Cours Sécurité

Cryptographie

Gérard Florin / Stéphane Natkin
- CNAM / Laboratoire CEDRIC -

Plan

Introduction

Chiffres à clés secrètes (symétriques)

Chiffres à clés publiques (dissymétriques)

Fonctions de hachage sécuritaires

Générateurs de nombres aléatoires

Conclusion

Cryptographie

Introduction

Sécurité des communications : Rappel des problèmes.

- **Confidentialité.**
  - Contrôle de l'accès en lecture
- **Intégrité.**
  - Contrôle de l'accès en écriture
- **Authentification.**
  - Vérification de l'identité des sujets.
- **Protection.**
  - Contrôle de droits d'accès pour des services quelconques.
- **Non répudiation.**
  - Impossibilité de nier avoir requis une action par un sujet ou exécuté une action par un objet.

Sécurité des communications et fonctions cryptographiques

- **A) Cryptographie classique** crypto/secret,graphie/écriture : la partie des sciences de la communication qui s'intéresse aux fonctions de transformation **syntaxique** des informations pour en empêcher la lecture par des tiers (**confidentialité**)
  - Les usagers autorisés à accéder à l'information possèdent comme attribut **une clé** qui matérialise leur droit d'accès.
- **B) Cryptographie par extension** : toutes les fonctions utiles pour réaliser des propriétés de sécurité.

  - Fonctions de hachages cryptographiques : absence de **collisions => ** fonctions nécessaires pour assurer la propriété **d'intégrité.**

  - Générateurs de nombres aléatoires : nombres **imprévisibles** (clés, nonces).
- **C) Protocoles de sécurité :** les protocoles de communication qui réalisent des propriétés de sécurité en utilisant comme outil essentiel des fonctions cryptographiques.

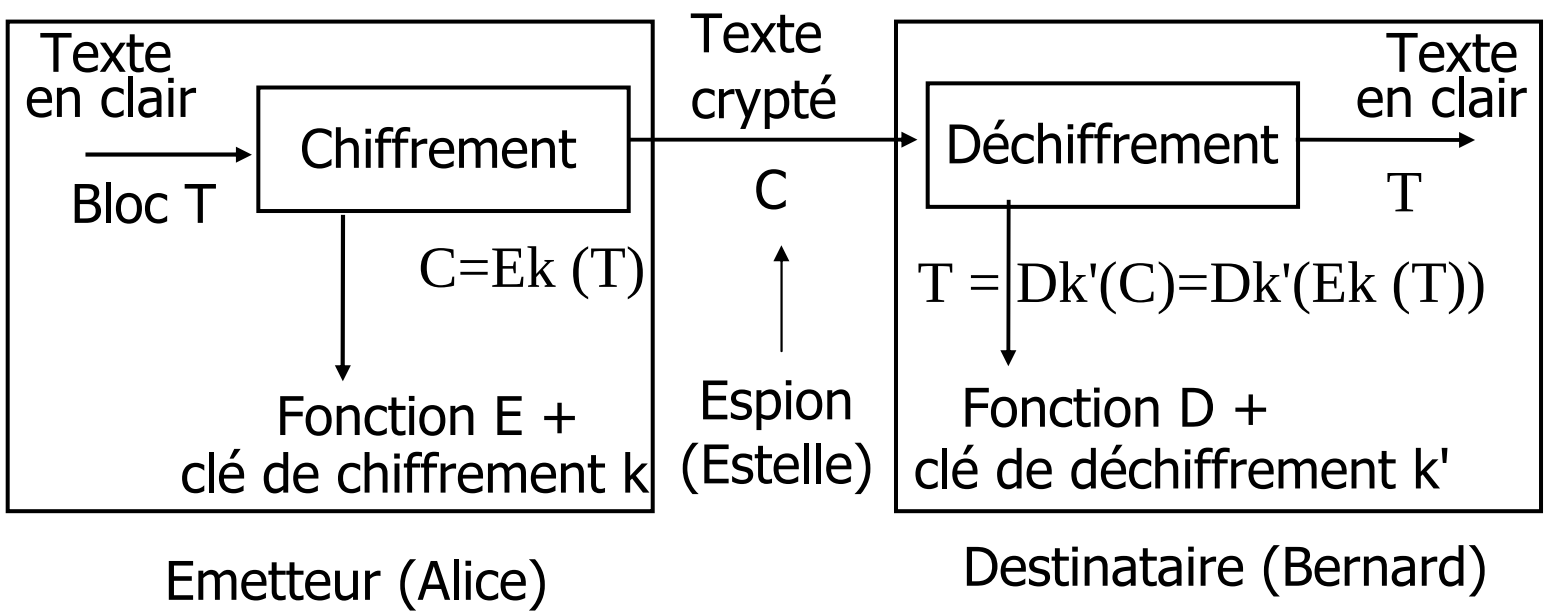
Introduction à la cryptographie :

Introduction aux chiffres

Cryptographie classique : Les trois approches

- A) **Un code** découpe un message en unités syntaxiques (mots, phrases) et remplace ces unités par d'autres selon une table.
  - La clé est la **table de transformation** (le code , '**codebook**')
- B) **La stéganographie** ('écriture couverte') cache un message en clair en le dispersant dans un message anodin
  Historiquement: crâne, masque, tablette cirée, encre invisible, point micro film. Actuellement: watermarking.
  - La clé est la **fonction de dispersion.**
- C) **Un chiffre** ('Cipher') transforme un message en clair en le découpant en unités d'information de taille fixe (par caractères ou par blocs de n bits) sans se préoccuper des unités syntaxiques du message.
  - Chiffre **en continu** ('Stream cipher' bit par bit, plutôtsynchrone), chiffre **par bloc** ('Block Cipher' plus asynchrone: temps pour chiffrer un bloc).
  - La clé est **un paramètre de la fonction de transformation.**

Notion de chiffre



Chiffre (crypto-système) : L'ensemble des deux méthodes de chiffrement Ek (une bijection de {T} dans {C}) et de déchiffrement Dk' inverse de E (bijection {C} dans {T}).
Ces fonctions dépendent de clés k, k'.
Notations : C=Ek(T) ou $C=\{T\}_k^E$; T=Dk'(C) ou $T=\{C\}_k^D$

Compléments relatifs à la définition des chiffres

- **Le chiffrement :** une fonction **bijective** de nature **purement syntaxique** (agissant sur les symboles, les unités lexicales).
- **Qualité essentielle d'un chiffre: Idée de confidentialité**
- En termes de complexité algorithmique: trouver Dk' inverse de Ek est **un problème difficile -> algorithme très long (NP-complet) .**
- Idée du **caractère temporaire de la validité d'un chiffre :** tout chiffre est cassé un jour ou l'autre, il doit résister pendant la durée de validité de l'information qu'il protège (DES , sac à dos).

Vocabulaire

- **Chiffrer, déchiffrer :** appliquer un chiffre avec connaissance de la clé.
- **Décrypter ou casser** un chiffre c'est parvenir au texte en clair sans posséder au départ la clé.
- **Cryptographie** : l'art de définir des chiffres (cryptographe).
- **Cryptanalyse** : l'art de casser des chiffres (cryptanalyste ou casseur de chiffre).
- **Cryptologie** : la réunion de la **cryptographie** et de la **cryptanalyse.**

Chiffres : le point de vue de Shannon* confusion et diffusion

Idée de confusion

La relation entre, d'une part le texte clair et la clef de chiffrement, et d'autre part le texte chiffré doit être aussi difficile que possible à établir.

Le calcul de l'inverse est de complexité élevée.

Principe présenté souvent comme réalisé par les substitutions.

Idée de diffusion

Une modification même mineure du texte en clair doit se traduire par une modification très importante du texte chiffré.

Chaque symbole en clair doit être 'diffusé' dans tout le texte chiffré (chaque symbole chiffré doit dépendre de beaucoup de symboles en clair).

Principe présenté souvent comme réalisé par les permutations.

* Claude Shannon 'Communication Theory Of Secrecy Systems' Bell Systems journal 1949

Chiffres: cryptanalyse
Quatre niveaux de difficultés d'attaque

A) Attaque à textes **chiffrés connus C** (ciphertext-only attack)

- L'attaquant dispose seulement de textes chiffrés C.

B) Attaque à textes **clair connus M,C** (known-plaintext attack)

- L'attaquant dispose de textes en clair et de leur chiffre.

C) Attaque à textes **clair choisis M->C** (chosen-plaintext attack)

- En plus de B) l'attaquant peut faire crypter ce qu'il veut par la méthode de chiffrement et voir ce qu'elle produit.

D) Attaque à textes **chiffrés choisis C->M** (adaptative-plaintext attack)

- L'attaquant ne connaît pas la clé mais il peut faire déchiffrer ce qu'il veut par la méthode de déchiffrement et voir le clair (il a un accès à l'algorithme de déchiffrement mais ne peut le désassembler pour avoir la clé).

Chiffres: cryptanalyse
Différentes étapes d'une attaque (1)

A) Détermination des méthodes de chiffrement et de déchiffrement utilisés : *taille des blocs, délimitation des blocs, méthodes E et D.*

B) Réduction de l'espace des clés

- Détermination de clés **impossibles ou peu probables**.
- Techniques de **cryptanalyse linéaire**: contraintes sur les clés déduites de relations linéaires entre texte en clair et texte chiffré.
- Cryptanalyse **différentielle**: utilisation de textes en clairs peu différents, observation des différences sur les textes chiffrés => contraintes (attaque à texte en clair choisi).
- Utilisation de **propriétés de la méthode de génération des clés** => prévision des clés utilisées (loi de génération, dictionnaire)

Chiffres: cryptanalyse
Différentes étapes d'une attaque

C) Essai force brute des clés encore possibles :
Problème détermination d'un test d'arrêt simple => utilisation de règles syntaxiques simples.

- **Recherche d'invariants syntaxiques du langage utilisé**: règles de construction des expressions, apparition de mots clés.
- **Utilisation des redondances:** fréquences d'apparition des lettres, des mots …

D) Utilisation de schémas d'interprétation complexes (sémantiques) => vers une interprétation sémantique du message (humaine).

Notion de complexité algorithmique : la classe P

- **Un cadre formel de hiérarchisation des problèmes (de la difficulté de résolution de problèmes)**
  - **Des problèmes types** : Problèmes de décision binaire . Autres problèmes on les ramène à une décision.
  - **Exemples :** la reconnaissance d'un langage, la satisfaction d'une expression booléenne.
- **Basé sur un modèle abstrait de machine**: la machine de Turing déterministe (entrées, sorties, ruban, unité de contrôle).
- **Classe des problèmes à temps polynomial sur machine de Turing déterministe** : **Classe P**
  - Pour toute entrée x il existe un polynôme p(|x|) **définissant le nombre d'étapes de la machine en fonction de la taille d'une instance d'entrée x => importance du degré du polynôme.**
  - Permettant de **décider de la solution: Exemple** s'arrêter en un état final d'acceptation ou de rejet (par exemple d'un langage).
- **Classe Exptime** : problèmes décidables en temps exponentiel par rapport à la taille de l'instance sur machine déterministe.

Notion de complexité algorithmique : la classe NP

- **Machine de Turing non déterministe :** Autre modèle abstrait de machine.
  - Chaque état peut donner lieu à **plusieurs transitions**.
  - On créé autant de branches **différentes de résolution** que d'évolutions possibles dans les transitions non déterministes.
- **Classe des problèmes à temps polynomial sur machine non déterministe** : **Classe NP** ('**N**on deterministic **P**olynomial time') et classe **co-NP complémentaire de NP** (la négation des problèmes de NP).
  - Pour toute entrée x il existe un polynôme p(x) **définissant le nombre d'étapes de la machine de Turing non déterministe.**
  - Permettant de **décider de la solution.**
- **Classe des problèmes NP-complets** (les plus difficiles de NP): tous les problèmes appartenant à NP sont réductibles en temps polynomial à un problème de la classe NP-complet.

Notion de complexité algorithmique : la complexité en espace

- **Dualité :** entre le temps de calcul et l'espace mémoire utilisé.
- **Classe PSPACE** : problèmes que l'on peut décider par un algorithme sur machine de Turing déterministe utilisant un espace polynomial par rapport à la taille de son instance.
- **Classe NPSPACE** : que l'on peut décider par un algorithme sur machine de Turing non déterministe utilisant un espace polynomial par rapport à la taille de son instance.

Chiffres : Critères de construction d'un bon chiffre (1)

- A) Rappel: trouver Dk' inverse de Ek est **un problème difficile (NP-complet).**

- B) **Principe de Kerckhoffs ou Kerchoff (1883):**
  La difficulté ne doit pas dépendre du secret des algorithmes mais du secret des clés.

- C) Un chiffre doit être **stable** (on ne peut le changer que très rarement).

- D) Réalisation **simple et rapide** du chiffrement et du déchiffrement (pour atteindre des débits élevés).

Chiffres : Critères de construction d'un bon chiffre (2)

- E) Une clé doit être choisie de **façon aléatoire** dans un espace de clés qui doit être très **grand**.

  => Une clé doit être **imprévisible**.

- F) Un chiffre dépend de clés qui doivent être **changées fréquemment et donc facilement => Éviter un encombrement important des clés** (pour le transport et le confort des utilisateurs).

- G) Les messages en clair doivent comporter **le moins de redondances possibles:**

  Soit par **compression** préalable

  Soit par **extension** en ajoutant des informations constituant du bruit.

Les chiffres symétriques (à clé privée)

Les méthodes de chiffrement Ek et de déchiffrement Dk' sont liées du point de vue du secret des clés k et k'.

- La connaissance de k entraîne celle de k' et réciproquement.
  - Pratiquement k = k'.
- La clé étant partagée entre l'émetteur et le destinataire, le secret de la clé doit être très bien gardé.
- Il faut pour N communicants potentiels N(N-1)/2 clés secrètes.

Les chiffres dissymétriques (à clé publique)

- Idée publiée pour la première fois par **Diffie et Hellman.**
- Les méthodes Ek et Dk’ sont telles que **E et D étant connues il est très difficile de déduire k'** de la connaissance de k.
- On peut **rendre k publique** (**notion de clé publique**) connue de tous dans un annuaire.
- **Tout le monde peut chiffrer**: C = Ek (M)
- Par contre la clé k' (la clé privée) **reste secrète.**
- Seul le **destinataire peut déchiffrer** M = Dk' (C)
- Propriété complémentaire (très utile): la **commutativité des méthodes de chiffrement et de déchiffrement.**

$$Dk'\,(\,Ek\,(\,M\,)) = Ek\,(\,Dk'\,(\,M\,) = M$$

Les chiffres symétriques (à clés secrètes)

Concepts classiques

Chiffres par blocs : DES , IDEA , AES

Chiffres en continu : LFSR , RC4

Chiffres symétriques par blocs (à clés secrètes)

Concepts classiques

Transposition

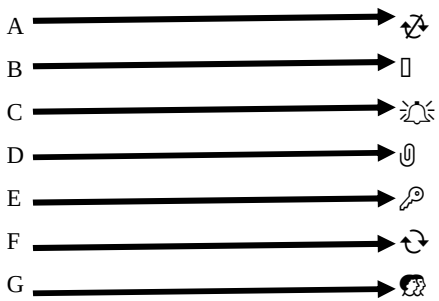
Substitution

Chiffres à clés secrètes classiques: substituer ou transposer

Solutions essentielles en cryptographie (utilisées depuis le début)

- **Substitution** : un symbole du texte en clair est remplacé par un autre symbole dans le texte chiffré.

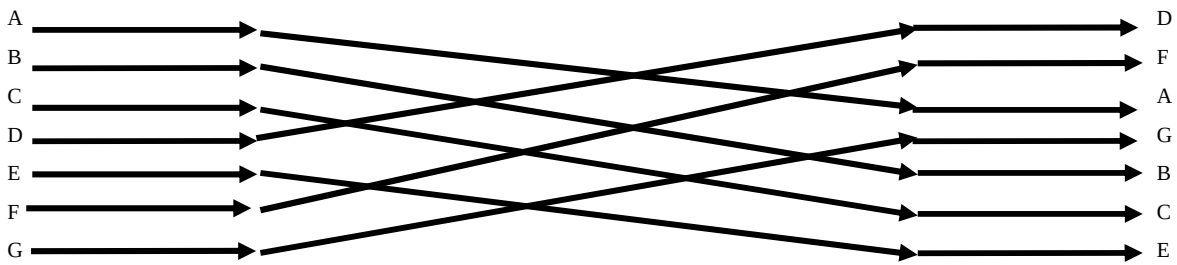
ABCDEFG -> 



Contribue à la **confusion.**

- **Transposition** : les symboles en clair sont réordonnés

ABCDEFG -> DFAGBCE



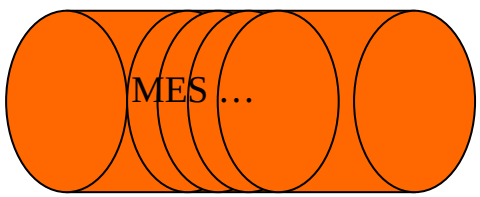
Contribue à la **diffusion.**

Chiffres à clés secrètes
Concepts classiques

Les transpositions

Chiffres à transpositions

- **Transposer = opérer une permutation** des symboles d'un message en clair : remplacer chaque symbole par un autre symbole du message.
- **Version très ancienne:**
  - Un cylindre sur lequel une ceinture est enroulée.
  - On écrit le message sur la ceinture selon une génératrice du cylindre.
  - Lorsqu'on déroule la ceinture le message en clair est rendu obscur.



- **Utilisation moderne:** transposition basée sur un tableau => qui définit l'ordre des symboles du texte en clair dans le texte chiffré.

Exemple élémentaire de chiffre à transpositions

P:16	E:5	R:18	M:13	U:21	T:20	A:1	I:8	O:15	N:14	S:19
T	R	A	N	S	P	O	S	I	T	I
O	N		A	V	E	C		U	N	
T	A	B	L	E	A	U		N	X	M

- On écrit le texte en clair en ligne dans un tableau.
- On se donne une clé pour déterminer l'ordre d'utilisation des colonnes dans le texte chiffré.
- **Exemple de clé** : la chaîne de caractères **'Permutations'** avec ses chiffres 16;5;18;13;21;20;1;8;15;14;19 (écrite la première ligne).
- On réordonne la clé 1;5;8;13;14;15;16;18;19;20 (ordre total la lettre répétée T est ôtée)
- On transmet en colonnes dans l'ordre de la clé =>

Message chiffré: 'OCURNAS NALTNXIUNTOTA BI MPEA SVE'

- **Cryptanalyse statistique très aisée** pour ce chiffre utilisé seul.
- **Utilisation dans les chiffres modernes** (DES, AES) : une étape de transposition réalisée par une table contribue à la diffusion.

Chiffres à clés secrètes
Concepts classiques

Les substitutions

1 - Substitution simples (mono alphabétiques).

2 - Substitutions poly alphabétiques.

3 - Substitution à masque jetable

1) Chiffres à substitutions de base (mono alphabétiques)

- **Famille des substitutions mono alphabétiques**: remplacer chaque lettre (chaque symbole) par une unique autre lettre (chaque symbole).
- **Idée de chiffrement :** on associe à chaque lettre un chiffre

  A->0, B->1, … , Z->25
- **Chiffres mono alphabétiques 'additifs': chiffre de César**

  Ek (x) = x + k (mod 26) (26 chiffres différents).
- **Chiffres mono alphabétiques 'multiplicatifs':**

  Ek (x) = k.x (mod 26) (avec k premier avec 26) (12 chiffres différents k=1,3,5,7,9,11,15,17,19,21,23,25)
- **Chiffres mono alphabétiques 'affines':**

  Ek1,k2 (x) = k1 (x + k2) (mod 26) (312 chiffres différents)

Chiffres mono alphabétiques quelconques : substitutions générales

- **Dénombrement:** 26! substitutions différentes des 26 lettres de l'alphabet par des lettres du même alphabet.
- **Chiffre de substitution mono alphabétique général défini par une permutation:** on choisit une chaîne de 26 lettres comportant chaque lettre une et une seule fois (permutation)

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
m	z	a	e	c	p	x	d	g	b	o	u	y	h	w	s	i	l	v	r	j	t	f	n	q	k

- **Chiffre à mot clé (solution mnémotechnique pour retenir la clé):** on choisit une lettre et un mot clé (un secret)

Exemple : lettre i et mot clé 'mnémotechnique' -> mneotchiqu

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
p	r	s	v	w	x	y	z	**m**	**n**	**e**	**o**	**t**	**c**	**h**	**i**	**q**	**u**	a	b	d	f	g	j	k	l

Chiffres mono alphabétiques: méthode statistique de cryptanalyse

- Détermination des **fréquences d'apparition des lettres.**
- Comparaison avec les **fréquences caractéristiques** de la langue utilisée.
- Chiffre mono alphabétique: la **fréquence d'une lettre** en clair se retrouve sur **une autre lettre** en chiffre.

Fréquences d'apparition en français

- A:6,16 B:0,40 C:5,35 D:3,86 E:18,61 F:2,24 G:1,79 H:1,48 I:6,35 J:0,04 K:0,13 L:5,26 M:1,97 N:6,02 O:5,12 P:2,92 Q:0,62 R:5,35 S:6,96 T:7,41 U:5,03 V:1,03 W:0,35 X:0,36 Y:1,39 Z:0,04
- Groupe e > 12% ; ainst 6-9% ; cdloru 3-6% ; fghmpvy 1-3% ; jkwxzb <1%
- Mais aussi fréquence des **digrammes** (et), **trigrammes** (ant).

- Le problème est de disposer de **suffisamment de texte** pour que la statistique soit significative.
- Une analyse statistique d'un texte suffisamment long permet de **casser facilement un chiffre mono alphabétique**.

2) Chiffres à substitutions multiples (poly-alphabétiques)

- **Utiliser plusieurs symboles différents pour un symbole donné** (avoirs prévu plusieurs substitutions possibles).
- **Substitutions à représentation multiple: substitutions homophoniques** (Mantoue 1401)
  - Une lettre (un symbole) est chiffré par plusieurs symboles différents.
  - Tous les symboles utilisés comme chiffre doivent être différents.
  - Pour chaque symbole, on devrait utiliser un nombre de représentations proportionnel à la fréquence d'apparition du symbole en clair (diffusion).
  - Ex : A chiffré par 4, 13, 22, 67 ; B chiffré par 21, 23 etc …
- **Substitutions poly-alphabétiques** (Léon Battista Alberti 1468)
  - Utilisation de plusieurs chiffres à substitution simples en même temps. Le chiffre simple utilisé dépend de la position du symbole (chiffre de Blaise de Vigenère 1583)
- **Substitutions simples par polygrammes:** suite de n lettres
  - Ex digrammes AB ->RT AC ->VD (chiffre Playfair 1854) (chiffre de Hill)

Substitutions poly alphabétiques
Chiffre de Vigenère

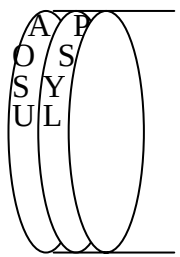
- Les 26 chiffres de César représentés en carré forment le **carré de Vigenère.**

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

- Les 26 chiffres associés aux 26 **décalages possibles sont représentés par une lettre.**
- On choisit comme clé **une suite de lettres**: un mot de passe ou une phrase ou un passage de livre
- Cette **clé répétée indéfiniment** vis à vis de chaque lettre d'un texte en clair à chiffrer **sert à déterminer le chiffre de César à utiliser.**
  - Ex: La clé BAC , le texte en clair VIGENERE, le chiffre XJJGOHTF
    BACBACBA
- La **longueur de la clé** et ses **propriétés statistiques** définissent la qualité du chiffre de Vigenère.

Substitutions poly alphabétiques Machines à rotors : a) Jefferson

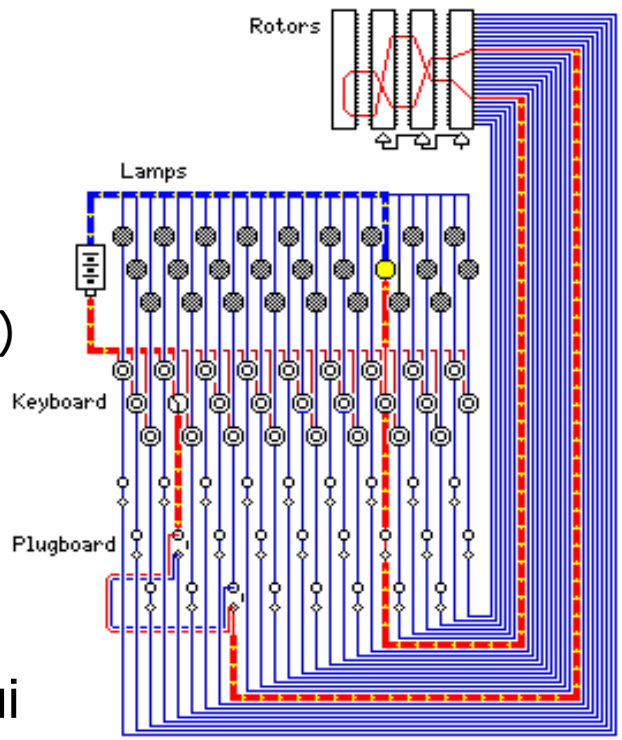
- **Machine de Jefferson** (1790):
  - Les substitutions simples utilisées sont **écrites sur 36 disques mobiles indépendants** (roues crantées).
  - Un message est **découpé en blocs de 36 lettres.**
  - On **aligne le texte en clair sur une rangée.**
  - On **transmet une autre rangée**
    - Une rangée **au hasard**.
    - Ou une rangée **fixe prédéfinie par son rang**.
    - Ou selon **une règle de changement des rangées** utilisées.
  - **Une implantation facilitée d'un chiffre poly alphabétique.**



- 36 rotors mobiles.
- Une permutation sur chaque rotor

Substitutions poly alphabétiques Machines à rotors : b) Enigma

- **Machine Enigma** (1919 - 1945)



- **Les entrées/sorties**
  - **Clavier** (keyboard) : le texte en clair ;
  - **Lampes** (lamps) : le message chiffré ;
  - **Tableau de fiches** (plugboard) : la clé (6 fiches qui échangent 12 lettres soient 100 millions de clés)
- **Trois rotors**: 26 contacts sur chaque face des disques réalisant une substitution au moyen de fils. Le quatrième rotor retourne le signal pour trois autres séries de substitutions.
- **Le rotor d'entrée** tourne d'une position à chaque lettre. Après 26 lettres le second tourne lui aussi … (26x26x26 = 17576 positions des rotors).

3) Substitution à masque jetable 'One Time Pad' : Principe

- **Rendre le texte crypté non analysable, sans relation avec le texte en clair** : confusion et diffusion totale.
- **Solution des clés jetables:** générer une suite de clés de blocs K_M qui est **une suite de nombres aléatoires**
  - **Définition de l'aléatoire :** si les clés K_M sont sur n bits l'entropie de la source génératrice de clés est n bits.
  - **Utilisation d'un phénomène physique** (ex : radio activité).
- **Chaque clé K_M ne doit servir qu'une fois (pour un seul message):** problème d'encombrement des clés.
- **Pour chiffrer un message faire le ou exclusif** du bloc et d'une clé : $C = M \oplus K_M$
- **Pour déchiffrer un message** l'opération est la même (avec la même clé): $M = C \oplus K_M$ $(M = M \oplus K_M \oplus K_M)$

Chiffres à masque jetable : Compléments

- **Utilisation**: téléphone rouge, espionnage, Che Guevara.

- **Les inventeurs :** ce chiffre est attribué à Robert Vernam.
  - Robert Vernam (1917) **Idée du ou exclusif** avec la clé,
  - Joseph Mauborgne (1920) **Idée d'une clé aléatoire**,
  - Claude Shannon (1948) **Preuve du caractère incassable.**

- **Décryptage** : Message chiffré C et essai de toutes les clés K_x
  - Ici toutes les clés sont équiprobables donc C a pu être construit à partir de n'importe quel message en clair)
  - **On décrypte tous les messages possibles** (tous les messages de syntaxe correcte): $\forall$ Mess $\exists$ K_x : Mess = C $\oplus$ K_x
  - **Sans moyen de décider qu'un message décrypté** est le message émis plutôt qu'un autre => Diffusion, confusion totale

Chiffres à masque jetable : Théorie de l'information

- **Un chiffre est incassable** si Prob (M/C) = Prob (M) c'est à dire M et C sont indépendants.

  Prob (M/C) = Prob (M et C) / Prob (C)

  Prob (M et C) = Prob (M) Prob (C).

- **L'information apportée par C sur M à un intrus est nulle:**

  $I = - \log_2 (\text{Prob } (M/C) / \text{Prob } (M)) = 0$

- **Le destinataire autorisé accède à l'information car il utilise la connaissance de la clé** (entropie égale à l'entropie des messages chiffrables).

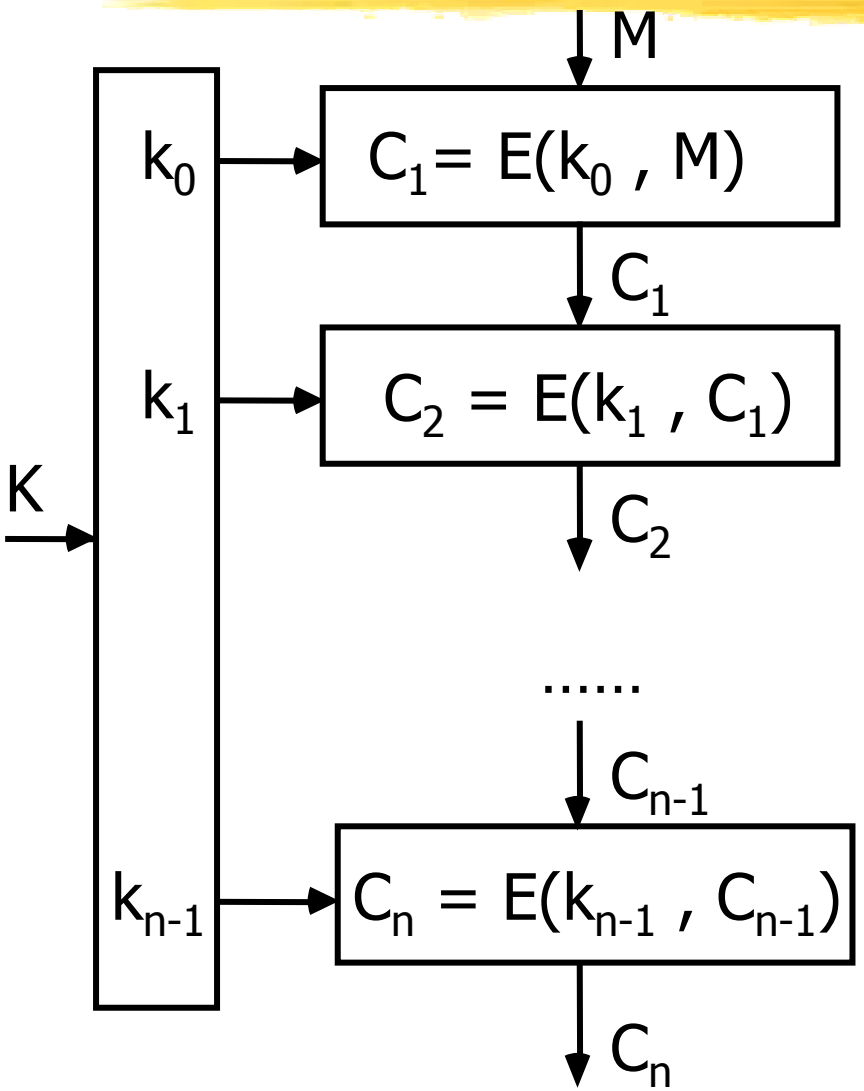
Chiffres par blocs à clés secrètes

DES ‘Data Encryption Standard’

DES ‘Data Encryption Standard’ : Introduction

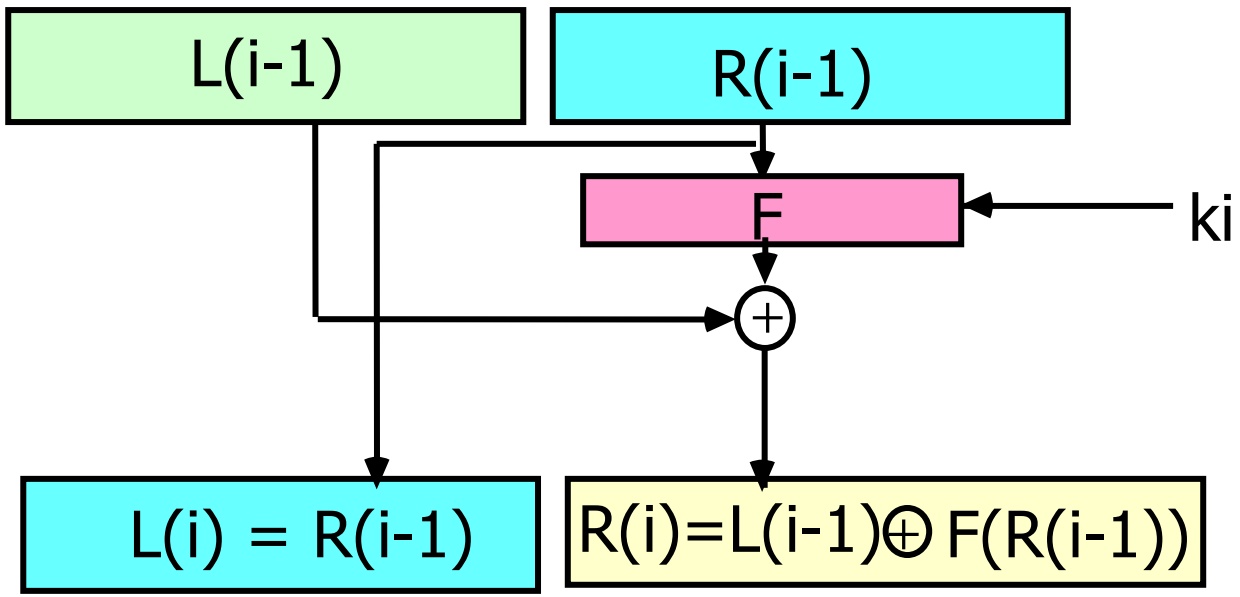
- **Années 1960:** des **circuits intégrés** ou des **ordinateurs** permettent de réaliser des fonctions complexes: en particulier des **substitutions** et des **transpositions.**
- **Application** de ces techniques dans un **produit de chiffres** utilisable en chiffrement et en déchiffrement (**Feistel 1963**).
- **Mise au point par IBM** à partir de 1968 d'une méthode de chiffrement **(‘Lucifer’)** basée sur **16 étages** (16 rondes).
- **Appel d'offre** NBS ("National Bureau of Standards") (1973) pour la mise au point d'un système de cryptographie.
- **Adoption définitive et normalisation** de la proposition **IBM (1976):** nom donné au départ **DES.**
- Normalisation FIPS 46-2 et ANSI X3.92 (1993) sous le nom de **DEA ("Data Encryption Algorithm").**

DES: Notion de chiffre itéré (produit de chiffres)



- Définition d'un **chiffre de base** E dépendant d'une clé k.
- **Répétition** de ce chiffre n fois avec des clés différentes.
- Les **clés de chaque étape (ou ronde)** sont le plus souvent dérivées par transformation d'une **clé maîtresse** K.
- Le nombre de rondes définit la **sécurité** de la méthode (mais aussi sa **rapidité**)

DES: Notion de réseau de Feistel Définition d'un ronde (d'une étape)



- Transformation sur un bloc **séparé en deux moitiés L et R**:

(L(i-1),R(i-1)) ->(L(i),R(i)) ; L(i)=R(i-1) et R(i)= L(i-1) ⊕ F(R(i-1))

- **Le chiffre n'agit que sur la partie droite** (en fonction d'une clé ki).
- C'est **l'itération de cette structure** qui **chiffre successivement les deux moitiés à un niveau satisfaisant.**

DES : Intérêt des réseaux de Feistel

- **Un réseau de Feistel définit une involution** : bijection qui est sa propre inverse.
- **Règle a) Le circuit qui chiffre est identique à celui qui déchiffre** à condition d'inverser l'ordre des clés présentées à chaque ronde.
  - **Calcul de chiffrement** : en entrée [a , b] en sortie [b , a ⊕ F(b)]

    [L(i-1)=a , R(i-1)=b] -> [L(i) = b , R(i) = a ⊕ F(b)]
- **Règle b)** Le dernier étage prépare le déchiffrement en permutant la partie droite et gauche.
  - En sortie du dernier étage [a ⊕ F(b) , b]: résultat final.
- **Calcul de déchiffrement** : En entrée du premier étage [a ⊕ F(b) , b]

Donc en sortie [b , a ⊕ F(b) ⊕ F(b) = a] soit un résultat inversé [b , a].

- Donc en déchiffrement on **inverse la droite et la gauche** sur tous les étages mais **à la fin on inverse une fois de plus** comme au chiffrement donc on obtient finalement le bon texte en clair.
- **Autres chiffres à réseaux de Feistel** : Gost, CAST, Blowfish ….

DES : Caractéristiques générales

1) Utilisation d'une clé sur 56 bits

- Clé sur 64 bits mais 8 octets avec parité donc seulement 56 bits
- Initialement conçu à 128 bits le DES a été ramené à 56 bits

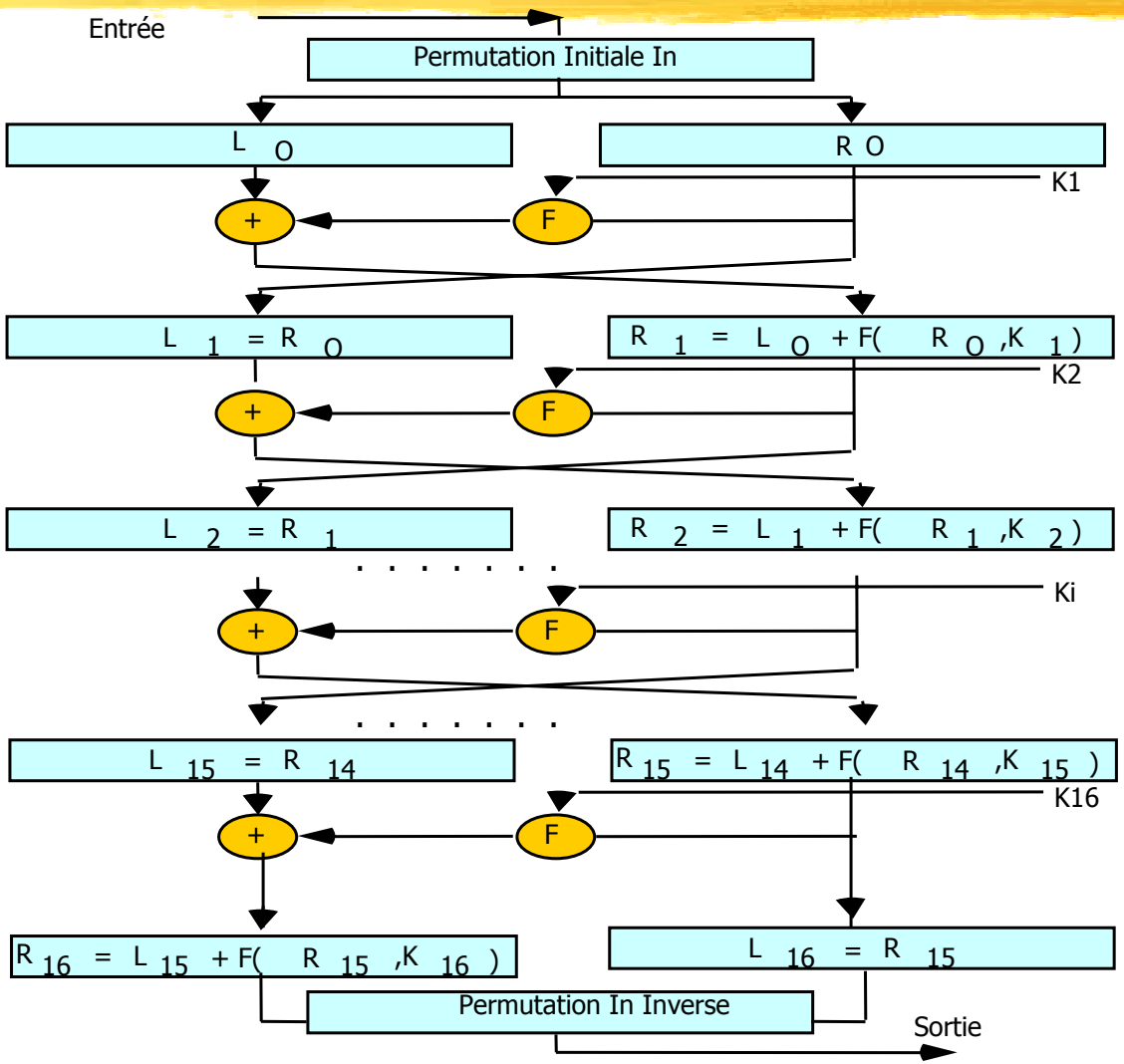
2) Rondes de chiffrement (18)

- Transpositions et/ou substitutions sur des blocs de 2 x 32 bits.
- Un étage amont et un étage aval sont des transpositions fixes.
- Seize étages intermédiaires sont basés sur des substitutions qui dépendent de la clé de façon complexe (rondes de Feistel).

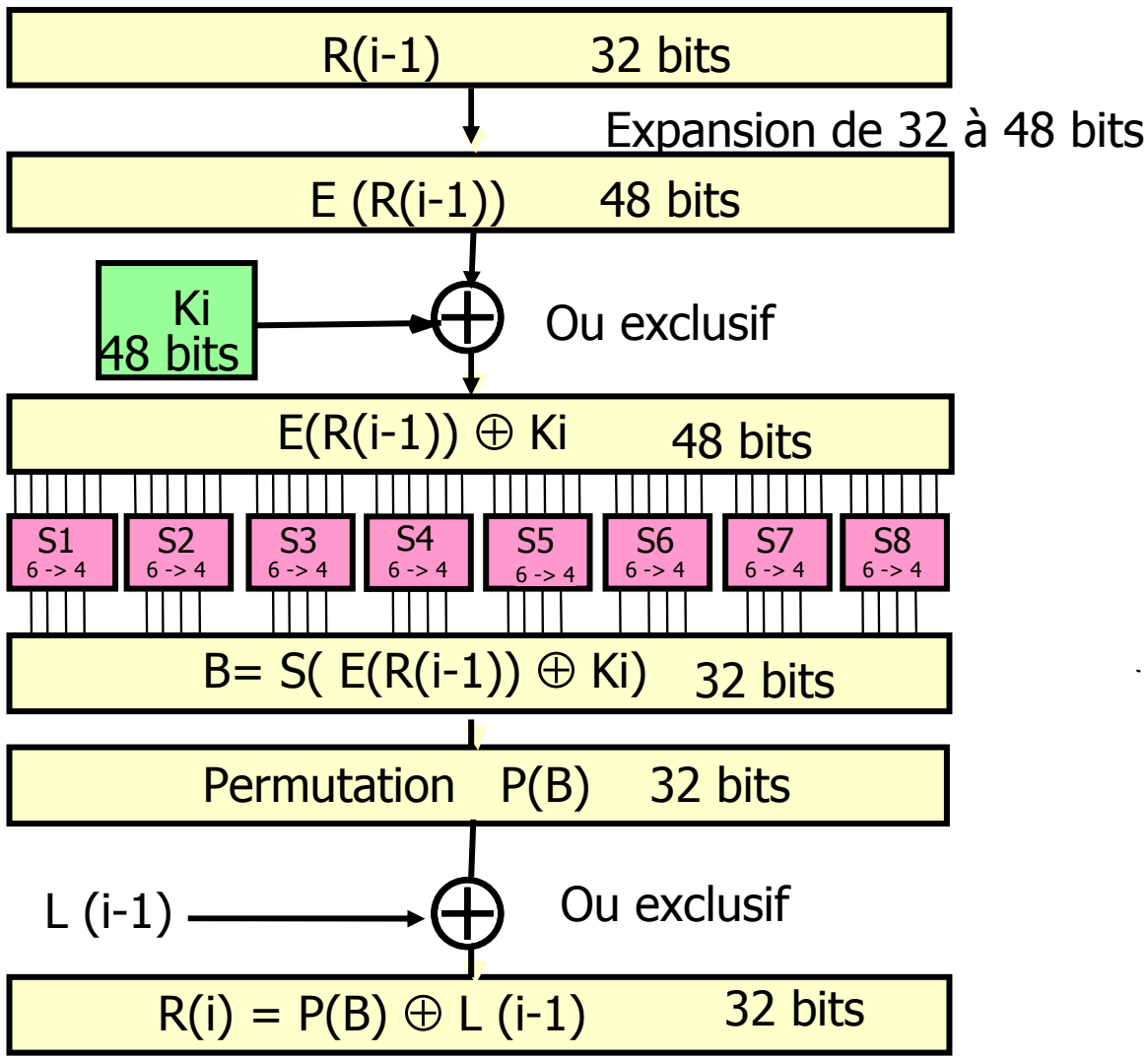
3) Deux modes de fonctionnement

- Mode cryptage par bloc de 64 bits.
- Mode cryptage à la volée "stream" : octet par octet avec des registres à décalage et recyclage.

DES : L'architecture générale



DES : Détail des fonctions de chiffrement de ronde



DES : Les boites de transposition (P-Box ‘ Permutation Box ’)

- **La permutation initiale du DES (64 bits)**

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	25	27	19	11	3
61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

Le bit 1 remplace le 58, le bit 2 remplace le 50 …

- **La permutation finale est l’inverse.**

- **Solution matérielle** : facile à réaliser par câblage.
- **Solution logicielle** : par consultation de tables.

DES : La fonction d'expansion de 32 à 48 bits

- **Solution basique** : certains bits sont recopiés plusieurs fois.
- **Table d'expansion** : 32 -> 48 bits, selon le parcours en lignes 32,1,2,3,4,5, 4,5,6,7,8,9 8,9…
- **Expansion inversible** : bijection entre les configurations de 32 bits et de 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

DES : Les boites de substitution (S-Box ‘ Substitution Box ’)

- **A chaque étage on réalise une substitution**
  - **Contribue à la confusion par une fonction non linéaire.**
- **Exemple:** Table S-1 du DES on substitue à une valeur sur 6 bits une valeur sur 4 bits. Les deux bits faible et fort sélectionnent la ligne, les 4 bits intermédiaires la colonne (autant de tables différentes que de rondes)

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

- **Solution matérielle :** par câblage de la correspondance.
- **Solution logicielle :** par consultation de la table

DES : La permutation de ronde

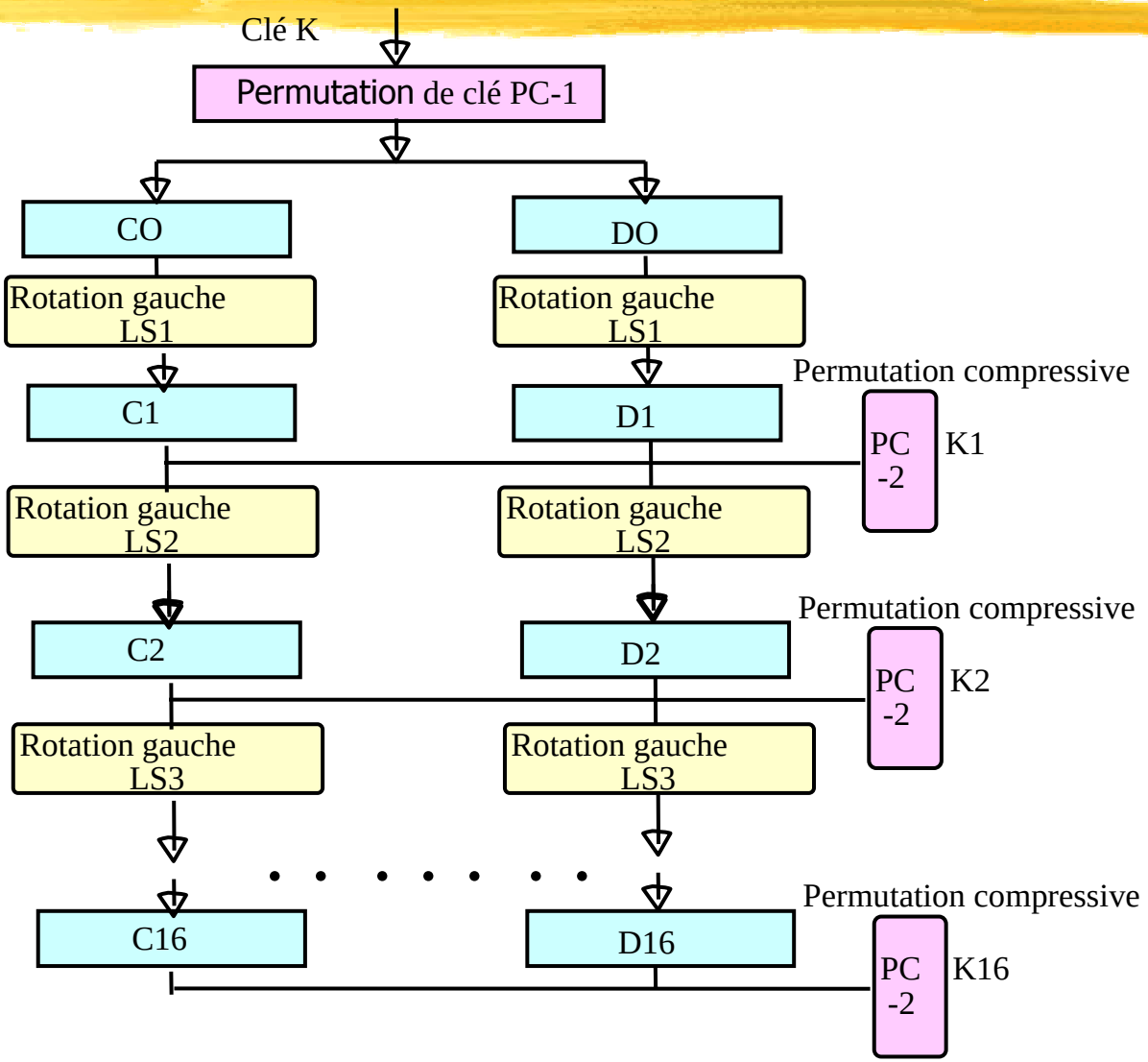
- **A chaque étage une permutation: toujours la même.**
- **Remplacer :** le bit 1 par le bit 16, le 2 par le 7 ….. (le 16 se retrouve en 1 et le 7 en 2 …. etc)
- **Contribue à la diffusion**.

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

DES : Génération des clés de ronde Différentes opérations

- Pour **chaque ronde** besoin d'une clé de ronde **spécifique** => 16 **clés différentes** sur 48 bits.
- **Réalisation d'une permutation initiale** sur la clé maîtresse (qui n'apporte rien).
- Chaque étage considère la clé de 56 bits comme **deux sous clés de 28 bits.**
- Chaque sous clé de 28 bits est **décalée à gauche** de une ou deux positions selon l'étage.
- Une **permutation compressive** est appliquée ensuite aux **56 bits pour générer 48 bits.**

DES : Génération des clés de ronde
Schéma général



DES : Explication de la méthode de génération des clés

- **Permutation PC-1** : à partir de 64 bits enlève les bits de parité et opère sur les 56 bits restants.

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

- Le résultat est divisé par **moitiés C0 et D0** (28 bits): on opère des rotations à gauche successives différentes selon les rondes

  => Table des valeurs de décalage pour les différentes clés

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1
1	2	4	6	8	10	12	14	15	17	19	21	23	25	27	28

DES : Explication de la méthode de génération des clés

- **Permutation PC-2** : on regroupe les deux moitiés de 28 bits pour former une clé de ronde de 48 bits (le bit 14 devient le 1, le bit 17 devient le 2 ….).

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

- Les bits 9, 18, 25, 35, 38, 43, 54 sont abandonnés : c'est pour cette raisons que la permutation est compressive.

DES : Améliorations

- **Faiblesse du DES :** la longueur de clé limitée à 56 bits.
- **Solution possible :** construire un sur-chiffrement à partir du DES en appliquant plusieurs fois de suite le DES.

- **Solution 1 :** Le double DES (le 2DES)

  $2DES_{k1k2}(M) = DES_{k2}(DES_{k1}(M))$

  Difficulté 2DES : pas vraiment meilleur que le DES.

- **Solution 2 :** Le triple DES (le 3DES ou TDES ici en mode EDE)

  $3DES_{k1k2k3}(M) = DES_{k3}(DES_{k2}^{-1}(DES_{k1}(M)))$

  Avantage : meilleure sécurité grâce à la clé de 168 bits.

  Inconvénient : 3DES prend trois fois plus de temps que DES.

- **Conclusion :** Quand les performances le permettent 3DES est recommandé.

DES : Sécurité

- **Le chiffre le plus controversé et le plus attaqué.**
- **Nombreuses attaques imaginées à propos du DES**

  A texte choisi : Pré-calcul exhaustif.

  Cryptanalyse linéaire.

  Cryptanalyse différentielle (le DES a peut-être été conçu pour parer ces attaques).

- **En 1996 bilan des moyens d'attaque.**
- **Finalement le DES est considéré comme un bon chiffre.**
- **Mais : clé 56 bits => possibilité des attaques force brute**

  Avec environ 200 KEuros de matériel on cassait le DES 56 bits en 20 jours.

  Avec 7,5 millions d'euros on cassait le DES 56 bits en 6 minutes.
- **L'attaque du DES est à la portée de plus en plus de monde.**

DES : Conclusion

- **DES a été pendant 25 ans le principal chiffre à clés secrètes.**
- **1976** : DES est certifié (standard officiel du gouvernement américain).
- **1981** : DES devient un standard pour le secteur privé;
- **1983** : DES est à nouveau certifié comme standard;
- **1987** : La NSA américaine ne veut plus certifier le DES, et propose à la place une série d'algorithmes de remplacement, tenus secrets
- **1988** : Suite à de longues discussions, le DES est finalement recertifié pour cinq ans, avec la promesse que ce sera pour la dernière fois;
- **1993** : Aucune alternative n'étant disponible, DES est recertifié pour cinq ans.

- **Problème posé des le départ: la taille de la clé** : 56 bits.
- En 25 ans les **méthodes de cryptanalyse** ont fait des progrès.
- Une **recherche exhaustive** peut être faite à condition d'en mettre le prix => **ne plus utiliser le DES sauf pour un chiffrement contre des attaques à petits moyens.**
- **Lancement d'un concours** en 1997 pour le successeur du DES.

Chiffres par blocs à clés secrètes

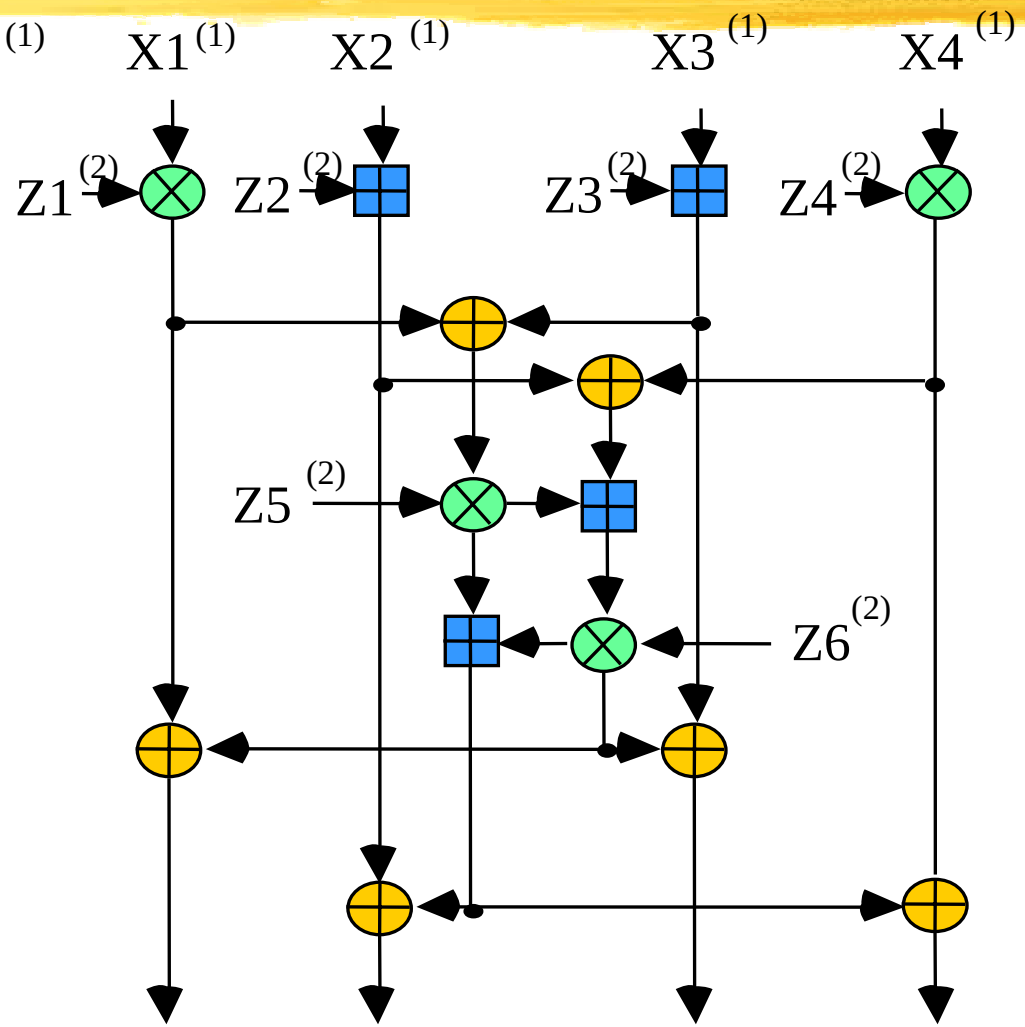
Le chiffre IDEA

‘International Data Encryption Algorithm’

IDEA 'International Data Encryption Algorithm' : Introduction

- **Chiffrement à clés privées** : 1990.
- **Longueur de clé** : 128 bits.
- **Chiffrement par blocs** : 64 bits (découpé en 4 fois 16).
- **Basé sur huit rondes :** ce n'est pas un chiffre de Feistel mais c'est aussi un chiffre qui est **son propre inverse.**
- **Réalisable aisément :** en matériel ou en logiciel (assez *bonnes performances)*.
- **Opérations utilisées :** arithmétique sur **16 bits:**
  - ou exclusif 
  - addition modulo 2^{16} 
  - multiplication modulo $2^{16}+1$ 

IDEA 'International Data Encryption Algorithm' : Schéma d'une ronde



7 autres étages analogues

(1) Xi : 4 blocs de 16 bits à chiffrer

(2) Zi : 6 Clés de ronde 16 bits soient 96 bits générées à partir de la clé initiale 128 bits par découpage et décalage

IDEA 'International Data Encryption Algorithm' : Conclusion

- **Breveté :** propriété ASCOM, usage non commercial libre.
- **Considéré par les spécialistes** comme un **bon chiffre** à clé privée.
- **Adopté dans différents systèmes de sécurité** : PGP 'Pretty Good Privacy' (messagerie électronique).
- **Longueur de clé plus élevée** que DES : 128 bits.
- **La vitesse de chiffrement** et de déchiffrement peut-être **élevée** au moyen de circuits spéciaux.
- Les attaques **ne semblent pas efficaces** mais le chiffre n'a pas été aussi attaqué que le DES.

Chiffres par blocs à clés secrètes

Le chiffre AES ‘Advanced Encryption Standard’ ou Rijndael

Introduction
Ensembles utilisés
Structure générale
Opérations de ronde
Génération des clés de ronde
Conclusion

AES / Rijndael : Les origines

- **Besoin** d'un chiffre en **remplacement du DES**.
- **Ouverture d'un concours** (septembre 1997) par le NIST (*National Institute of Standards and Technology* américain):
  - un algorithme **public, non-classifié, sans droits** d'utilisation.
  - un chiffre **symétrique** sur des blocs d'au moins **128 bits** et une taille de clé de **128, 192 et 256 bits.**
  - **de sécurité** au sens des méthodes de cryptanalyse connues.
  - **performant** pour toutes les implantations (des petits processeurs carte à puce aux circuits spécialisés).
- **15 candidats** -> pré sélection mars 1999 -> cinq finalistes: MARS ; RC6 ; Rijndael (1996) ; Serpent ; TwoFish
- **Résultats définitifs (2000):** le NIST choisit **Rijndael**.
- **Rijndael :** Construit selon les noms Vincent **Ri**jmen & Joan **Dae**men

AES / Rijndael : un chiffre à clé secrète par blocs

- A) **Blocs** : **128** bits ou **192** bits ou **256** bits.
- B) **Clés** de **128** bits ou **192** bits ou **256** bits.
- C) **Chiffre itéré** : de 10 à 14 rondes identiques.
- D) **Chaque ronde utilise une clé de ronde :** générée à partir de la clé principale.
- E) **AES n'est pas un chiffre de Feistel**
- F) **Le déchiffrement utilise une méthode différente** du chiffrement.

Maths pour AES/Rijndael : Calcul sur les octets (le corps GF(2^8))

- **Octet binaire b coefficients dans {0, 1}** : 8 bits
  Notation binaire (b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7)
  Notation en hexadécimal 'H_1 H_2'ou (H_1 H_2)$_H$
- **Polynôme** de degré ≤ 7 soit 8 coefficients binaires:
  $b(x) = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + b_4 x^4 + b_5 x^5 + b_6 x^6 + b_7 x^7$

- **Addition ⊕ des polynômes** (ou exclusif au niveau des bits 'xor' 1⊕1=0): addition des coefficients de chacun des polynômes modulo 2.
- **Multiplication⊗ des polynômes** : pour revenir au degré 7 il faut réduire modulo un polynôme m(x) de degré 8. c(x) = a(x) ⊗ b(x) mod (m(x)),
  Polynôme retenu pour Rijndael: **$m(x) = 1 + x + x^3 + x^4 + x^8$**

- **Structure du corps fini GF(2^8)** noté aussi **$F_2[x]/m(x)$** : m(x) irréductible
  256 octets binaires formant un corps avec l'addition et la multiplication précédente.
- **En particulier : tout polynôme binaire a(x) de degré au plus 7 a un inverse**: on peut calculer b(x) = a^{-1}(x) tel que a(x)⊗ b(x) = 1 mod (m(x)). (pour aller vite les 256 inverses sont tabulés).

Maths pour AES/Rijndael : Calcul sur les mots de 32 bits , l'anneau F_2^8 [x]

- **Utilisation de polynômes à coefficients dans GF(2^8) :** les coefficients de ces polynômes sont des octets (voir transparent précédent).
- **Mot de 32 bits (4 octets) :** Polynômes de degré ≤ 3 (ayant 4 coefficients)

- **Addition ⊕ des polynômes** : comme précédemment.
- **Multiplication ⊗ des polynômes** : comme précédemment pour revenir au degré 4 il faut réduire modulo un polynôme M(x) de degré 4.

  Dans AES/Rijndael **$M(x) = 1 \oplus x^4$**

- **Structure d'anneau de polynômes F_2^8 [x]** ($1 \oplus x^4$ n'est pas irréductible).
- **Produit par un polynôme A(x) fixé:** $C(x) = A(x) \otimes B(x) \mod (M(x))$
  - Réalisé en fait par une multiplication matricielle (matrice circulante)

$$\begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix} = \begin{bmatrix} A_0 & A_3 & A_2 & A_1 \\ A_1 & A_0 & A_3 & A_2 \\ A_2 & A_1 & A_0 & A_3 \\ A_3 & A_2 & A_1 & A_0 \end{bmatrix} \begin{bmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{bmatrix}$$

AES / Rijndael Généralités : Représentation de l'état et des clés

- **Etat** : le bloc initial en clair, un résultat intermédiaire, le résultat final chiffré: sur 128, 192 ou 256 bits.
- **Type de l'état : tableau rectangulaire** d'octets de 4 lignes, et 4, 6 ou 8 colonnes.
- **Représentation des clés** identique à celle de l'état.
- Pour une version donnée du chiffre, la clé à la **même structure** que l'état.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$	$a_{0,6}$	$a_{0,7}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,6}$	$a_{1,7}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,6}$	$a_{2,7}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,6}$	$a_{3,7}$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$	$k_{0,4}$	$k_{0,5}$	$k_{0,6}$	$k_{0,7}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$	$k_{1,4}$	$k_{1,5}$	$k_{1,6}$	$k_{1,7}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$	$k_{2,4}$	$k_{2,5}$	$k_{2,6}$	$k_{2,7}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$	$k_{3,4}$	$k_{3,5}$	$k_{3,6}$	$k_{3,7}$

AES / Rijndael Généralités : Pseudo code du chiffre

```
Rijndael(State,CipherKey) {
   # Génération des clés de ronde
   KeyExpansion(CipherKey,ExpandedKey) ;
   # Ou exclusif avec la première clé de ronde
   AddRoundKey(State,ExpandedKey);
   # Réalisation de Nr – 1 rondes identiques
   For( i=1 ; i<Nr ; i++ )
        Round(State,ExpandedKey + Nb*i) ;
   # Ronde finale un peu différente
   FinalRound(State,ExpandedKey + Nb*Nr) ; }
```

AES / Rijndael Généralités : Le nombre de rondes

- **Nombre de rondes variable.**
- **Fonction croissante de la taille du bloc à chiffrer Nb et de la taille des clés Nk.**
- **Remarque:** AES (diagonale seulement) un peu différent de Rijndael qui autorise plus de variabilité taille de bloc taille de clé.

	Nb=4	Nb=6	Nb=8
Nk=4	10	12	14
Nk=6	12	12	14
Nk=8	14	14	14

AES / Rijndael Généralités : Structure d'une ronde

- **1) Ronde standard** : Quatre transformations successives

```
Round(State,RoundKey) {
    ByteSub(State);                  # Substitution octet par octet
    ShiftRow(State);                 # Transposition des lignes
    MixColumn(State);                # Mélange sur les colonnes
    AddRoundKey(State,RoundKey);} # Ou exclusif avec la clé
```

- **2) Ronde finale** : Supprime mixcolumn pour le déchiffrage

```
FinalRound(State,RoundKey) {
    ByteSub(State) ;                      # Substitution
    ShiftRow(State) ;                     # Transposition
    AddRoundKey(State,RoundKey);}  # Ou exclusif
```

AES / Rijndael : Opération de ronde Substitution sur les octets 'ByteSub'

- **La même substitution** sur chaque **octet** X de l'état.

A) On prend **l'inverse multiplicatif** de X dans GF(2^8) Z=1/X

B) On applique à **l'octet résultat la substitution** mono-alphabétique affine $Y = K_1 Z \oplus K_2$ (matrice K_1 et vecteur K_2)

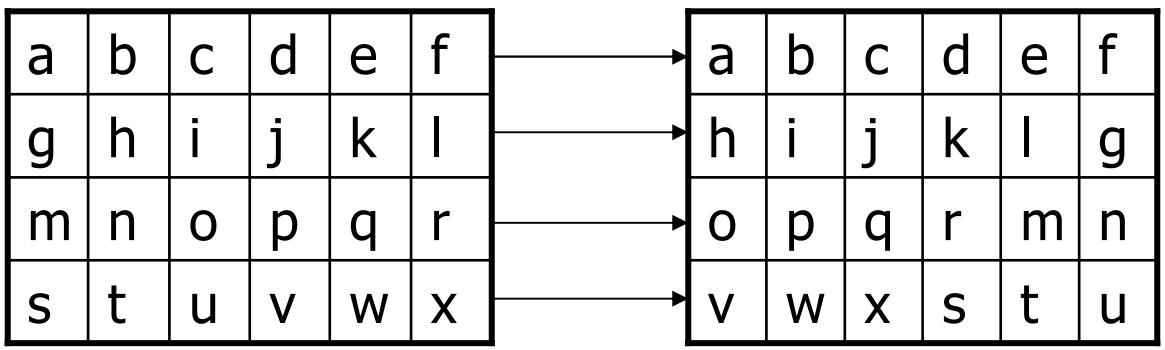
$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

- **Objectif** : **définir une transformation non linéaire** **($Y = K_1 . 1/X \oplus K_2$)** : un calcul **facile à tabuler.**

AES / Rijndael : Opération de ronde Transposition des lignes 'ShiftRow'

- **Chaque ligne** de l'état est **décalée circulairement** (décalage par octets).
- **Tableau des décalages** pour chaque ligne selon la taille de l'état en nombre de colonnes (pas de décalage pour la première ligne, un pour la seconde…).
- **Exemple de décalage** pour le tableau 4,6 (192 bits).

Nb	L0	L1	L2	L3
4	0	1	2	3
6	0	1	2	3
8	0	1	3	4



- **Objectif : Définir une transposition qui garantit la diffusion dans les lignes (sur plusieurs rondes).**

AES / Rijndael : Opération de ronde Mélange des colonnes 'MixColumn'

- **Les colonnes forment des mots de 32 bits.**
- **Chaque colonne considérée comme un polynôme sur F_2^8 [x]** est multipliée par le polynôme c(x) fixe (modulo $x^4 \oplus 1$):

$$c(x) = \text{'03'}\, x^3 \oplus \text{'01'}\, x^2 \oplus \text{'01'}\, x \oplus \text{'02'}$$

- **Chiffrement en pratique :** multiplication par la matrice circulante.

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

$$b_0 = \text{'02'}\, a_0 \oplus \text{'03'}\, a_1 \oplus \text{'01'}\, a_2 \oplus \text{'01'}\, a_3$$

- **Déchiffrement: multiplication par l'inverse du polynôme c(x):**

$$c^{-1}(x) = \text{'0B'}\, x^3 \oplus \text{'0D'}\, x^2 \oplus \text{'09'}\, x \oplus \text{'0E'}$$

- **Objectif : Définir une transformation qui garantit une bonne diffusion entre colonnes (sur plusieurs rondes). Calculable facilement.**

AES / Rijndael : Opération de ronde Addition clé de ronde 'Add Round Key'

- **Ou exclusif** entre la valeur de l'état et la valeur de la clé de ronde (exactement comme en DES).
- **Déchiffrement:** réalisation du ou exclusif **inverse.**

$b_{0,0}$	$b_{0,1}$	$b_{0,2}$	$b_{0,3}$	$b_{0,4}$	$b_{0,5}$	$b_{0,6}$	$b_{0,7}$
$b_{1,0}$	$b_{1,1}$	$b_{1,2}$	$b_{1,3}$	$b_{1,4}$	$b_{1,5}$	$b_{1,6}$	$b_{1,7}$
$b_{2,0}$	$b_{2,1}$	$b_{2,2}$	$b_{2,3}$	$b_{2,4}$	$b_{2,5}$	$b_{2,6}$	$b_{2,7}$
$b_{3,0}$	$b_{3,1}$	$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	$b_{3,5}$	$b_{3,6}$	$b_{3,7}$

$=$

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$	$a_{0,6}$	$a_{0,7}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,6}$	$a_{1,7}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,6}$	$a_{2,7}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,6}$	$a_{3,7}$

$\oplus$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$	$k_{0,4}$	$k_{0,5}$	$k_{0,6}$	$k_{0,7}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$	$k_{1,4}$	$k_{1,5}$	$k_{1,6}$	$k_{1,7}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$	$k_{2,4}$	$k_{2,5}$	$k_{2,6}$	$k_{2,7}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$	$k_{3,4}$	$k_{3,5}$	$k_{3,6}$	$k_{3,7}$

- **Objectif : Faire que chaque ronde dépende de la clé secrète.**

AES / Rijndael : Génération des clés de ronde (cas de 4 ou 6 colonnes)

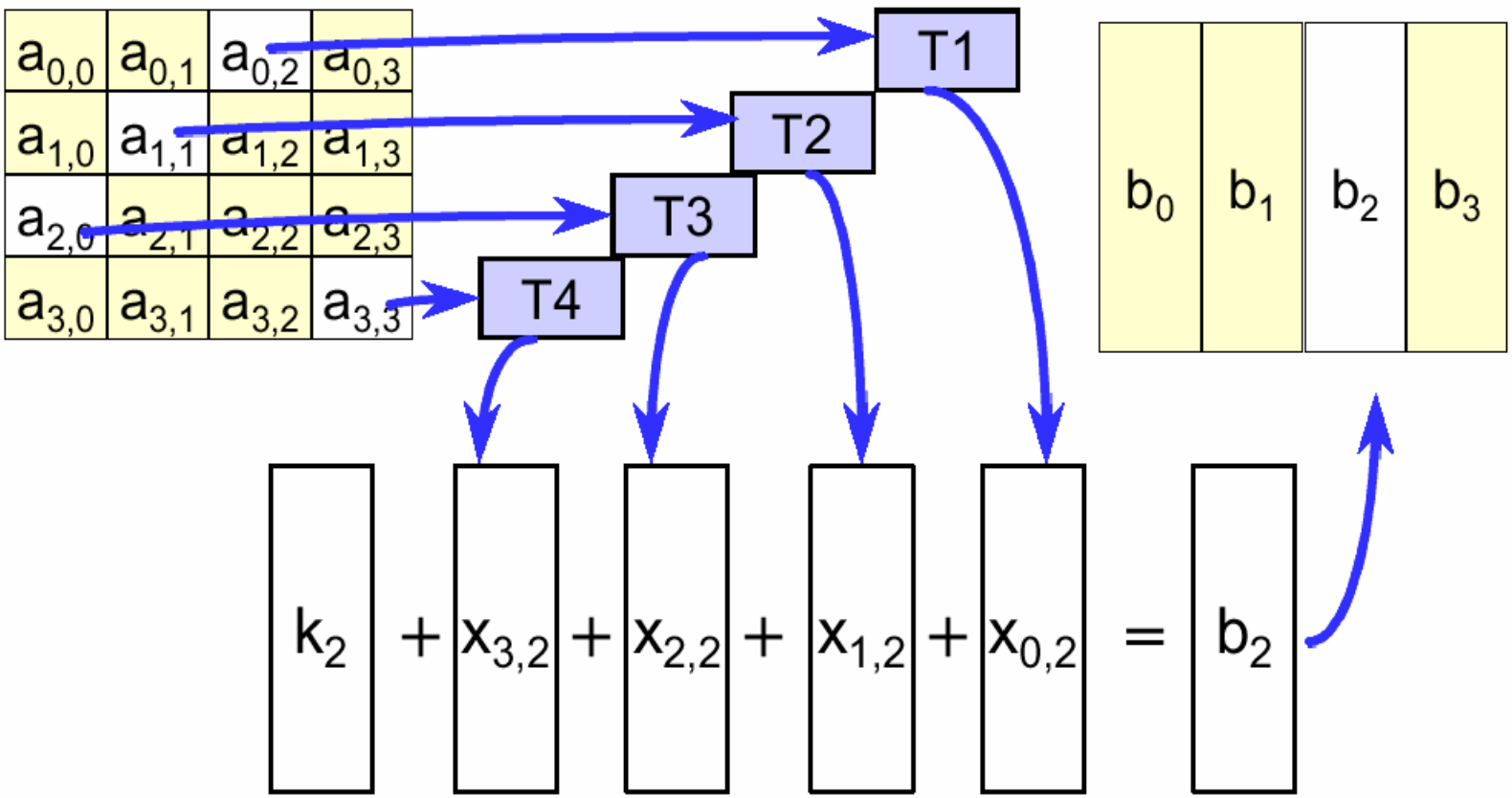
- **Code pour l'expansion de la clé maîtresse si** $N_k \leq 6$

```
KeyExpansion(byte Key[4*Nk] , word W[Nb*(Nr+1)]) {
    for(i = 0; i < Nk; i++)
        W[i] = (Key[4*i],Key[4*i+1],Key[4*i+2],Key[4*i+3]);
    for(i = Nk; i < Nb * (Nr + 1); i++) {
        temp = W[i - 1];
        if (i mod Nk == 0)
            temp = SubByte(RotByte(temp)) ⊕ Rcon[i / Nk];
        W[i] = W[i - Nk] ⊕ temp; } }
```

- **Clé maîtresse :** tableau d'octets key (4 *N_k).
- **Tableau des clés de ronde:** tableau de mots de 32 bits W[Nb*(Nr+1). On génère Nr+1 (nombre de rondes plus une clé) qui font Nb mots de 32 bits.
- **Temp :** variable auxiliaire 32 bits.
- **SubByte** : substitution par octets, **RotByte** : décalage par octet (un octet) **Rcon** une constante.

AES / Rijndael : Un chiffrement rapide

- AES sur 128 bits : 16 recherches en table et 16 ou exclusif par ronde (selon figure Rijmen/Daemen)



AES / Rijndael : Conclusion

- **1) Sécurité :** actuellement pas de méthode de cryptanalyse pour AES connue.
- **2) Performances :** structure symétrique et parallèle du chiffre.
  - Bien adapté aux processeurs actuels (Pentium, processeurs RISC).
  - Adapté aux processeurs des cartes à puces.
  - Adapté à la réalisation en circuit intégré.
- **Utilisable dans de nombreux protocoles de sécurité** : WIFI, IPSEC…

Chiffres à clés secrètes

Chiffres en continu
(‘Stream Ciphers’)

Principes généraux
A5/1 (GSM)
RC4 (Wifi)

Principes généraux des chiffres en continu

- **Chiffre en continu:** un chiffre qui travaille bit par bit ou octet par octet
  - **Anglais : Stream cipher** ou chiffrement par flot
  - Quelquefois **State cipher** chiffrement dépendant d'un état interne.
- **Combinaison** d'un texte en clair en ou exclusif avec un flot de clés obtenu par un générateur de nombres pseudo aléatoires.
- **La même clé ne doit jamais être utilisée deux fois**.
- **Solution rapide** en chiffrement, **plus facile à implanter en matériel** (solution appréciée des électroniciens).
- **Solutions phares A5, RC4** : mal considérés en sécurité.
- **Variantes :**
  - **Synchronous Stream Cipher** : générateur de nombres aléatoires à partir d'une clé (il faut un mécanisme externe de synchronisation).
  - **Self-synchronizing Stream Ciphers** : on utilise les données des n symboles chiffrés précédents pour générer la clé (si on a perdu la synchronisation on se resynchronise après n symboles).

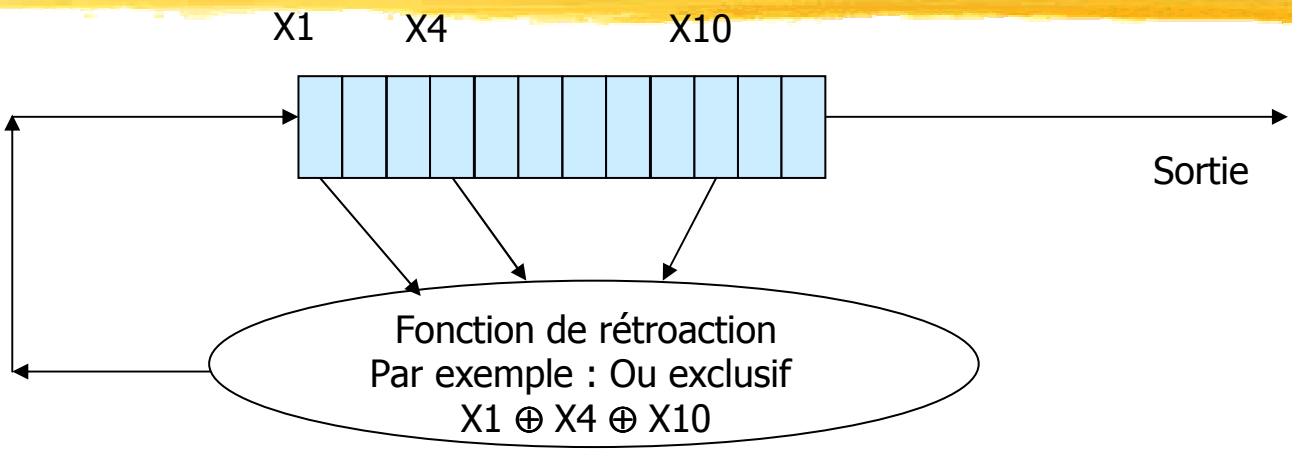
Chiffres en continu ('Stream Ciphers')

Registres à décalage à rétroaction linéaire

LFSR 'Linear Feedback Shift Register'

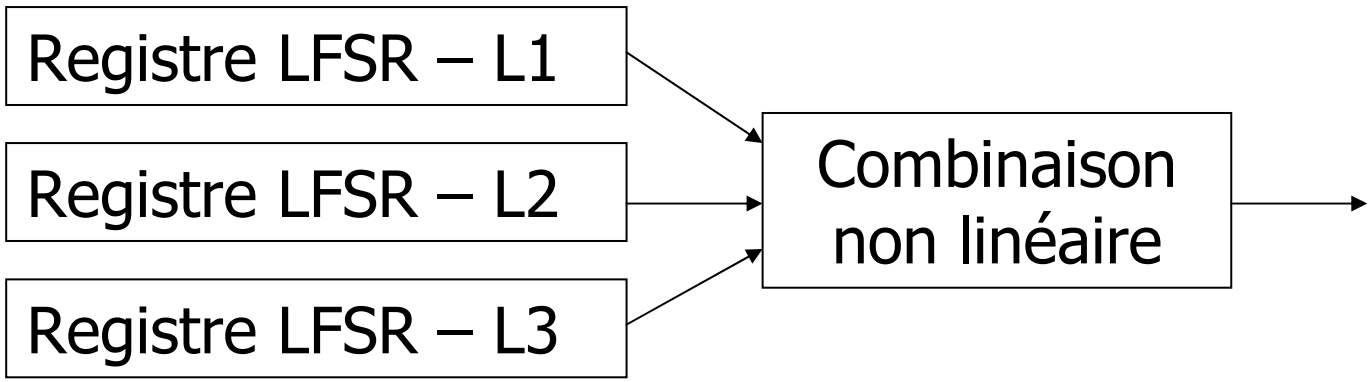
Exemple du chiffre A5/1 (GSM)

Registres à décalage à rétroaction linéaire : LFSR



- **Construction d'un générateur** de nombres aléatoires : des clés.
- **Registre (Register)** : suites binaires stockées dans un registre (clés).
- **Décalage (Shift)** : à droite d'une position à chaque nouveau bit.
- **Linéarité (Linear)** : une fonction de certains bits du registre en ou exclusif (XOR) (autres fonctions possibles -> non linéaires).
- **Rétroaction (Feedback)** : on replace le résultat du calcul dans le bit le plus à gauche.
- **Initialisation du registre** : une clé secrète.
- **Cyclicité des clés produites** : les sorties sont prévisibles selon un cycle peu long.

Combinaisons non linéaires de registres à décalages linéaires



- **Pour lutter contre la mauvaise qualité** de chiffrement des registres à rétroaction linéaire.
- **Utilisation de plusieurs registres.**
- **Combinaison non linéaire des résultats des registres pour allonger la cyclicité des clés produites.**
- **Exemple de fonction non linéaire** (Geffe):

$$X1\ X2 \oplus X2\ X3 \oplus X3$$

Le chiffre A5/1 du GSM

- **A5 version 1 :** chiffrement en continu
- **Trois registres LFSR:** 19, 22 et 23 bits (64 bits).
- **Clé secrète de 64-bit :** pour initialiser les registres.
- **Utilisation du numéro de trame TDMA :** 22 bits pour différencier les clés (ne pas utiliser la même).
- **Une fonction majorité (non linéaire) :** produit un bit à partir des trois sorties.
- **Fabrication de deux flots de clé regroupés par groupes de 114 bits :** utilisés pour chiffrer chaque sens de communication (ou exclusif sur 114 bits)
- **A5 aurait une longueur de clé "effective" de 40 bits => ** essais force brute rapide => décrypté en quelques heures ou quelques jours selon les moyens.

Chiffres en continu ('Stream Cipher')

Le chiffre RC4

"Rivest Cipher N°4"

"Ron's Code N°4"

RC4 : Introduction

- **1) Chiffre propriétaire à clés privées** société RSA-DSI 1987
- **2) Algorithme secret mais le programme a été désassemblé et publié en 1994 :**
  - Le nom **RC4 est resté protégé** (Rivest Cipher n°4).
  - Usage libre possible (utilisation du nom 'arcfour')
- **3) Longueur de clé variable**: ex de longueur 40, 104 bits.
- **4) Fonctionnement en mode flot de clés 'Stream cipher'**
  - Génération d'une séquence pseudo aléatoire d'octets.
  - Le message en clair est en ou exclusif avec cette séquence.
- **5) Extrêmement simple et implantation très efficace** : en matériel ou en logiciel.
- **6) Les opérations utilisées sont des transpositions.**

RC4 : Pseudo code du chiffre

1) Initialisation du tableau S de 256 octets

pour i = 0..255
  S[i] := i ;
pour i = 0..255
  j:=(j+S[i]+key[i mod key_length]) mod 256;
  aux:=S[i] ; S[i]:=S[j] ; S[j]:=aux ; **(échange S[i], S[j])**

2) Boucle de chiffrement/déchiffrement octet par octet

i:=0; j:=0;
répéter
  i:=(i+1) mod 256;
  j:=(j+S[i]) mod 256;
  aux:=S[i] ; S[i]:=S[j] ; S[j]:=aux ; **(échange de S[i], S[j])**
  k=S[(S[i]+S[j]) mod 256]
  octet_chiffré := octet_en_clair XOR k ;
jusqu'à fin du texte en clair ;

RC4 : La méthode Java

```
public void RC4_stream_cipher(int data[]) {
    int aux, xorIndice, i, j;
    for (int compteur = 0; compteur < data.length; compteur++) {
        // Calcul des indices
        i = (i + 1) % SSize;
        j = (j + S[i]) % SSize;
        // Echange des contenus de S[i] et S[j]
        aux = S[i];
        S[i] = S[j];
        S[j] = aux;
        // Ou exclusif XOR d'un octet state avec un octet de données
        xorIndice = (S[i] + S[j]) % SSize;
        data[compteur] ^= S[xorIndice];
                                                    }
}
```

RC4 : Conclusion

- **Extrêmement rapide :** environ 10 fois plus rapide que le DES => Beaucoup d'intérêt pour ce chiffre.
- **Utilisation dans de très nombreux outils:** SSL, Lotus Notes, Encryptage de mots de passe Windows, MS Access, Adobe Acrobat, MS-PPTP, Oracle Secure SQL, Wifi ...
- **Cryptanalyse aisée de l'implantation RC4 / Wifi en Wep**:
  - Mauvaise implantation des vecteurs d'initialisation 24 bits.
  - Mauvaise sécurité du mécanisme d'intégrité.
  - Très facile pour des clés courtes (40 + 24 bits), pas plus difficile pour des clés de 104 + 24 bits (attaque réussie en assez peu de temps).
  - **Recommandation : un chiffre à éviter en Wifi.**
- **RC4 bien utilisé, avec des clés longues, est encore considéré comme sécuritaire.**

La cryptographie: Chapitre II

Les chiffres dissymétriques (à clé publique)

Introduction

RSA

El Gamal

Chiffres à clé publique : Introduction

- **Problème des chiffres à clés privées** : la distribution des clés.
- **En solution à clés secrètes** : pour communiquer dans un groupe de n participants il faut **n(n-1)/2 clés**.
- **Distribution des clés** à réaliser en **utilisant le réseau** (pour être pratique).
- **La cryptographie à clés publique (**1976)
  - **Diffie et Hellman** définissent les principes d'une nouvelle approche en cryptographie sans proposer de solution concrète au problème qu'ils posent.
  - **Diffie et Hellman** proposent un protocole d'échange de clés.

Chiffres à clé publique: Rappel des principes de l'approche Diffie-Hellman

- **Deux fonctions** Ek et Dk' dépendent de clés k et k' différentes: **Ek** est la méthode de **chiffrement**.
  Dk' est la méthode de **déchiffrement**.
- **Propriété de base**: le déchiffrement est l'inverse du chiffrement.

  Dk' (Ek (M)) = M
- **Propriété majeure : Il est très difficile de déduire le déchiffrement Dk' de la connaissance la méthode de chiffrement Ek** => Un problème de complexité très élevée.

- **Idéalement** : Ek(M) et Dk'(M) devraient être **faciles à calculer**.

Chiffres à clé publique : Une révolution en cryptographie

- **Chaque utilisateur** reçoit un couple (Ek, Dk')
- **Ek peut-être rendue publique** (en fait k est publiée par exemple dans un **annuaire)**.
- **Tout le monde peut connaître Ek** et envoyer des messages confidentiels qu'**un seul destinataire** (celui qui connaît Dk') **peut comprendre.**
- **Dk' reste privée** : k' reste secrète et nécessairement différente de k**.**
- **Hypothèse fondamentale** d'un tel système : **on ne doit pas pouvoir trouver Dk' quand on connaît Ek et des messages chiffrés par Ek** (casser Dk en essayant des messages connus)**.**

Cryptographie à clé publique et fonctions à sens unique

- **Notion de fonction à sens unique 'One Way function':** une notion issue de la théorie de la complexité.
- **Définition (informelle)** : Une fonction f(M) **facile à calculer** mais telle qu'il est **extrêmement difficile** de calculer la fonction inverse (de déduire M de f(M) pour un M aléatoire).
  - On peut éventuellement connaître un petit nombre de valeurs pour lesquelles la fonction inverse est connue mais hors de ces valeurs le problème est difficile.
- **Idéalement on aimerait trouver f : calculer f(M) est un problème polynomial mais calculer f^{-1} problème NP**

  => On ne peut pas **affirmer** que de **telles fonctions existent** car cela équivaut à affirmer que P ≠ NP.
- En fait on cherche seulement des **fonctions qui rendent le service : f est rapide** mais il n'existe **pas de méthodes de faible complexité connues** pour calculer l'inverse.

Cryptographie à clé publique et fonctions à sens unique 'One way'

- **Deux exemples : fonctions faciles à calculer**
  - 1) $C=M^2$ (mod n=pq) ;
  - 2) $C= M^e$ (mod n) avec M et n entiers.
- **Difficiles à inverser : fonctions difficiles à calculer**
  - 1) Racine_carrée(C) (mod n=pq) ;
  - 2) log(C) (mod n) avec C et n entiers.
- **Utilisation des fonctions à sens unique**
  - Pour garder sous forme **inaccessible des mots de passe.**
  - Pour la **construction de chiffres** : problème une fois M chiffré on ne sait plus déchiffrer M.

=> Une définition **complémentaire**: fonctions à **sens unique avec trappe secrète**.

Cryptographie à clé publique : Fonction à sens unique à trappe ('trapdoor')

- **Définition :** fonction à **sens unique à trappe** (ou à **brèche** secrète)
  - C = E(M) **facile à calculer** telle qu'il est **difficile** (problème de complexité élevée) de déduire **$D=E^{-1}$** telle que D(E(M))=M **sauf si l'on connaît un secret K** (une trappe, une brèche).
- **Une autre façon de présenter la cryptographie à clé publique.**
- **La publication de la fonction E est possible** : elle ne doit pas permettre de trouver D)
  - **E** encryptage (**partie publique**),
  - **D** Déchiffrement (**partie secrète**).

Chiffres à clé publique : Quelques exemples

- **Merckle, Hellman (1977)**
  - Chiffrement : problème de sac à dos.
  - Décryptage : assez rapidement découvert.
- **RSA Rivest, Shamir, Adleman (1978)**
  - Chiffrement : Elévation à la puissance n dans un corps fini.
  - Décryptage : Factorisation.
- **Rabin 1979**
  - Elévation au carré dans un corps fini
  - Décryptage trouver une racine carrée dans un corps fini.
- **El Gamal 1985**
  - Chiffrement : exponentiation
  - Décryptage problème du logarithme discret
- **Cryptographie elliptique Koblitz-Miller 1985**
  - Utilisation des propriétés des fonctions elliptiques.

Chiffres à clés publiques

Le chiffre RSA
‘Rivest, Shamir, Adleman’

Méthodes de chiffrement et de déchiffrement

Détermination des clés

Réalisation des calculs du RSA

Sécurité du RSA

RSA : Introduction

- **Idée pour la confusion** : élévation d'un entier à une puissance entière modulo un grand entier (Diffie, Hellman).
- **Utiliser la complexité algorithmique de la factorisation d'un grand nombre n en deux facteurs premiers p et q:** problème pour lequel on ne connaît pas d'algorithme efficace.
- **Solution finalement très différente et très puissante.**
  - Ronald.L.**Rivest**, Adi.**Shamir**, Leonard.M.**Adleman** (MIT)

`*A method for obtaining digital signatures and public-key cryptosystems*' Communications of the ACM, 21(2):120,126, 1978.

- **Solution brevetée** pendant une vingtaine d'année puis tombée dans le domaine public.

RSA : Chiffrement et déchiffrement

- **Fonction de chiffrement E (partie publique).**

  La clé publique comporte deux entiers: **k = (e, n)**

  Chiffrement d'un bloc M considéré comme un entier b bits

  On élève M à la puissance e modulo n:

  $$\mathbf{Ek\ (M) = M^e\ (mod\ n)}$$

- **Fonction de déchiffrement D (partie secrète)**

  La clé secrète est un couple d'entiers: **k' = (d, n)**

  Déchiffrement d'un bloc chiffré C sur b bits

  Elévation à la puissance d modulo n:

  $$\mathbf{Dk'\ (C) = C^d\ (mod\ n)}$$

- **Remarque:** Les entiers n, e, d doivent être choisis selon des règles précises.

RSA : Détermination des clés

Détermination de n: valeur de modulo

- **Choisir deux entiers premiers p et q** (différents, grands, aléatoires).
- **Calculer n = p . q**
- La sécurité de RSA repose sur la **difficulté de factoriser** un entier n en deux entiers premiers p et q => n grand 320 bits, 512 bits, 1024 bits ….
- La taille de n **conditionne la vitesse** de chiffrement et de déchiffrement.

Détermination de la clé publique (e , n)

- Calculer **z = (p-1) (q-1)**
- Choisir un entier **e premier avec z** (1< e < z , e et z n'ont pas de diviseurs communs).

Détermination de la clé privée (d , n)

- Choisir un entier d tel que :
- **e d = 1 (mod z)** d est un inverse de e dans l'arithmétique modulo z (soit encore ed = k (p-1)(q-1) + 1).

RSA : Remarques complémentaires

- **1) Contrainte RSA : les blocs à chiffrer M sont des entiers inférieurs à l'entier n**
  - M plus petit que n => les calculs sont conduits modulo n.
  - Exemple : si on choisit n sur d+1 bits, on chiffre des messages M de d bits (taille inférieure) => les messages chiffrés sont sur d+1 bits (restes modulo n).
  - Autre aspect: utilisation d'une méthode de bourrage des messages en clair courts pour ne pas chiffrer de trop petites valeurs.
- **2) Propriété liée à la méthode RSA :** commutativité des opérations de chiffrement et de déchiffrement:
  - $D (E (M)) = E (D (M)) = M$
  - $(M^e)^d = (M^d)^e = M^{ed} \pmod n$

RSA : Preuve de fonctionnement Bases mathématiques

- **Notion de fonction indicatrice d'Euler ϕ**
  - Pour n entier: $z = \phi(n)$ l'indicatrice d'Euler est définie comme le nombre d'entiers premiers avec n plus petits que n: $\phi(n) = \text{card}\ \{ j : 1 \leq j \leq n\ ,\ \text{pgcd}(j,n) = 1 \}$
  - Exemples simples d'indicatrice d'Euler :
    - Si n est premier : $\phi(n) = n-1$
    - Si $n = p \cdot q$ avec p et q premiers : $\phi(n) = (p-1).(q-1)$
- **Théorème de Fermat-Euler** (Euler extension du petit théorème de Fermat)
  - Fermat : si n est premier et $a<n$: $a^{n-1} = 1 \pmod n$
  - Euler : Si a et n sont premiers entre eux ($\text{pgcd}(a,n) = 1$)
    $a^{\phi(n)} = 1 \pmod n$

RSA : Preuve de fonctionnement
Déchiffrement inverse du chiffrement

- **Chiffrement suivi du déchiffrement** d'un bloc M :

$$D(E(M)) = ((M)^e \pmod n)^d \pmod n$$
$$= (M^e)^d \pmod n = M^{e.d} \pmod n$$
$$= M \qquad \text{On doit retrouver M}$$

- **Rappel:** on a choisi : $e.d = 1 \pmod z$ Notons $e.d = j\,z + 1$
- **On doit avoir** : $M^{e.d} \pmod n = M^{j.z}\, M \pmod n = M \pmod n$

- **En effet** $M^{jz} = (M^z)^j = (M^{\phi(n)})^j = (1)^j = 1 \pmod n$
- **Parce que théorème d'Euler** : si M et n sont premiers entre eux $M^{\phi(n)} = 1 \pmod n$

RSA : Exemple tiré de B Schneier
Choix des clés

- **Choisir deux entiers premiers** : p = 47, q = 71
- **Calculer :** n = p . q = 3337
- **Calculer :** z = ϕ(n) = (p-1) . (q-1) = 46 . 70 = 3220
- **Choisir e (la clé publique):** exemple e = 79 (premier avec z)
- **Choisir d** : inverse de e (modulo z)

A) Solution habituelle: l'algorithme d'Euclide étendu

Comme on a ed = k z + 1 soit ed + k (p-1)(q-1) = 1

On doit résoudre l'identité de Bezout ici 79 d + 3220 k = 1

Utilisation de divisions successives et de substitutions: d=1019

B) Autre solution possible: utiliser le théorème d'Euler

$e.d = e\ e^{-1} = 1 = e^{\phi(n)} = e\ e^{\phi(n)-1} \pmod z$

Donc $d = e^{-1} = e^{\phi(n)-1} \pmod z$

Numériquement $79^{3219} \pmod{3220} = 1019$ => d=1019

RSA : Exemple tiré de B Schneier Chiffrement et déchiffrement

- **Chiffrer un message M** : M= 688232687966683
- **Décomposition en blocs** de taille inférieure à n= 3337
  => Ici des blocs de 3 chiffres M= 688 232 687 966 668 3
- **Chiffrer 688**: 688^{79} (mod 3337) = 1570 puis les autres blocs.
  => E(M) = 1570 2756 2091 2276 2423 158 6
- **Déchiffrer bloc par bloc** en commençant par 1570.
  1570^{1019} (mod 3337) = 688 etc ….

- **RSA pose deux problèmes d'algorithmique difficiles :**
  - **Trouver de grands nombres premiers.**
  - **Elever à la puissance modulo n** de grands entiers

RSA Algorithmique : Calcul des puissances modulo n

- **Chiffrer M : c'est calculer $C = M^e \pmod n$**
- **e sur k bits** : décomposition en binaire $e = \Sigma_{i=0,k-1}\ e(i)\ 2^i$

```
début
    C := 1 ;                                  // Initialisation résultat
    pour i := k-1 à 0 faire                   // k étapes
            C := C * C (mod n) ;              // Elever le résultat au carré
            si ( e(i) = 1 ) alors             // Si le bit est à 1
                    C := C * M (mod n) ;      // Multiplier par M
            finsi;
    finpour;
fin;
```

- Exemple $M^5 = M^{b'101'} = M^4 * M^1$.
  - C=1 ; C*C = 1 ; bit de fort poids =1 : multiplier par M.
  - C= M ; C*C = M^2 ; bit intermédiaire = 0 : ne pas multiplier.
  - C = M^2 ; C*C = M^4 ; dernier bit = 1 : multiplier par M.
- Autre problème : accélérer les multiplications modulo n.

RSA Algorithmique : Calcul des multiplications modulo n

- **Faire des calculs modulo n très grand :** avec n = pq.
  - n de longueur 300 à 2048 bits avec des opérations de 32 bits.
- **Solution des restes chinois** : Sun Tsu (1er siècle)
- **Représentation des restes chinois de x (mod n):**

  si n = pq : x est un couple (a , b) = (x (mod p) , x (mod q)).
- **Reconstruction de x**: $x = (((a-b)(q^{-1} \pmod p) \pmod p)q + b$
- **Pour un x donné** : un seul (a,b) et réciproquement.
- **x+y est représenté par** : (x+y (mod p) , x+y (mod q))
  - Pas vraiment de gain en temps de calcul (deux fois k/2 opérations)
- **xy est représenté par :** (xy (mod p) , xy (mod q))
  - Temps de calcul d'une multiplication divisé par deux.
- **Pour le calcul du RSA $C = M^e \pmod n$** calculer le couple

  $(M^{e \pmod{(p-1)}} \pmod p , M^{e \pmod{(q-1)}} \pmod q)$
  - Même nombre d'opérations ariths en modulo p ou q qu'en modulo N.
  - En longueur moitié => Temps de calcul total divisé par 3 ou 4.

RSA Algorithmique : choix de grands nombres premiers déterministes

Méthodes déterministes

- **Méthode type** : le crible d'Erathostène
  - Tester par des divisions tous les nombres en déterminant progressivement la liste des nombres premiers.
  - Inutilisable pour les grands nombres dont on a besoin.
- **Résultats théoriques** prouvant que certains nombres sont premiers.
  - Nombres premiers trop bien connus : peu sécuritaires.
- **Vrais grands nombres premiers originaux :**
  - Besoin d'une puissance de calcul énorme
  - Domaine protégé militaire

RSA Algorithmique : Grands nombres premiers probabilistes

- **Choisir aléatoirement un grand entier p** : tester si ce nombre est premier en probabilité.
- **Utiliser un test pour filtrer les nombres décomposables** : notion de test de primalité.
  - Test de Rabin-Miller
  - Test de Soloway- Strassen
  - Progrès algorithmiques toujours en cours (été 2005).
- **On applique n fois le test** : si test = VRAI la probabilité que p soit décomposable est inférieure à 2^{-n} .
- **Pour la probabilité 2^{-128}** : que le nombre ne soit pas premier proba très inférieure à tous les autres problèmes de pannes possibles.
- **Si le nombre n'est pas premier** : pas de problème de sécurité.
- **Si le nombre n'est pas premier** Problème possible : ne pas pouvoir déchiffrer (une solution faire des essais).

RSA Algorithmique Grands nombres premiers : Méthode de Rabin-Miller

Procédure Rabin_Miller (n , test , m) // Test n entier impair non premier à 2^{-m}
Début k := 0 ; test := VRAI ; // On fait k essais; le test doit rester vrai
Calculer s et t tel que s est impair et 2^t . s = n-1 ; // On utilise Fermat $a^{n-1} = 1 \pmod n$
tant que (k ≤ m et test = VRAI) **faire** // On teste jusqu'à la proba exigée
Choisir un nombre aléatoire a tel que 2 ≤ a ≤ n-1 ; // Sur des entiers a aléatoires
Calculer v := $a^s \pmod n$; // v est l'exponentielle puissance s de a modulo n.
si (v ≠ 1) **alors** // Si n est premier la suite v , v^2 , v^4 , v^8 , … v^2^t doit
i := 0 ; // finir à 1 et l'avant dernière valeur est n-1
tant que (v ≠ n-1) **faire**
si (i = t-1) **alors** test := FAUX; // On a trouvé n-1 trop tôt
sinon v = v*v (mod n) ; i := i+1 ; **finsi ;** // On continue à tester
fin tant que;
finsi ;
k := k + 2 ; // On continue a avoir test = vrai et on a divisé la probabilité par 4.
fin tant que;
fin

RSA Algorithmique Grands nombres premiers :Méthode de Soloway-Strassen

- **On utilise pour deux nombres a et p une fonction : le symbole de Jacobi J(a,p) non décrit ici (calcul récursif assez compliqué)**

début

i := 0;
test := VRAI ;
tant que (i ≤ m et test = VRAI) **faire**
i := i + 1 ;
choisir aléatoirement a (1≤a≤p) ;
calculer j = a(p-1)/2 mod p ;
calculer J(a,p) symbole de Jacobi;
si (j ≠ J(a,p)) **alors**
test := FAUX;
finsi ;
fin tant que;

fin

- Si test = VRAI la probabilité que p soit décomposable est inférieure à 2^{-m} .
- A chaque étape si l'on passe le test on a 50% de chances de décider que le nombre est décomposable et 50% de chances de ne pas pouvoir décider.

RSA sécurité (intuitivement) : diffusion et confusion

- **Diffusion RSA :** Argument de la taille des blocs, il faut que les blocs chiffrés soient de taille importante.
- **Confusion RSA :** Pour rendre le sens inaccessible.

Utilisation de l'élévation à la puissance puis utilisation d'un modulo.

- L'élévation a une puissance permet de changer **le registre des entiers choisis**. Exemple précédent e = 79 et n = 3337

  Pour M = 687, M' = 688 peu différents.

  $M^{79} = 687^{79}$ $M'^{79} = 688^{79}$

  => valeurs gigantesques très différentes

- Le modulo n introduit des **discontinuités**

  $M^{79} = 687^{79} \pmod{3337} = 2091$

  $M'^{79} = 688^{79} \pmod{3337} = 1540$

RSA : Principes de mise en œuvre pour la sécurité sur n = pq

- **1) n = pq doit être grand:** difficulté de factorisation.
- **2) p et q doivent être grands tous les deux:** ne pas choisir l'un des deux petit.
- **3) p et q ne doivent pas être prévisibles :** pouvant être sortis d'un dictionnaire de nombres premiers.
- **4) p et q doivent être de taille voisine**
  - Rendre difficile la factorisation par différences de carrés

$$pq = [(p+q)/2]^2 - [(p-q)/2]^2$$

  - On peut essayer des petits carrés en (p-q)/2 mais alors il faut prendre la racine carrée d'un grand nombre
- **5) Ne pas utiliser le même n pour plusieurs clés :** si ces clés doivent chiffrer le même message cela facilite la factorisation.

RSA : Autres principes de mise en œuvre pour la sécurité

- **6) Ne pas chiffrer des messages de petite taille**
  - Taille petite : ne favorise pas la diffusion.
  - Chiffrer des octets en RSA : chiffre mono alphabétique.
- **7) Ne pas utiliser une clé secrète trop courte**
  - Facilite une recherche exhaustive
- **8) Par contre la clé de chiffrement publique peut être courte:** simplification des calculs pour l'émetteur.
- **9) Ne pas chiffrer ou signer** un message proposé par un tiers sans ajouter quelques bits pour le modifier.

RSA : Sécurité
Longueur de n = pq

- **Attaque principale du RSA:** factoriser n = pq
- **Méthodes connues de factorisation :** toutes très lentes et très dépendantes de la longueur des clés. Performance actuelle de factorisation avec beaucoup de moyens 400 -> 500 bits.
- **Longueur de clés classiques**: 320 , 512 , 768 bits.
- **Actuellement** : 1024 bits est considéré comme un minimum pour assurer une bonne sécurité.
- **Conseil pour des applications très durables et de grande sécurité** : 2048 bits ou même 4096 bits.
- **Problème des temps de calcul** : sur des processeurs de faible puissance (cartes à puces)
- **En résumé: utilisation de longueurs de clés de plus en plus importantes** avec l'accroissement de la puissance de calcul qui permet la factorisation de nombres de plus en plus grands.

RSA : Conclusion

- **1) Performances** : Problème principal, lenteur de la méthode en chiffrement déchiffrement.
- **2) Solution pour l'utilisation de RSA : chiffrer de faibles quantité de données**
  - En confidentialité : utiliser le RSA pour échanger des clés secrètes de session.
  - En intégrité : signer des résumés de messages.
  - En authentification : chiffrer des nonces.
- **3) Sécurité** :
  - **Le chiffre est considéré comme sûr** si l'on respecte les contraintes diverses d'utilisation.
  - **Tant qu'on ne trouve pas de solution** efficace au problème de factorisation.

Chiffres à clés publiques

Le chiffre El Gamal

Introduction : le logarithme discret

Méthodes de chiffrement et de déchiffrement

Chiffre à clés publiques El Gamal : Introduction

- **Utiliser la complexité algorithmique** d'un problème

  Ici le calcul d'un logarithme dans les entiers modulo p (comme dans la méthode d'échange de clés de Diffie et Hellman)
- **Problème du logarithme discret** : pour lequel on ne connaît pas d'algorithme efficace.
  - Ensemble G des entiers modulo p (p éléments, groupe multiplicatif)
  - Soient a et b deux éléments: logarithme discret de a en base b
  - Problème trouver x tel que $b^x = a \pmod p$
- **Construction d'un chiffre à clés publiques:** Taher El Gamal

  '*A public-key cryptosystem and a signature scheme based on discrete logarithms*' IEEE Transactions on Information Theory , 31:469,472,1985.
- **Solution non brevetée.**

Chiffre à clés publiques El Gamal : Le choix des clés

- **1) Choisir un grand nombre entier premier**: p
  - Pour faire un ensemble modulo p (groupe multiplicatif)
- **2) Choisir deux nombres entiers imprévisibles** inférieurs à p : g , x
  - **g (générateur)**
  - **x (future clé secrète) :** aléatoire, imprévisible
- **3) Calculer $y = g^x \pmod p$**

- **4) Clé publique : g , p , y**
- **5) Clé secrète : x , p** difficulté du logarithme discret

Chiffre à clés publiques El Gamal : Chiffrement et déchiffrement

Chiffrement

- **A partir de la clé publique :** les entiers p, g, y.
- **Choisir k entier:** k et p-1 premiers entre eux (pgcd(k,p-1)=1)
- **Chiffrer M (taille inférieure à p-1): un couple de deux entiers a,b**
  - **$a = g^k \pmod p$**
  - **$b = y^k M \pmod p$**
- **Avantage :** Le chiffrement dépend en plus d'une valeur k aléatoire: si l'on chiffre deux fois la même chose on a deux chiffres différents.
- **Inconvénient :** la taille du chiffre est le double de la taille du message en clair.

Déchiffrement

- **Principe général** (clé x , p ; chiffre a , b) : $M = b/a^x \pmod p$
  - **Parce que $y = g^x \pmod p$ donc $b = g^{kx} M \pmod p$**
  - **Et $a^x = g^{kx} \pmod p$**

Chiffre à clés publiques El Gamal : Conclusion

- **Pas utilisé en pratique en tant que chiffre.**
  - Doublement du volume des données chiffrées.
  - Complexité des calculs à effectuer.
- **Utilisé dans le cadre de la méthode de signature numérique DSA**
  - Méthode répandue aux USA.

La cryptographie : Chapitre III

Fonctions de hachage sécuritaires (cryptographiques)

Introduction

MD5

SHA

Introduction Fonctions de hachage: Définition de base d'un hachage

- **1)** Une fonction de **hachage** est une fonction H qui à une **entrée de taille quelconque M** (sur m bits , m quelconque) fait correspondre une **sortie de taille fixe H(M) (n bits).**
  - **Idée de compression** : le hachage est plus court que la donnée.
  - **Idée de simplicité de calcul** : le calcul d'un hachage doit être rapide => calcul direct, vérification fréquente.
- **2) Exemples de hachages**:
  - Parité longitudinale, code polynomial.
  - Nombreuses fonctions étudiées des les années 1960 (Donald Knuth) : itérations d'une fonction sur M découpé en blocs de n bits.
- **3) Terminologie**: hachage, résumé, contraction, empreinte, "digest", "hash code"....

Introduction aux fonctions de hachage: Utilisation des hachages

- **1) Emploi des fonctions de hachage selon la taille**
  - Hachages **généraux**: m bits arbitraire sur n bits fixes.
  - Hachage **compressifs**: m bits fixes vers n bits fixes ($n \leq m$).
    - En général on ne hache que des données de taille au moins égale au hachage (sinon on rajoute un bourrage).
  - Hachage **sans compression**: m=n fixe (utilisation voisine d'un chiffre).
- **2) Utilisation classique des hachages :** génération de clés pour caractériser des données (obtention d'une adresse à partir d'une clé)**.**
  - **Eviter les collisions** : deux enregistrements ont la même clé.
  - **Dans un usage d'indexation** : possibilité de gérer les débordements.
- **3) Utilisation des hachages en sécurité:**
  - **Confirmation de connaissance:** vérification d'intégrité (signatures) , stockage de mots de passe**.**
  - **Génération de suites pseudo aléatoires.**
  - **Génération de clés, de mots de passe.**

  => Besoin de propriétés supplémentaires : notion de hachages sécuritaires ou cryptographiques.

Fonctions de hachage: Propriétés de base

- **1) Un hachage compressif ne peut être injectif**
  - Tous les **H(M) ne peuvent être différents**
  - Image d'un ensemble de taille arbitraire de messages M dans un ensemble de taille limitée (n bits): **il existe une infinité de M qui ont le même H(M).**
- **2)** Un hachage devrait être **surjectif** (surjection : toutes les configurations à n bits sont atteintes)
  - **Vue théorique**: comme on part d'un nombre infini de M => **on doit atteindre toutes les images possibles.**
  - **Vue pratique**: pour une application donnée on ne génère qu'un nombre fini et assez petit de messages différents **=> le nombre des hachages différents générés réellement est très petit par rapport au nombre des hachages possibles.**
- **3) Définition informelle** (Schneier/Ferguson): une bonne fonction de hachage doit réaliser **un mappage 'aléatoire' de l'ensemble des entrées sur l'ensemble des sorties.**
  - **Aléatoire :** Equi-répartition, imprévisibilité des valeurs atteintes.

Fonctions de hachages sécuritaires : Propriété a) Hachage à sens unique

- **1) Anglais : OWHF 'O**ne **W**ay **H**ash **F**unction'

  'Pre-image resistance' : qui résiste à l'obtention d'une pré image.
- **2) Utilisation de la notion de sens unique** dans le domaine **des fonctions de hachage** (des fonctions de compression).
- **3) Définition : sens unique pour un hachage**
  - Etant donné une valeur de n bits **a** (un hachage) **il est difficile (c'est un problème très difficile, de complexité élevée)** de trouver **un message M** (m bits) tel que **H(M)= a.**
- **4) Une utilisation typique:** codage de mots de passe pour transmission réseau ou stockage sur disque et vérification ultérieure.

Fonctions de hachage sécuritaires : Propriété b) : Collisions faibles difficiles

- **1) 'Weak Collision Resistance' 'Second Pre-image resistance'**
- **2) Définition:** Il est difficile (de complexité algorithmique très élevée) de trouver **un second message M' différent de M** ayant le même hachage que M c'est à dire **H(M')=H(M).**
- **3) Utilisation en intégrité :**
  - Pour un message valide M , a=H(M)
  - Un intrus ne peut **remplacer le message M**
  - **par un autre qu'il a pu forger M'** tel que H(M') = a
  - pour **violer l'intégrité** (perturber les échanges).

Fonctions de hachage sécuritaires : Propriété c) : Collisions fortes difficiles

- **1) CRHF** 'Collision Resistance Hash Function' 'Strong Collision Resistance'
- **2) Définition:** il est **difficile (complexité algorithmique)** de générer des collisions c'est-à-dire de construire **deux messages différents quelconques M et M' tels que H(M)=H(M').**
- **3)** Fonctions de hachage **recherchées en sécurité**
  - **Attaque** typiquement réalisée: **recherche de collisions.**
- **4)** La construction d'une fonction de hachage à collisions fortes serait **plus difficile qu'un chiffre => calcul plus long.**

Fonctions de hachage sécuritaires : Collisions fortes difficiles (2)

- **Utilisation en intégrité:**
  - Propriété qui empêche un intrus qui a forgé deux messages **M et M' avec même hachage**
  - de **faire signer l'un anodin (M**) par une personne autorisée
  - puis de **remplacer M par M'** avec une signature parfaite.
- **Solution :** si on a des doutes sur le fait que H soit résistant aux collisions rajouter toujours avant de signer un peu d'infos.

Fonctions de hachage sécuritaires : Avec ou sans clé (MDC ou MAC)

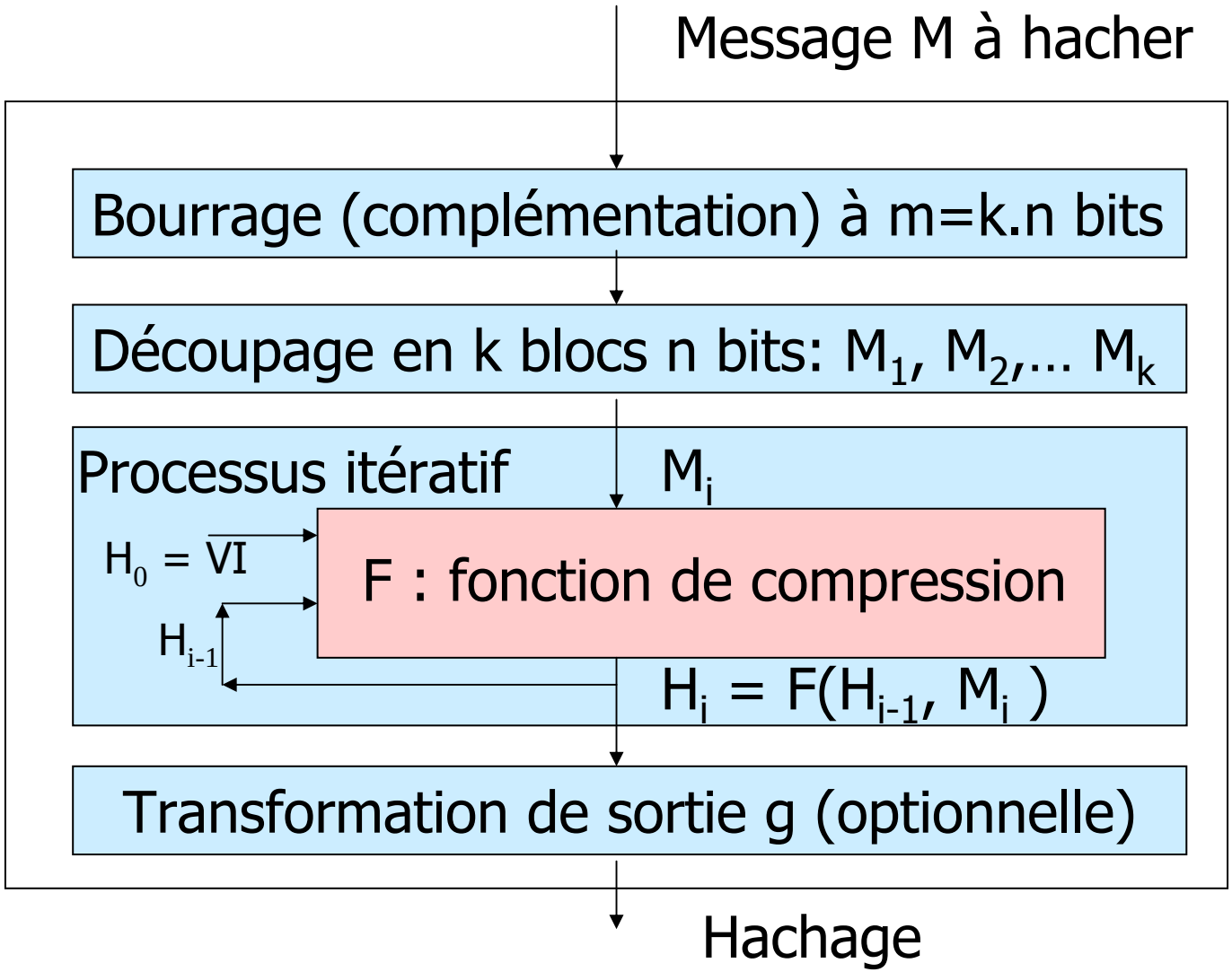
- **1) Fonction de hachage sécuritaire sans clé: MDC**
  'Message Digest Code' => Empreinte de message.
  - **Une fonction de hachage H qui peut être calculée sans connaissance d'un secret.**
  - H est uniquement définie par un **algorithme public.**
  - Exemples type: MD4, MD5, SHA etc...

- **2) Fonction de hachage sécuritaire avec clé: MAC**
  'Message Authentication Code' => Authentificateur de message
  - **Une fonction de hachage qui ne peut être calculée que par une entité détentrice d'un secret K.**
  - Exemple type: **MAC(M) = H (K , M)** ou H est un MDC sur la concaténation d'un secret K et du message en clair M.
  - Nombreuses fonctions de hachage avec clé (déduites de hachages ou de chiffres)

Fonctions de hachages sécuritaires : Renforcement d'une fonction sans clé

- **1) H étant donné : Renforcer H** par une application multiple du même hachage.
- **2) Proposition d : $H_d = H (H(M))$ Double hachage**
  - On ne peut faire **d'extension de longueur de M**.
  - **Pas beaucoup plus lent** qu'un hachage simple.
  - Considéré comme **un peu plus sécuritaire que H.**
- **3) Proposition dbl $H_{dbl} = H (H(M) , M)$**
  - Le hachage agit des le départ sur une initialisation H(M) qui représente tous les bits du message.
  - On ne peut faire **d'extension de longueur** de M.
  - **Deux fois plus lent** qu'un hachage simple.
  - Considéré comme **nettement plus sécuritaire que H**.

Fonctions de hachage sécuritaires : Hachage itéré (Merckle)



Fonctions de hachage sécuritaires : Choix d'une fonction de compression

- **Solution 1 : Définir une fonction complètement nouvelle** dont on prouve qu'elle a de bonnes propriétés relativement aux collisions.
- **Solution 2 : Utiliser directement un chiffre à clés secrètes par blocs** $E_K(X)$ si l'on peut montrer qu'il a de bonnes propriétés relativement aux collisions
  - $f(K, X) = E_K(X)$ (2 n bits -> n bits)
- **Solution 3 : Si l'on a pas de bonnes propriétés de collisions essayer d'améliorer le chiffre par blocs par des modifications:** Quelques propositions possibles
  - $f(K, X) = E_K(X) \oplus X$
  - $f(K, X) = E_K(X) \oplus X \oplus K$
  - $f(K, X) = E_K(X \oplus K) \oplus X$
  - $f(K, X) = E_K(X \oplus K) \oplus X \oplus K$

Fonctions de hachage sécuritaires

MD5
‘Message Digest N°5’

MD5 : Introduction

- **1) Conçu par Ronald Rivest**: publication avril 1992
- **2) Architecture de Merckle.**
- **3) Série de hachages MD1 à MD4**: corrige de plus en plus de défauts

  => **MD5** une amélioration de **MD4**.
- **4) Génère un hachage:** 128 bits.
- **5) IETF:** RFC 1321.
- **6) Algorithme de hachage très souvent utilisé** avant d'être considéré comme **faible** : possibilité de **générer des collisions** en assez peu d'opérations.

MD5 : Algorithme (1)

- **1) Calcul itératif du résultat de 128 bits** sous la forme de quatre mots de 32 bits a, b, c, d initialisés au départ à des constantes:

  a = 01234567 b = 89ABCDEF c = FEDCBA98 d = 76543210
- **2) Un message est décomposé en blocs de 512 bits** soient 16 sous-blocs de 32 bits notés Mj: un message est complété à un nombre entier de blocs de 512 bits par bourrage.
- **3) Pour chaque bloc de 512 bits** on réalise un calcul à partir de la dernière valeur de a,b,c,d -> nouvelle valeur a,b,c,d.
- **4) Définition des fonctions** F, G, H, I : elles sont non linéaires sur 32 bits à partir de ou, et, non, ouex (ou exclusif).

$$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$

$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$

$$H(X,Y,Z) = (X \oplus Y \oplus Z)$$

$$I(X,Y,Z) = Y \oplus (X \vee \neg Z)$$

MD5 : Algorithme (2)

- **5)** Le calcul comprend **4 rondes** de **16 applications successives de quatre fonctions FF, GG , HH, II**
- **6)** Ces fonctions dépendent des fonctions **F, G, H, I,** des sous-blocs **Mj,** des variables **a, b, c, d,** et de constantes **ti.**
- **7)** On utilise deux opérateurs:
  - << s dénote **une rotation à gauche de s positions.**
  - \+ est **l'addition modulo 2^{32}**

FF (a , b , c , d , Mj , s , ti) : a = b + ((a + F(b,c,d) + Mj + T[i]) << s)
GG(a , b , c , d , Mj , s , ti) : a = b + ((a + G(b,c,d) + Mj + T[i]) << s)
HH(a , b , c , d , Mj , s , ti) : a = b + ((a + H(b,c,d) + Mj + T[i]) << s)
II (a , b , c , d , Mj , s , ti) : a = b + ((a + I(b,c,d) + Mj + T[i]) << s)

MD5 : Algorithme (3)

- **La première ronde:**

FF (a,b,c,d, M(0),7, D76AA478)
FF (d,a,b,c, M(1),12, E8C7B756)
FF (c,d,a,b, M(2),17, 242070DB)
FF (b,c,d,a, M(3),22, C1BDCEEE)
FF (a,b,c,d, M(4),7, F757C0FAF)
FF (d,a,b,c, M(5),12, 4787C62A)
FF (c,d,a,b, M(6),17, A8304613)
FF (b,c,d,a, M(7),22, FD469501)
FF (a,b,c,d, M(8),7, 698098D8)
FF (d,a,b,c, M(9),12, 8B44F7AF)
FF (c,d,a,b, M(10),17, FFFF5BB1)
FF (b,c,d,a, M(11),22, 895CD7BE)
FF (a,b,c,d, M(12),7, 6B901122)
FF (d,a,b,c, M(13),12, FD987193)
FF (c,d,a,b, M(14),17, A679438E)
FF (b,c,d,a, M(15),22, 49B40821)

- **Quatre rondes analogues: basées sur les fonctions FF, GG, HH, II.**
- **Les constantes utilisées changent selon les rondes.**

MD5 : Conclusion

- **1)** En 1996, **une attaque réussie**: H. Dobbertin a montré qu'on pourrait **obtenir des collisions** sur MD5.

- **2)** Depuis MD5 est **considéré comme inutilisable pour des applications de haute sécurité:** attaque complète réussie à l'été 2004 (collisions trouvées).

- **3)** MD5 reste utilisé dans **de nombreuses applications** ou la sécurité n'a pas à être de haut niveau
  - Exemple: transport chiffré des mots de passe utilisé par des FAI pour autoriser l'accès au réseau Internet.

Fonctions de hachage sécuritaires

SHA
‘Secure Hash Algorithm’

SHA : 'Secure Hash Algorithm' Introduction

- **1) Conçu par la NSA** normalisé par le **NIST**: nom de la norme SHS 'Secure Hash Standard' (1993-1995).
- **2) Version SHA-0** : corrigée assez vite en **SHA-1**.
- **3) IETF:** RFC 3174
- **4) SHA :** une amélioration de MD4.
- **5) SHA-1 :** des hachages de 160 bits

  => Meilleure longueur que MD5 (128 bits)
- **6) Algorithme préféré pour le hachage** (après MD5) : avant de montrer des faiblesses.
- **7) Trois versions plus évoluées**: versions appelées **SHA-256, SHA-384, SHA-512** (2002) (SHA-2)

SHA-1 : Algorithme (1)

- **1) Architecture de Merckle.**
- **2) Ressemblances avec MD4 et MD5.**
- **3) Travail sur des blocs de 512 bits** : bourrage jusqu'à 448 bits dans le dernier bloc et concaténation de la longueur des données utiles sur 64 bits.
- **4) Résultat** dans cinq variables de 32 bits **=> 160 bits** A,B,C,D,E (valeurs initialisées au départ par des constantes).
- **5) Quatre rondes de 20 itérations** basées sur quatre fonctions f_1 , f_2 , f_3 , f_4 .

$f_1(B,C,D) = f_t(B,C,D) = (B \wedge C) \vee ((\neg B) \wedge D)$ $\quad (0 <= t <= 19)$

$f_2(B,C,D) = f_t(B,C,D) = B \oplus C \oplus D$ $\quad (20 <= t <= 39)$

$f_3(B,C,D) = f_t(B,C,D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$ $\quad (40 <= t <= 59)$

$f_4(B,C,D) = f_t(B,C,D) = B \oplus C \oplus D$ $\quad (60 <= t <= 79)$

SHA-1 : Algorithme (2)

- **Utilisation de 80 constantes additives** K_t sur 32 bits
  - K_t = 5A827999 (0 <= t <= 19)
  - K_t = 6ED9EBA1 (20 <= t <= 39)
  - K_t = 8F1BBCDC (40 <= t <= 59)
  - K_t = CA62C1D6 (60 <= t <= 79)

- **Utilisation de 5 constantes d'initialisation**
  - H_0 = 67452301
  - H_1 = EFCDAB89
  - H_2 = 98BADCFE
  - H_3 = 10325476
  - H_4 = C3D2E1F0

SHA-1 : Algorithme (3)

1 **Commentaire :** Découper M_i en 16 mots de 32 bits W_0, W_1, …. , W_{15} puis initialiser les 80 Wt

2 **Pour** t = 16 **jusqu'à** 79 **faire**

3 $W_t := W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16} << 1$;

4 **Finpour** ;

5 **Commentaire :** Initialiser A , B , C , D , E (valeur initiale ou valeur résultant du précédent bloc)

6 $A := H_0$; $B := H_1$; $C := H_2$; $D := H_3$; $E := H_4$;

7 **Commentaire :** Calcul principal

8 **Pour** t = 0 **jusqu'à** 79 **faire**

9 $TEMP := (A << 5) + f_t(B,C,D) + E + W_t + K_t$;

10 $E := D$; $D := C$; $C := (B << 30)$; $B := A$; $A := TEMP$;

11 **Finpour** ;

12 **Commentaire :** présentation du résultat

13 $H_0 := H_0 + A$; $H_1 := H_1 + B$; $H_2 := H_2 + C$;
$H_3 := H_3 + D$; $H_4 := H_4 + E$;

Secure Hash Algorithm: SHA-256 , SHA-384 , SHA-512

- **1) Norme NIST FIPS 180-2 :** pour allonger les hachages produits par SHA-1 (2002).
- **2) Utilisation d'idées similaires**: adaptation de SHA-1 mais avec plus de sécurité.
- **3) Conçu pour fonctionner avec AES**
  - Clés de 128, 192, 256 bits
- **4) Trois versions sur 256 , 384 , 512 bits** (en plus du SHA-1 qui reste normalisé).
  - **SHA-384 a peu d'intérêt** (encombrement moindre dans les messages) mais aussi long à calculer que SHA-512.
  - **SHA-256 aussi long à calculer que AES.**

SHA : Conclusion

- **1) SHA-1 : attaque force brute complète 2**80 opérations** (n/2 selon la longueur de hachage 160 bits).

- **4) Février 2005** : **obtention de collisions sur SHA-1 en 2**69** opérations (Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu Université de Shandong Chine) par des techniques déjà connues auparavant (été 2005 2**63 opérations).

- **3) SHA-1** va être considéré comme **trop faible pour des applications de haute sécurité =>** Utilisation de SHA-1 ramenée à des **applications de base**.

- **4) Conseil d'utilisation** : pour des applications de haute sécurité **SHA-256, SHA-384 ou SHA-512** éventuellement **en mode SHA_d** ou **SHA_{dbl}** (double application).

Fonctions de hachage : Conclusion

- **1) Problème d'intégrité d'importance croissante** (par rapport à la confidentialité).
- **2) Beaucoup moins d'études et d'attaques** sur les hachages que sur les chiffres.
- **3) Les propriétés des hachages sécuritaires** (collisions) **sont plus difficiles** à satisfaire que les propriétés des **chiffres**.
- **4) Il faudra accepter que les hachages aient des temps de calcul longs** ce qui n'était pas le cas avec MD5 ou SHA1

Générateurs de nombres aléatoires cryptographiques

Introduction

Générateurs de nombres aléatoires d'origine physique

Générateurs de nombres pseudo-aléatoires

Générateurs de nombres aléatoires : Introduction

Besoins de nombres aléatoires en sécurité (à différents niveaux)

- **1) Génération des clés pour les chiffres par blocs à clés secrètes** : changement fréquent des clés.
- **2) Génération des clés pour les chiffres en continu**: une clé nouvelle par bloc (mode OFB 'Output Feedback Block').
- **3) Génération des nonces** : nombres aléatoires ne prenant pas la même valeur (éviter le rejeu, authentifier sur un défi).

Problème des attaques sur les générateurs : si l'on peut prévoir la clé, un chiffre ne sert à rien.

Générateurs de nombres aléatoires : Propriétés statistiques

- **Approche classique** de la définition de l'aléatoire.
- **La distribution des nombres générés est uniforme:**

$$\text{Prob}(X_n = a) = 1/N = \text{Prob}(X_n = a / X_1, X_2, \ldots X_{n-1})$$

- **Autre formulation:** une suite aléatoire ne peut être comprimée **=> son entropie est maximale.**
- **Par extension :** sur une suite générée **uniformité** des distributions des couples, triplets, groupes de n nombres successifs ….
- **Problème : Une suite peut posséder de bonnes propriétés statistiques tout en étant 'prévisible' :** l'effort pour deviner le nombre suivant généré est faible.

Générateurs de nombres aléatoires : Propriétés de complexité

- **Approche récente de la définition de ce qui est aléatoire par la complexité**
- **Complexité de Kolmogorov d'une suite aléatoire finie**: approche de complexité par la taille d'un programme qui génère la suite (en instructions d'une machine universelle)
  - Le caractère aléatoire d'une suite est proportionnel à la taille du plus petit programme permettant de générer la suite : plus la suite est aléatoire plus un programme qui peut la produire est grand.
  - Autre formulation : une suite aléatoire est difficile à compresser.
- **Complexité dans les générateurs de nombres aléatoires sécuritaires : clés, nonces …**
  - **Les nombres produits doivent être difficiles à prévoir.**
  - **C'est un problème de complexité algorithmique élevée** (grand nombre d'opérations, grand code) de prévoir le prochain nombre généré (même quelques bits ou des aspects de ce nombre).
  - **Même si l'on peut observer une longue suite de nombres déjà générés** par le générateur.

1) Générateurs de nombres aléatoires d'origine physique

- **1) Solution de bonne qualité :** acheter un périphérique spécial de génération de nombres aléatoires .

- **2) Utilisation d'un phénomène physique réputé aléatoire:** observé comme générant des suites aléatoires: typiquement relevant de la physique quantique.

- **3) Choix d'un phénomène produisant de bonnes variables aléatoires mesurables :** (jargon fabrication d'une source d'entropie).
  - **Amplitude d'un bruit thermique** de résistance (variance).
  - **Nombre de photons** qui franchissent une barrière.
  - **Nombre de particules** émises par radio activité (+- dangereux).

Générateurs de nombres aléatoires physiques : Périphérique spéciaux

- **1) Construction d'un périphérique de comptage** et connexion sur une interface standard d'ordinateur (typiquement USB).
- **2) Protection du dispositif physique** contre les attaques
  - Eviter que des influences externes (par exemple un pirate exerçant un champ magnétique)
  - ne restreigne le caractère aléatoire du phénomène physique.
- **3) Protection de la connexion avec l'ordinateur** pour que les données aléatoires ne puissent être observées: elles doivent rester imprévisibles.

2) Nombres aléatoires d'origine physique : Périphériques habituels d'un ordinateur

- **1) Source d'entropie: un périphérique habituel d'un calculateur**
  - **Clavier :** Caractéristiques de la frappe: délais, caractères utilisés
  - **Souris :** Amplitude du déplacement, vitesse, délais entre clics.
  - **Disques** : Caractéristiques des accès disques positionnement des bras et des secteurs, délais entre requêtes.
  - **Microphone** : Utilisée à vide comme source de bruit.
  - **Nombreuses autres possibilités** : affichage, charge processeur …
- **2) Sources imparfaites :** combinaison de sources de nature différentes pour faire un bon générateur aléatoire.
  - Prendre sur chaque événement peu de bits.
  - Exemple mesure d'un intervalle entre événements: on génère un bit si deux intervalles successifs sont plus ou moins grands.
- **3) Absence de sources au démarrage** (peu de durée d'observation d'un phénomène) : constituer lors de chaque session un fichier utilisable en début de session suivante.

3) Générateurs de nombres pseudo-aléatoires GNPA : Introduction

- **1) PRNG : Pseudo Random Number Generator.**
- **2) Générateurs GNPA classiques** (D. Knuth 'The art of computer programming' Vol 2)
  - **Exemple type :** les générateurs basés sur des modulo (congruences) Bonnes propriétés statistiques
  - Récurrence d'ordre 1 linéaire

$$X_n = a\,X_{n-1} + b \pmod m$$

  - Récurrence d'ordre 1 polynomiale (variantes multiples)

$$X_n = a\,X^2_{n-1} + b\,X_{n-1} + c \pmod m$$

  - Utilisation **déconseillée** en sécurité car les suites produites ne sont pas imprévisibles.

Générateurs de nombres pseudo-aléatoires : GNPA sécuritaires

- **1) Impossibilité** de générer des suites de nombres aléatoires de longueur arbitraire par programme: cyclicité des suites.
- **2) Solution adoptée** : partir de nombres aléatoires imprévisibles c'est-à-dire issus d'un phénomène physique (notion de **graine 'seed'**).
- **3) Appliquer une transformation:** par une fonction CRYPTOGRAPHIQUE qui génère à partir de la graine des nombres aléatoires aussi imprévisibles que possible, avec un cycle le plus long possible.
- **4) Exemple d'un GNPA:** à base d'une suite récurrente double d'ordre 1

X_0 , S_0 graine (seed) **X_0 ne doit pas être utilisé**

$X_n = f(X_{n-1}, S_{n-1})$ **Le nombre généré**

$S_n = g(X_{n-1}, S_{n-1})$ **La graine pour le prochain nombre.**

Générateurs de nombres pseudo-aléatoires GNPA : Classes de solutions

- **1) Utiliser un chiffre à clé publique, Exemple : générateur basé sur RSA**
  - Choix de n=pq grand, e une clé, X_0 une graine
  - $X_n = (X_{n-1})^e \bmod n$
  - Prédire le prochain nombre est équivalent à casser RSA.
  - Solution lente
- **2) Utiliser un chiffre à clé privée**
  - Nombreuses variantes utilisant un chiffrement par blocs.
  - Typiquement en mode CBC (chaînage sur la précédente valeur).
  - Norme ANSI X9-T17 Choisir une clé DES k et une graine X_0

    $S_n = DES_k ((DES_k(T_n), X_{n-1}))$ Tn l'heure courante

    $X_n = DES_k ((DES_k(T_n), S_n))$
- **3) Utiliser une fonction de hachage**
  - On part d'une graine X0
  - On applique une fonction de hachage itérativement
  - $X_n = H (X_{n-1})$

Générateurs de nombres pseudo-aléatoires GNPA : Exemple (Schneier)

- **Graine :** obtenue à partir d'un générateur de nombre aléatoires d'origine physique.
- **Exemple type de générateur actuel: utilisation d'un chiffrement par blocs en mode compteur**
  - Exemple de choix d'un chiffre par blocs: **AES Rijndael**.
  - **Notion d'état du générateur:** les variables utilisées pour générer chaque nombre.
    - Clé secrète K (la graine) : 256 bits
    - Compteur cpt (un numéro de séquence) : 128 bits
  - **Xi = AES (K , cpt) sur 256 bits :** on conserve 128 bits
  - **Ne pas générer trop de nombres** à partir de la même clé: sur 128 bits on risque une collision avec probabilité ½ en moyenne après 2^{64} nombres générés.
  - **Générer seulement des suites de 2^{16}** nombres à partir d'une graine amène une probabilité de collision en 2^{-100}

Cryptographie :

Conclusion

Cryptographie : domaine en évolution constante

- **1) Des fonctions sécuritaires aux propriétés difficiles à satisfaire:**
  - Les chiffres : complexité des inverse,
  - Les hachages : complexité de la génération des collisions,
  - Les générateurs de nombres aléatoires : complexité des suites générées
- **2) Domaine basé essentiellement sur la théorie de l'information et de la complexité algorithmique.**
- **3) Fonctions à redéfinir en permanence.**
  - En relation avec la progression des attaques et de leurs parades.
  - En relation avec l'accroissement exponentiel de la puissance de calcul disponible qui nécessite l'augmentation des tailles.
- **4) Conséquence : il faut voir la cryptographie comme un processus plus que comme des solutions toutes faites (B Schneier).**

Bibliographie

- **Bruce Schneier**, Cryptographie appliquée , Thomson Publishing, Paris 1995
- **Bruno Martin**, Codage , cryptologie et applications, Presses polytechniques et universitaires romandes 2004
- **Stéphane Natkin**, Les protocoles de sécurité de l'Internet, Dunod, 2002
- **Niels Ferguson , Bruce Schneier**, Cryptographie en pratique , Wiley 2003, Vuibert 2004, Paris 1995
- **A Menezes, P Van Oorschot, S Vanstone**, Handbook of applied cryptography, CRC Press Inc, 1997
- Très nombreux cours disponibles en ligne et pages spécialisées sur des concepts, normes ou produits

Exemple Cours "Cryptographie et Sécurité" http://cui.unige.ch/tcs/cours/crypto/ **José Rolim, Frédéric Schutz**