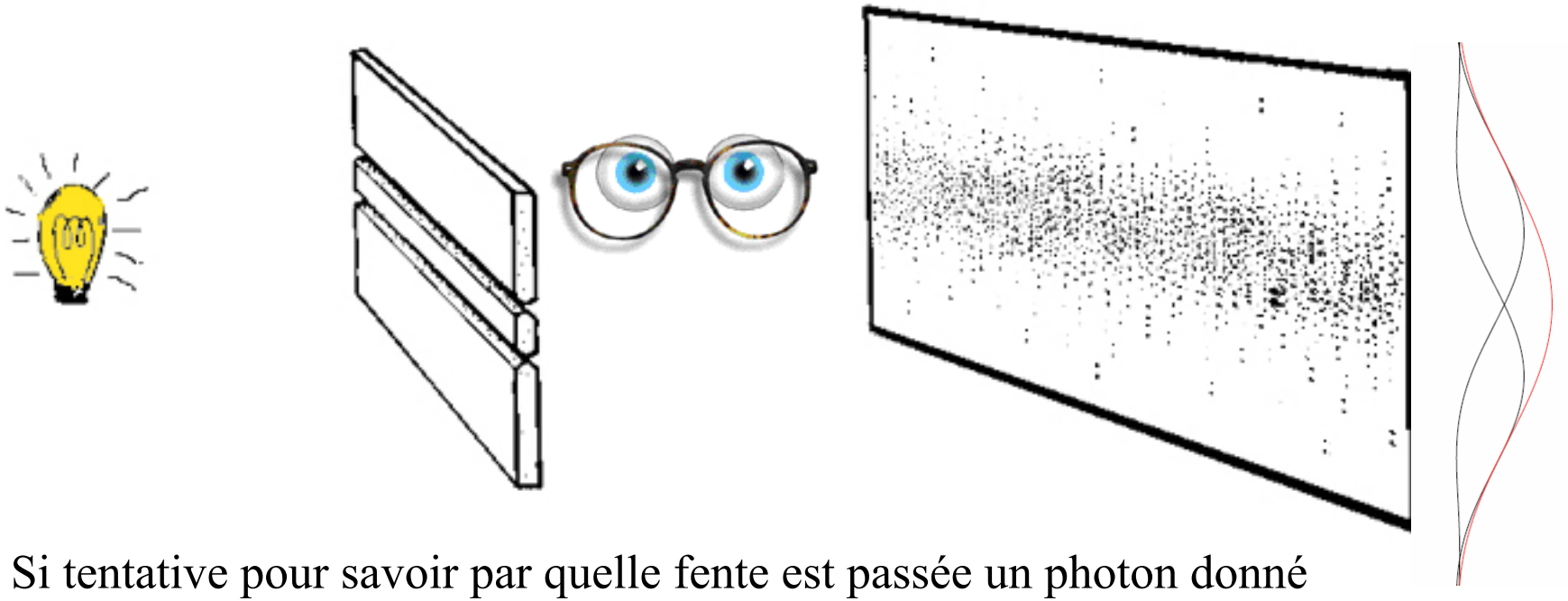
- Que se passe-t-il si la source lumineuse envoie les photons un par un ?



- Photons diffractés par les fentes 1 et 2 : un nombre suffisamment grand d'impacts reconstitue en moyenne une figure d'interférence… Lumière = « Onduscule » ?

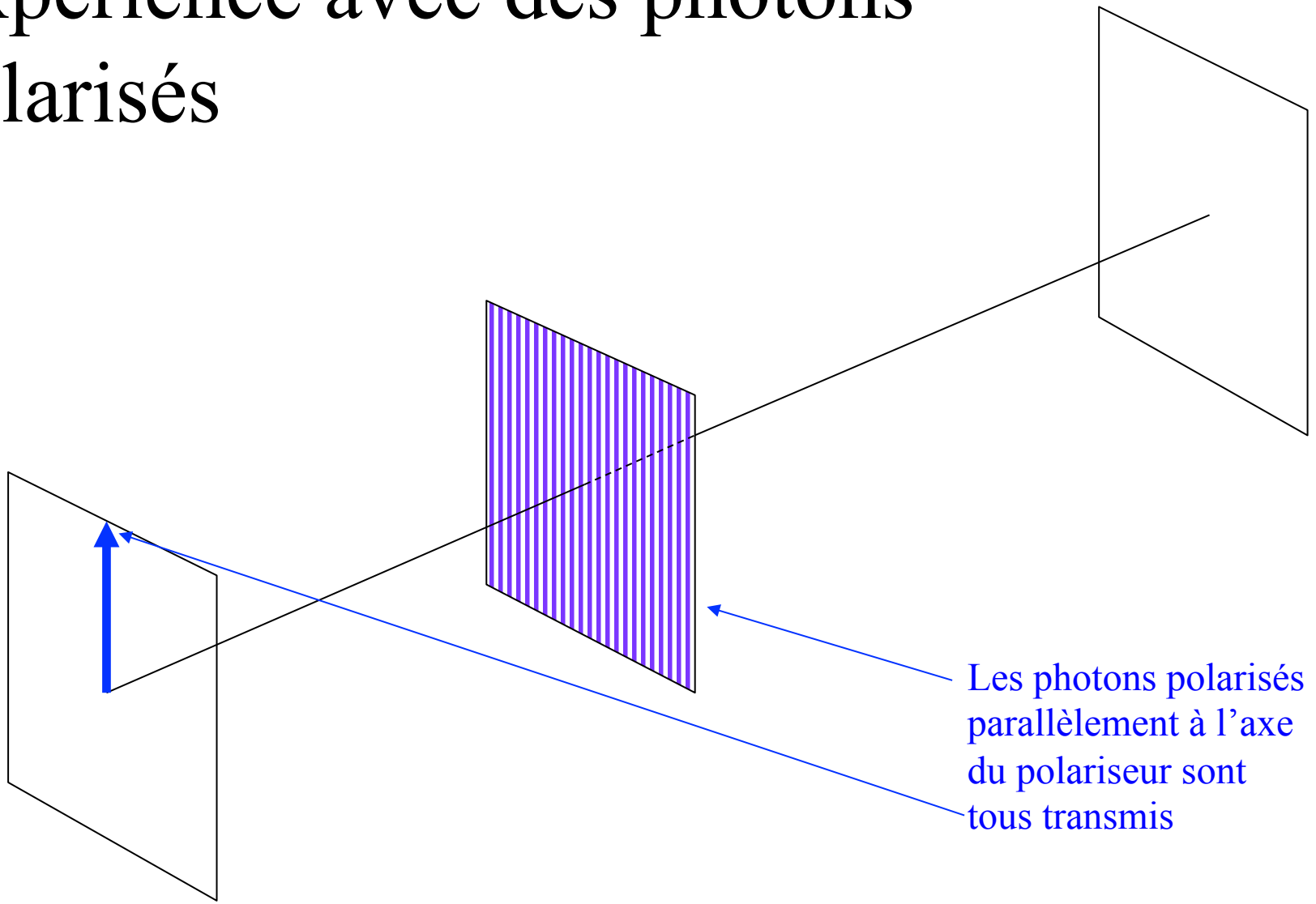
L'expérience des fentes d'Young

- Que se passe-t-il si la source lumineuse envoie les photons un par un ?

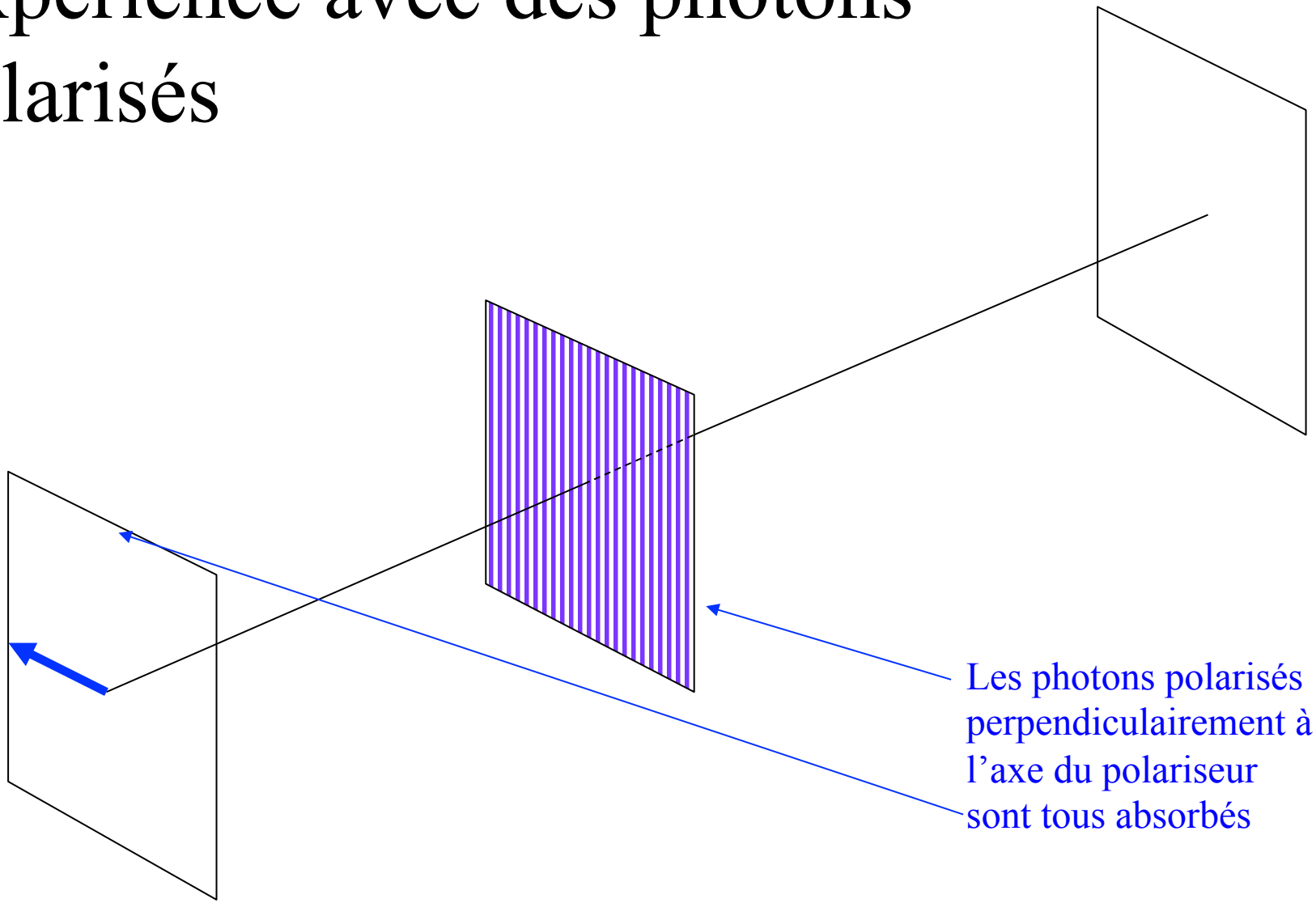


- Si tentative pour savoir par quelle fente est passée un photon donné (instrument de mesure…) => retour au cas des balles de fusil…

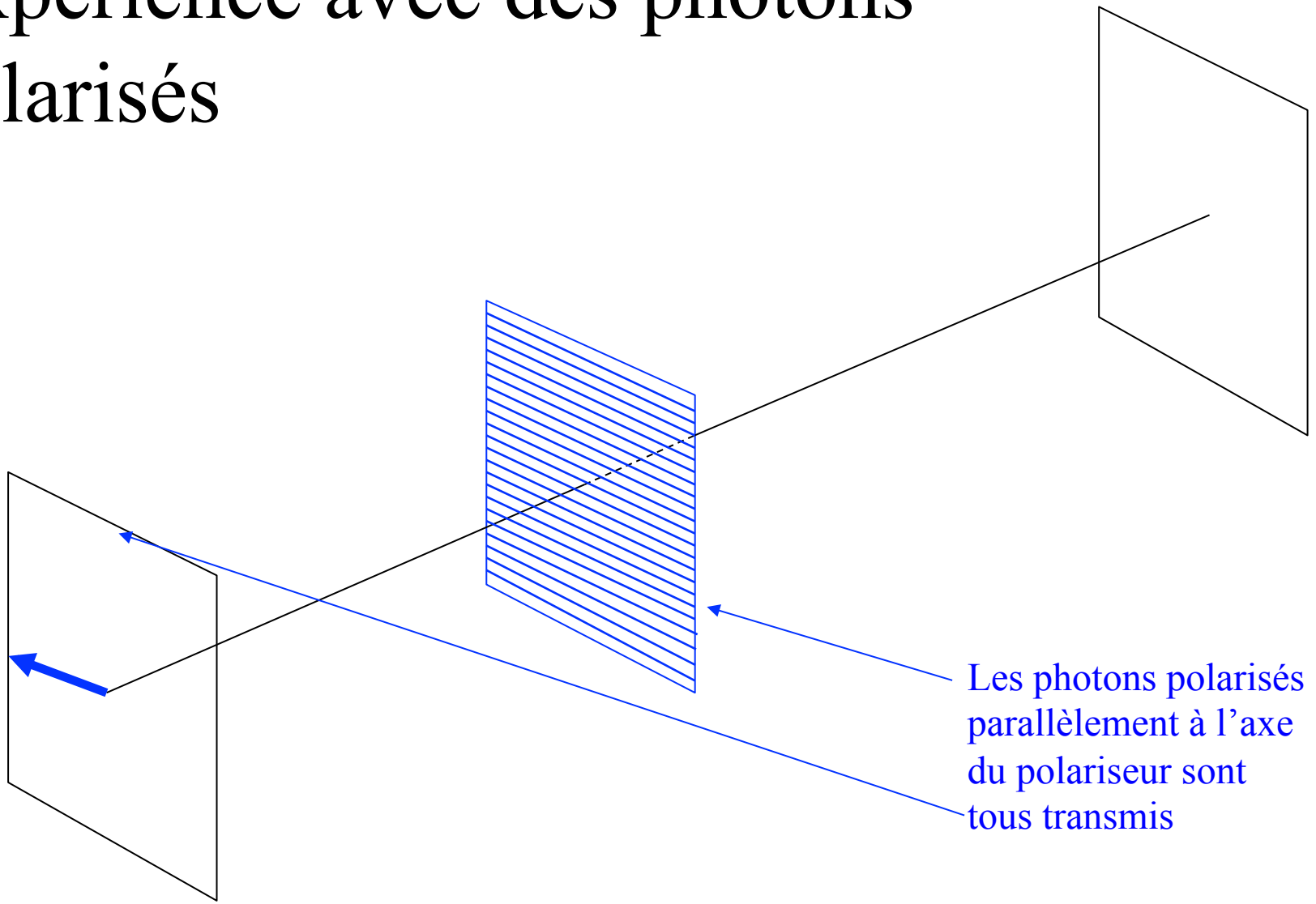
Expérience avec des photons polarisés



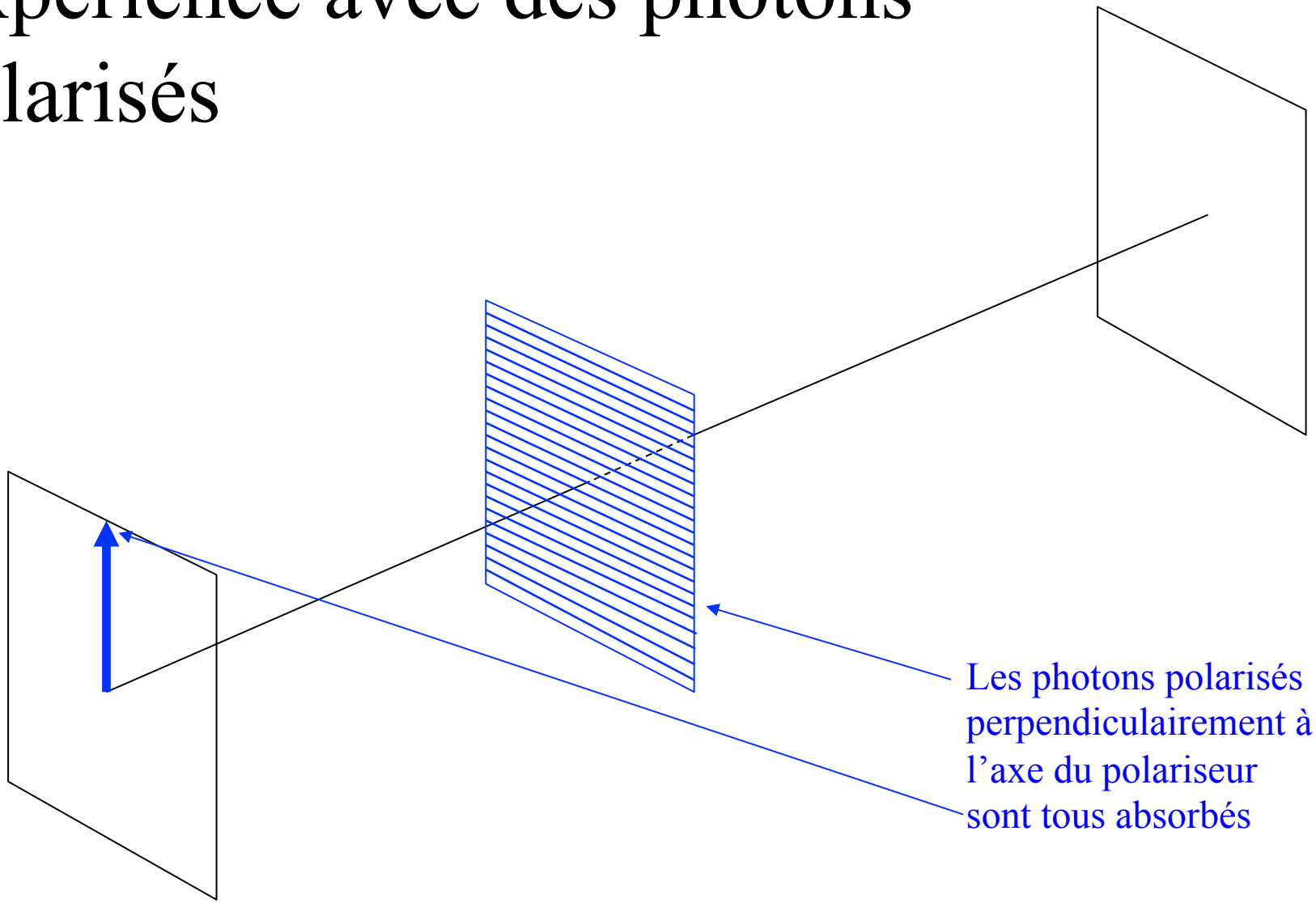
Expérience avec des photons polarisés



Expérience avec des photons polarisés

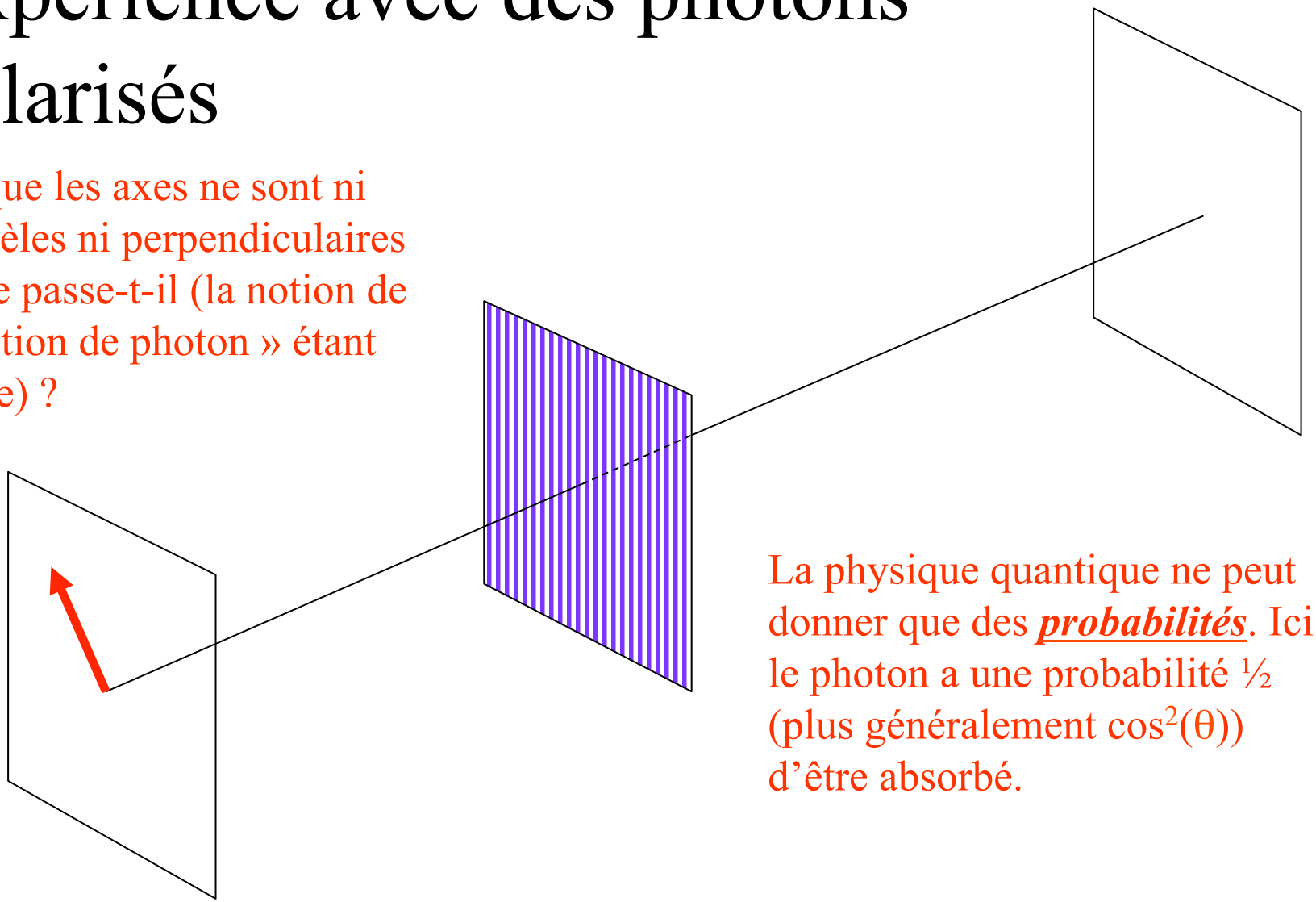


Expérience avec des photons polarisés



Expérience avec des photons polarisés

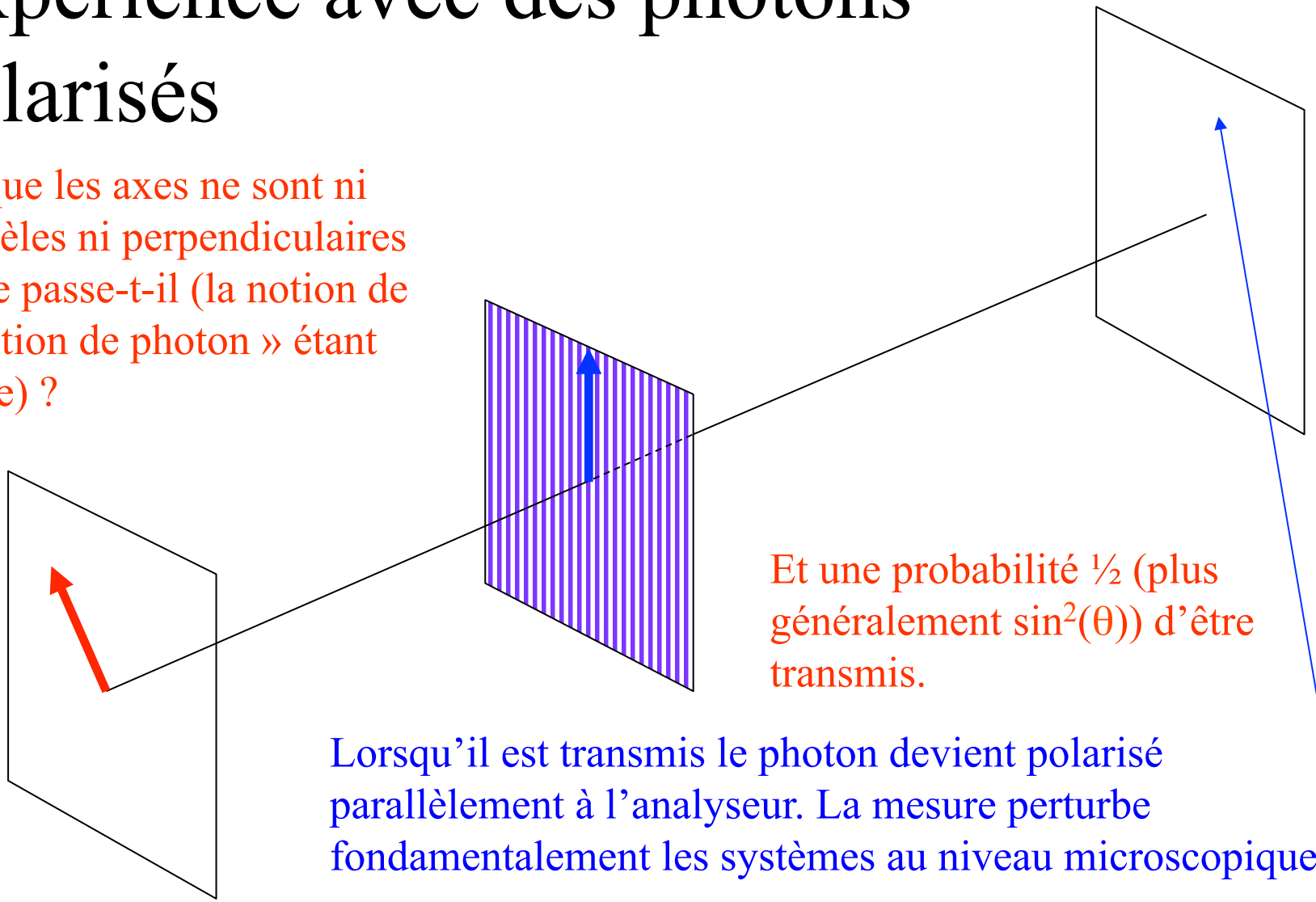
Lorsque les axes ne sont ni parallèles ni perpendiculaires que se passe-t-il (la notion de « fraction de photon » étant exclue) ?



La physique quantique ne peut donner que des ***probabilités***. Ici le photon a une probabilité ½ (plus généralement $\cos^2(\theta)$) d'être absorbé.

Expérience avec des photons polarisés

Lorsque les axes ne sont ni parallèles ni perpendiculaires que se passe-t-il (la notion de « fraction de photon » étant exclue) ?



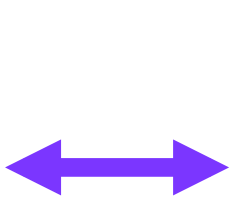
Et une probabilité ½ (plus généralement $\sin^2(\theta)$) d'être transmis.

Lorsqu'il est transmis le photon devient polarisé parallèlement à l'analyseur. La mesure perturbe fondamentalement les systèmes au niveau microscopique.

Physique quantique et représentation du monde : le fondamental aléatoire

- Le fait de ne donner que des probabilités de résultats de mesure n'est pas dû au fait que la théorie quantique est incomplète, c'est au contraire un élément fondamental de la théorie.
- Cela choquait beaucoup Einstein (« Dieu ne joue pas aux dés ») pour qui le fait de s'exprimer en termes de probabilités était la conséquence de notre méconnaissance de « variables cachées » (cf. notre méconnaissance des conditions précises de jet d'un dé).
- Or l'introduction de variables cachées a des conséquences sur certaines prévisions théoriques qui deviennent différentes de celles de la pure théorie quantique (inégalités de Bell 1965).
- L'expérience d'Alain Aspect (1982) donne raison à la théorie quantique (d'autres expériences l'ont confirmé depuis).
- Le dernier mot semble donc pour l'instant à Niels Bohr : « Einstein, cessez de dire à Dieu ce qu'il doit faire… »

Représentation des états quantiques



$|0\rangle$

$|\langle 0|0\rangle|^2=1$

$|\langle 1|0\rangle|^2=0$

$|1\rangle$

$|\langle 0|1\rangle|^2=0$

$|\langle 1|1\rangle|^2=1$

$|0\rangle = (|0\rangle-|1\rangle)/\sqrt{2}$

$|\langle 0|0\rangle|^2=1/2$

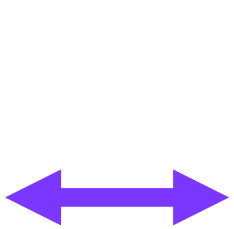
$|\langle 1|0\rangle|^2=1/2$

$|1\rangle = (|0\rangle+|1\rangle)/\sqrt{2}$

$|\langle 0|1\rangle|^2=1/2$

$|\langle 1|1\rangle|^2=1/2$

Représentation des états quantiques



$|0\rangle = (|0\rangle+|1\rangle)/\sqrt{2}$

$|1\rangle = (|1\rangle-|0\rangle)/\sqrt{2}$

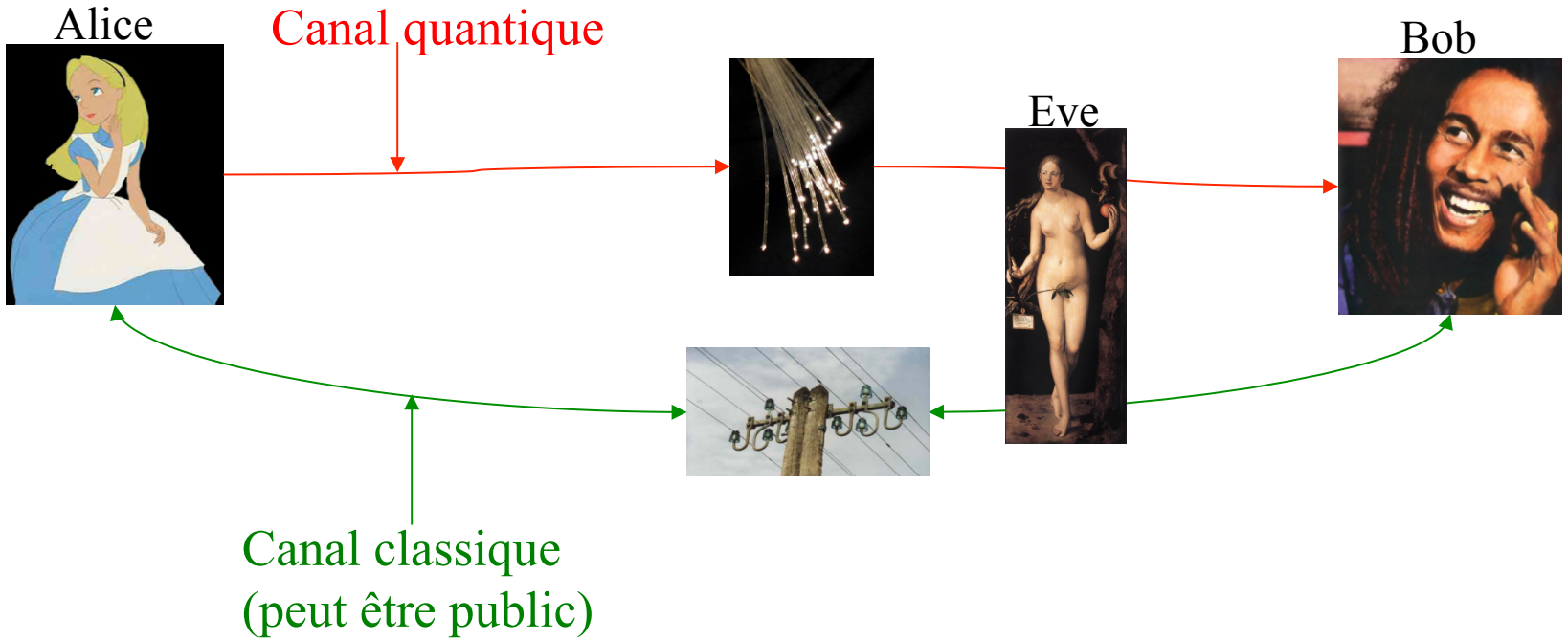
$|0\rangle$

$|1\rangle$

$\lvert\langle 0\rvert 0\rangle\rvert^2 = 1/2$	$\lvert\langle 1\rvert 0\rangle\rvert^2 = 1/2$
$\lvert\langle 0\rvert 1\rangle\rvert^2 = 1/2$	$\lvert\langle 1\rvert 1\rangle\rvert^2 = 1/2$
$\lvert\langle 0\rvert 0\rangle\rvert^2 = 1$	$\lvert\langle 1\rvert 0\rangle\rvert^2 = 0$
$\lvert\langle 0\rvert 1\rangle\rvert^2 = 0$	$\lvert\langle 1\rvert 1\rangle\rvert^2 = 1$

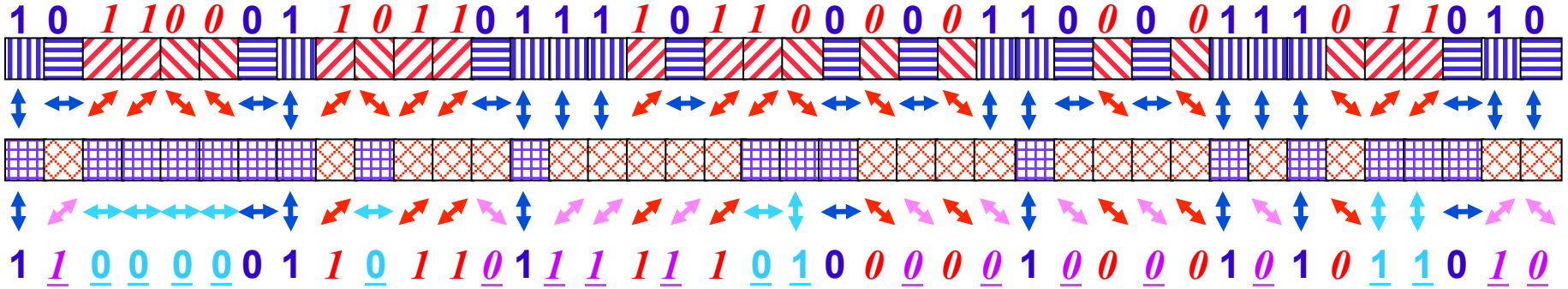
Protocole BB84 (Charles Bennet et Gilles Brassard 1984) : les acteurs

Dans tous les ouvrages de cryptographie, l'émetteur s'appelle Alice (A) et le destinataire légitime Bob (B). Une indiscrète (Eavesdropper) nommée Eve intervient parfois.

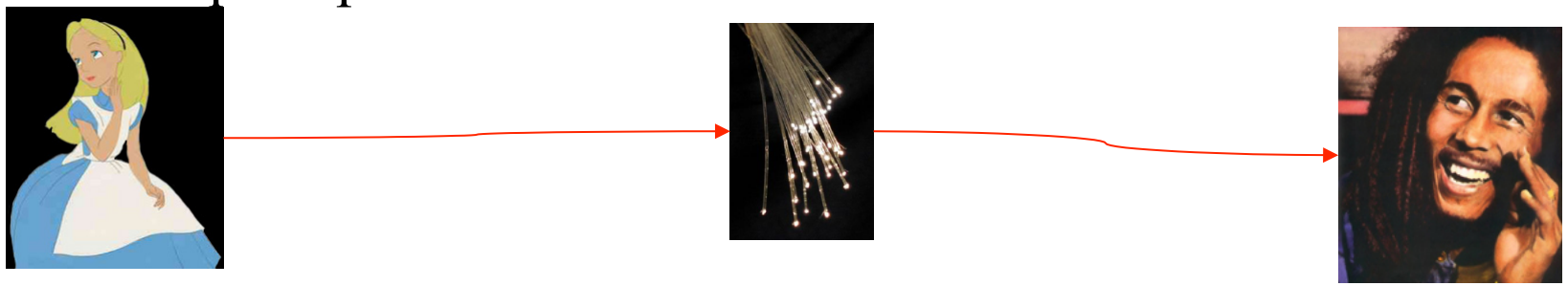


BB 84 : fonctionnement

Sur le canal quantique, Alice émet une suite de bits aléatoires codés aléatoirement sur la base horizontale ou diagonale :

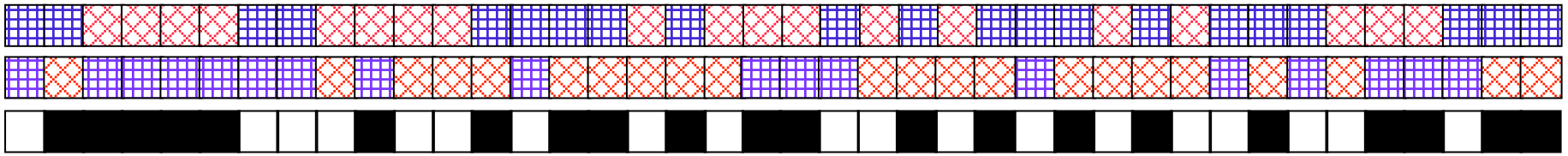


A l'arrivée Bob les mesure en utilisant aléatoirement la base horizontale ou diagonale. Certains (50%) sont donc mal mesurés (le résultat est aléatoire et le photon correspondant modifié) mais il ignore lesquels pour l'instant :

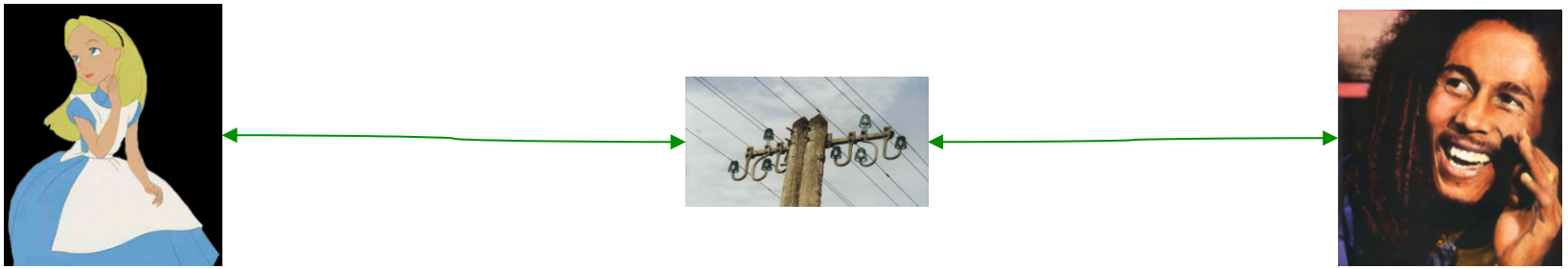


BB 84 : fonctionnement

Sur le canal classique, Alice et Bob échangent alors la *base* (horizontale ou diagonale) dans laquelle ils ont respectivement émis et mesuré chaque photon (mais évidemment pas la valeur 0 ou 1 qu'ils ont émise ou mesurée dans cette base) :

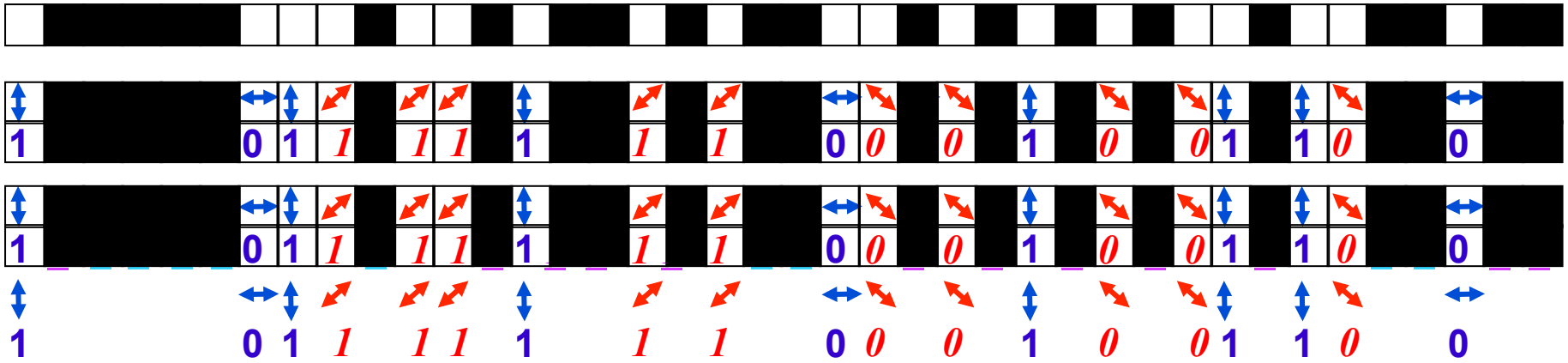


Par comparaison, chacun sait maintenant quels sont les bits qui ont été correctement transmis :



BB 84 : fonctionnement

Il ne reste plus à chacun qu'à appliquer ce masque à sa propre séquence pour trouver les bits (communs) transmis de manière sûre :



Secret partagé pouvant être utilisé comme clé secrète pour un algorithme symétrique :

10111111100010011100

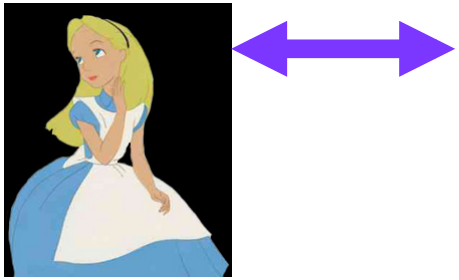
Si Eve espionnait, elle connaît aussi les bits conservés, mais cela lui sera sans utilité… Pire, en espionnant la ligne quantique, elle a trahi sa présence…

La signature d'Eve

- Comme Bob, Eve n'a d'autre choix que de tenter des mesures suivant des bases (horizontale ou diagonale) aléatoires.
- Elle va donc inévitablement perturber la transmission.
- Comme elle perturbe exactement autant les 50% de bits rejetés par Alice et Bob (car constatés en désaccord de base plus tard dans leur dialogue sur la ligne classique) que ceux qui sont conservés, concentrons nous sur ces derniers pour analyser en détails le fonctionnement.

La signature d'Eve

- Lorsque la mesure d'Eve est sur la bonne base (proba=1/2) et laisse passer le photon elle a la bonne valeur *et elle est indétectable* ce cas se produit une fois sur 4…



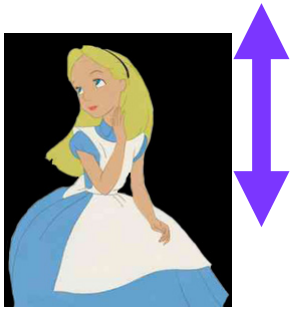
0

0

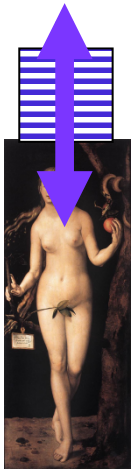
0

La signature d'Eve

- Lorsque la mesure d'Eve est sur la bonne base (proba=1/2) et qu'elle réémet la photon absorbé elle a (et réémet) la bonne valeur et *demeure indétectable* ce cas se produit une fois sur 4…



1



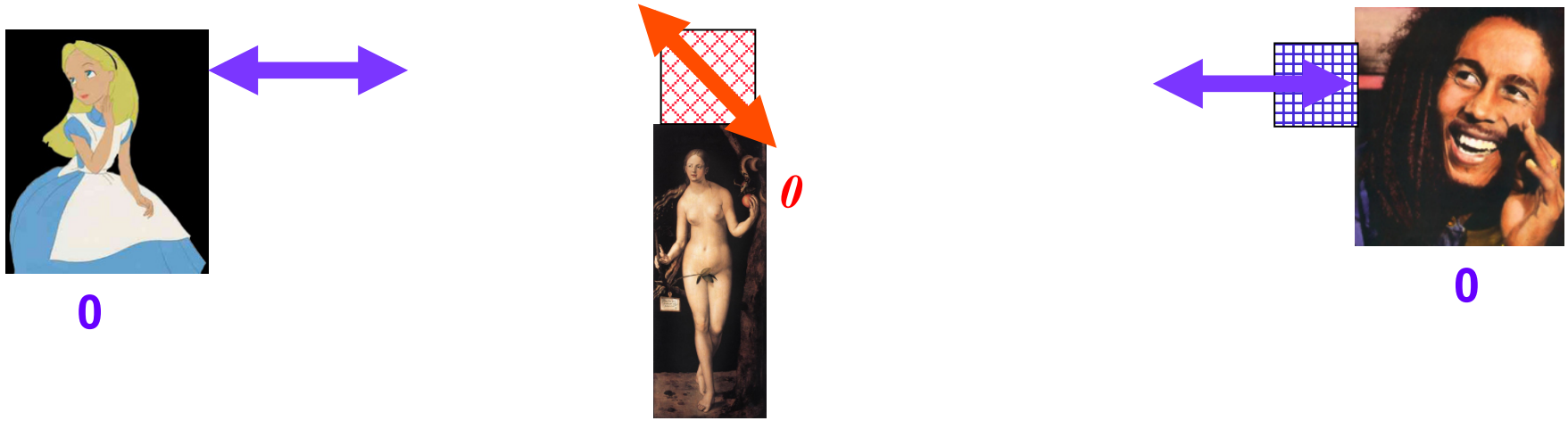
1

1

- Lorsque la mesure d'Eve a absorbé le photon, il lui faut renvoyer un photon correspondant à sa mesure pour espérer demeurer indétectable

La signature d'Eve

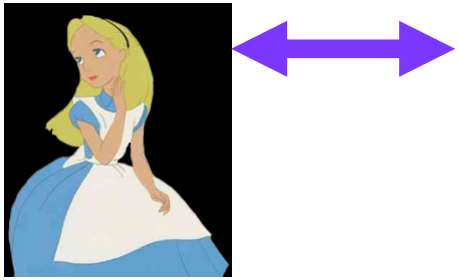
- Bob qui mesure sur la bonne base (par hypothèse) obtient donc une valeur aléatoire qui peut s'avérer correcte par malchance (p=1/2). Dans ce cas (1 sur 4 donc) Eve a une valeur aléatoire *mais demeure indétectée* (les deux perturbations s'étant compensées)…



- Lorsque la mesure d'Eve est sur la mauvaise base (p=1/2) la valeur obtenue est aléatoire (0 ou 1 avec p=1/2 donc correcte ou erronée avec des probabilités 1/2). Elle laisse donc passer (ou réémet) cette valeur aléatoire codée sur la mauvaise base.

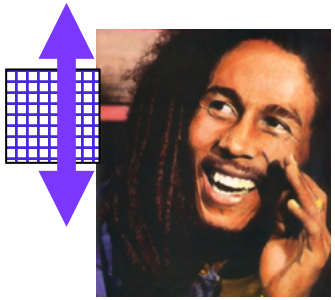
La signature d'Eve

- Mais de manière aussi probable la mesure de Bob peut donner un résultat erroné. Dans ce cas (1 sur 4) Eve a toujours une valeur aléatoire *mais a cette fois signé son forfait…*



0

0



1

La signature d'Eve : Bilan

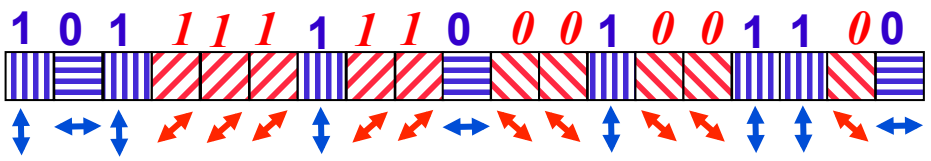
Seuls 50% des bits émis sont retenus (nécessite l'accord des bases Alice et Bob)

Perturbe 50% des bits et obtient un taux de bits corrects de 75% (dont 25% par hasard), proportions valables pour tous les bits donc pour le sous ensemble des bits retenus (indépendance des choix Eve/Bob)

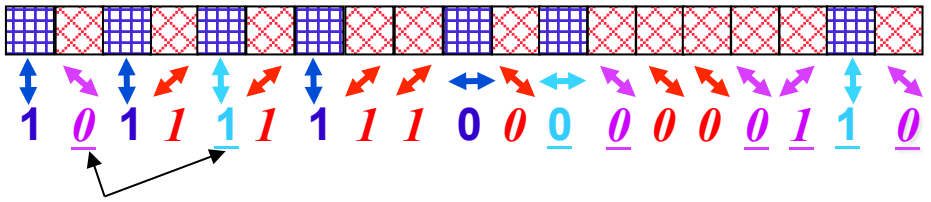
Sur les bits retenus, reçoit 50% de bits perturbés dont la moitié seulement donne une erreur : taux d'erreur constatable : 25% (12,5% du total émis mais ce n'est pas ce taux qui est mesurable)

La signature d'Eve : Exemple

Bits émis par Alice qui seront retenus par Bob

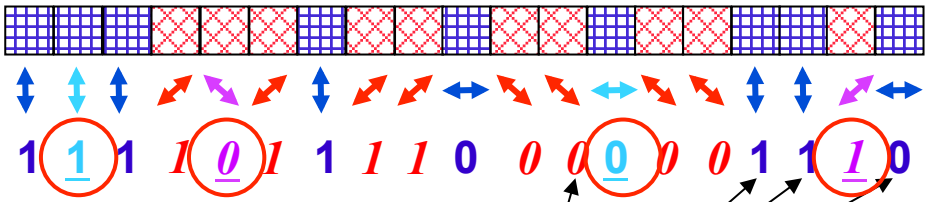


Base d'Ève



Mesure d'Eve : 50% de bits perturbés dont 25% fortuitement valeur correcte

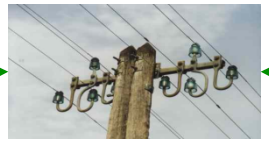
Base Bob=Base Alice pour le sous ensemble choisi



Reste un taux d'erreur de 25% des bits retenus

Mesure de Bob : Compensation des perturbations pour 50% des bits perturbés par Eve.

BB84 : détection d'Eve



Alice et Bob n'ont plus qu'à mesurer le taux d'erreur en sacrifiant N de leurs bits communs et en les comparant par dialogue sur la ligne classique. Ces bits de comparaison sont ensuite jetés.

Si elle écoute cette conversation, Eve ne peut que constater qu'elle est démasquée…

Si la ligne quantique est parfaite (taux d'erreur intrinsèque=0 : intrusion d'Eve détectée dès la première erreur), la probabilité de non détection d'Eve sur N bits est de $(3/4)^N$, qui tend très rapidement vers 0 (32 bits de mesure d'erreur suffisent pour que cette probabilité soit de 0,01%)

Plus généralement si le taux d'erreur intrinsèque de la ligne est suffisamment petit devant 25% la probabilité pour Eve de générer un taux d'erreur inférieur à ce taux intrinsèque (donc non détectable) est faible (calcul facile au cas par cas : le nombre d'erreurs injecté par Eve est régi par une loi binomiale).

BB84 : Considérations pratiques

- Difficultés pratiques pour émettre des impulsions à un seul photon. En pratique impulsions d'intensité très faible => Sensibilité au bruit.
- Si impulsion à plusieurs photons, possibilité d'en dévier quelques uns pour espionner (« Beam splitting Attack » indétectable).
- Sensibilité au bruit => Taux d'erreur intrinsèque qui doit rester petit devant 25%
- Des solutions de cryptographie quantique commencent toutefois à être commercialisées sur environ 100km en fibre optique, environ 20km en propagation libre.

Cryptographie quantique : rendre à César ce qui est à César

- La monnaie quantique de Wiesner (fin des années 1960)

20 photons polarisés sont enfermés dans 20 « pièges à photons ». La séquence base / valeur est unique et correspond au numéro de série (signature infalsifiable)

Faire un faux billet nécessite de mesurer les photons puis de les remettre dans la copie et l'original. Or la probabilité de ne rien perturber ce faisant est de (3/4)^20 (soit 0,3%)

Pas de mise en pratique (faute d'existence de technologie de piège à photon…) mais les idées qui allaient donner BB84 étaient là…

Législation Française en matière de cryptographie

- En vertu de l'article 30-I de la loi 2004-575 du 21 juin 2004, l'utilisation des moyens de cryptologie est libre.

- En revanche, la fourniture, l'importation et l'exportation de ces moyens sont réglementées en France. Ces opérations sont soumises soit au régime de la déclaration, soit au régime de l'autorisation.
- La DCSSI (Direction Centrale de Sécurité des Systèmes d'Information) est chargée d'instruire les demandes d'autorisation des moyens et prestations de cryptologie conformément à la législation.

Pour en savoir plus...

- Bruce SCHNEIER : Cryptographie Appliquée (Vuibert 2001) : La bible dans sa traduction Française
- Douglas STINSON : Cryptographie, Théorie et pratique (Vuibert 2001) : autre très bon livre traduit en Français
- A. MENEZES, P. VAN OORSCHOT, S. VANSTONE : Handbook of applied Cryptography (CRC Press 1996) : un livre impressionnant par la richesse de son contenu, très rigoureux et donc parfois austère.
  Le plus : on peut en télécharger gratuitement de nombreux chapitres sur Internet :
  www.cacr.math.uwaterloo.ca/hac