Data Encryption Standard

- Années 1970 Lucifer d'IBM, standard américain FIPS 46-2
  - Substitutions, Transpositions, Fonction de Feistel
  - Chiffrement par blocs de 64 bits
  - Clefs de 64 bits, 1 bit sur 8 de parité : 56 bits effectifs (plaidé par la NSA)
    - ⇨ Génération de 16 sous-clefs de 48 bits

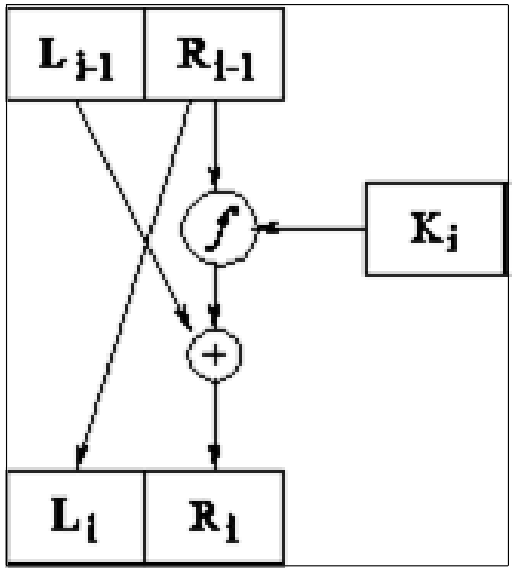
- Structure
  - Permutation initiale
  - 16 itérations de Feistel : expansion, substitution, permutation
    - 16 sous-clefs de 48 bits dérivées de la clef initiale
  - Permutation inverse

- Après 5 tours, chaque bit du chiffré dépend de chaque bit du message en clair et de chaque bit de la clef
- Dès 1995, une puce dédiée chiffre 64 Mo/s

Détail du DES

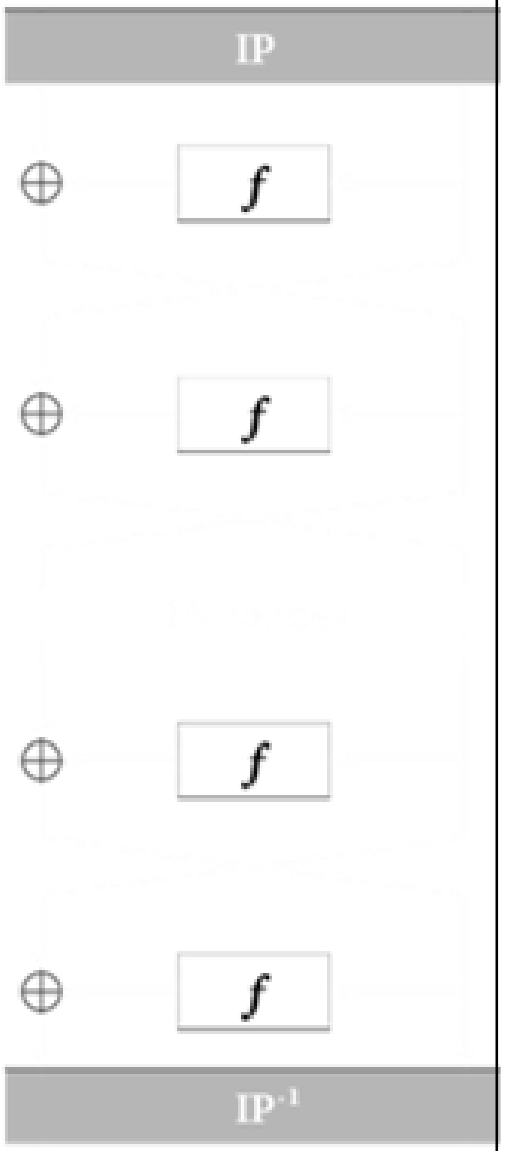
1. $L_0|R_0 = IP(M)$
2. Faire 16 fois

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



3. $C=IP^{-1}(R_{16}|L_{16})$

- Déchiffrer : $C=DES_k(M) \Rightarrow \underline{M}=DES_{\underline{k}}(\underline{C})$



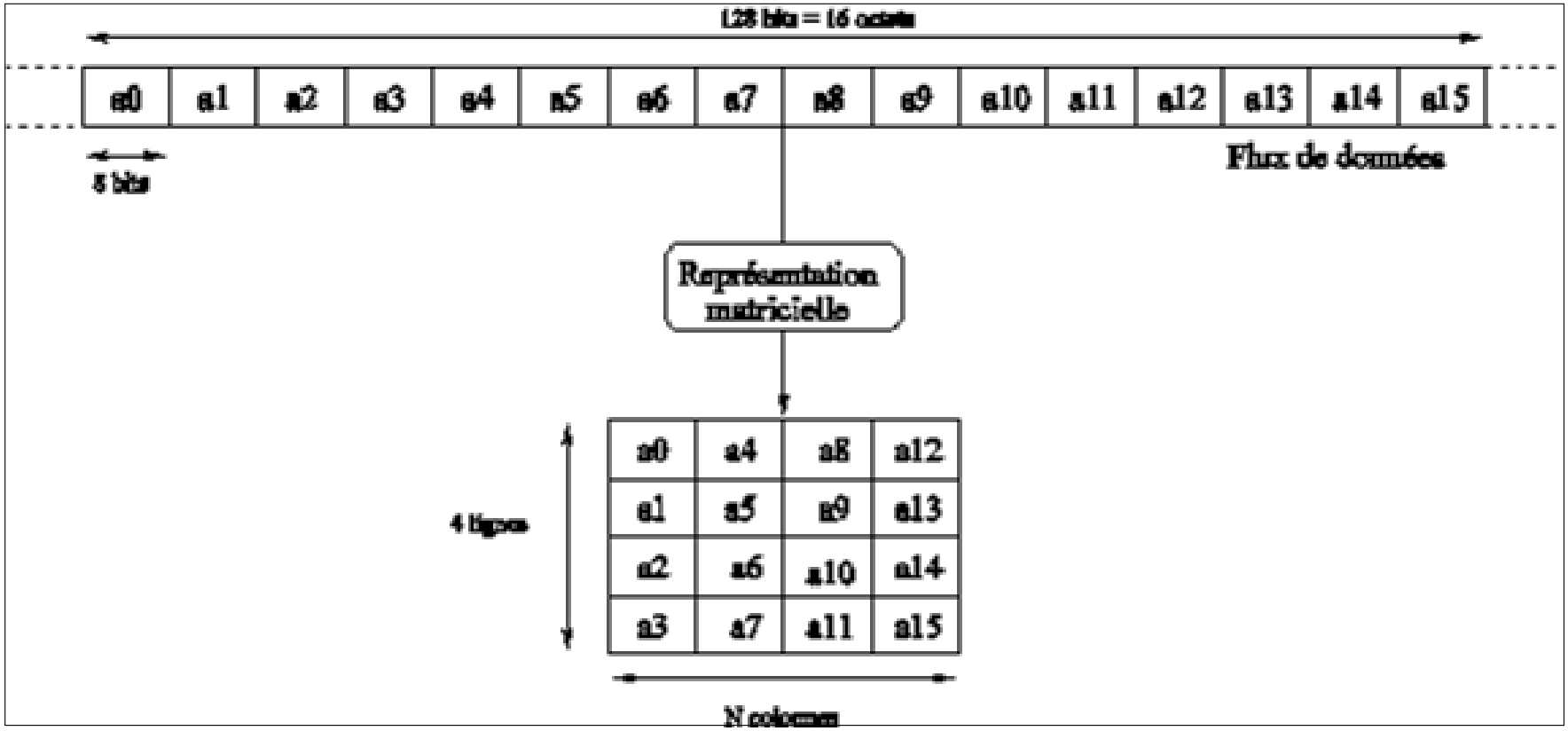
Advanced Encryption Standard

- Processus de sélection
  - Janvier 1997 : appel à candidatures
  - 5 finalistes : Rijndaël, Serpent, Twofish, RC6, Mars
  - Août 2000 : sélection de Rijndaël

Algorithme	Taille de clef	Cycle/octet	
		Cryptage	Décryptage
DES	56	59	59
3-DES	112	154	155
IDEA	128	56	56
Grand cru	128	1250	1518
RC6	256	18	17
Rijndael	256	34	35
Serpent	256	68	80
Mars	256	31	30
Twofish	256	29	25

AES

- Représentation Matricielle d'un flux de 128 bits = 16 octets = 4 blocs



- Configurations (N_k=|clef|, N_b=#blocs, N_r=#tours) $\times$ 32 bits :
- AES-128 (4,4,10) ; AES-192 (6,4,12) ; AES-256 (8,4,14)

Tour de Rijndaël-128

Faire 10 fois

1. AddRoundKey : $\oplus$ bit à bit entre bloc et partie de la clef

Blocs sous forme de matrice (ex: 128 bits => 4x4 octets)
octets considérés comme des éléments du corps fini GF(256)
avec GF(256) $\cong$ Z/2Z[X] modulo $P_8=X^8+X^4+X^3+X+1$

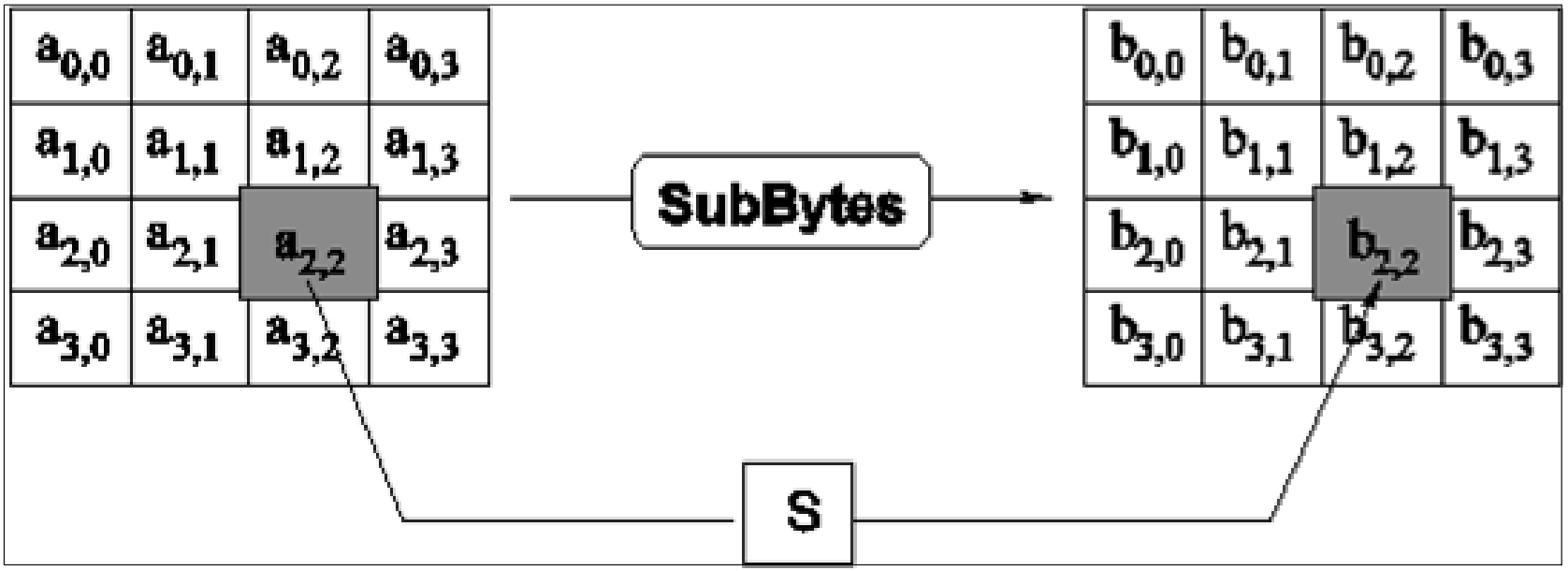
2. SubBytes : pour chaque octet S(a) = A (a^{-1} GF(256)) + C mod 2

3. ShiftRow : chaque ligne est décalée de i

4. MixColumn : brouillage des colonnes (code correcteur linéaire)
   Chaque colonne, polynôme de GF(256)[Y], est multipliée par
   ([03] Y^3 + Y^2 + Y + [02]) modulo (Y^4+[01])

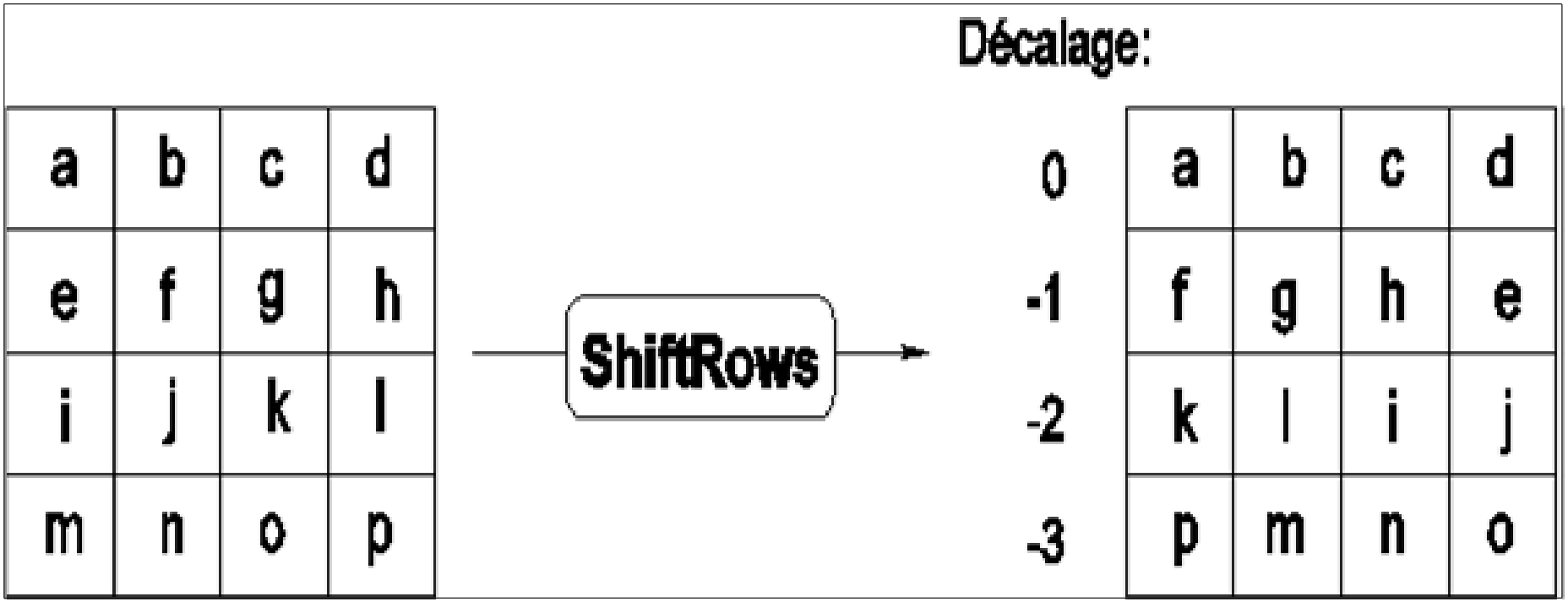
SubBytes

- SubBytes : pour chaque octet
  $S(a) = A\ (a^{-1}\ GF(256)) + C \bmod 2$
  $S(0) = C$



- $b_{i,j} = S(a_{i,j})$

ShiftRows


Décalage:
a b c d
e f g h
i j k l
m n o p
ShiftRows
0
-1
-2
-3
a b c d
f g h e
k l i j
p m n o

MixColumn

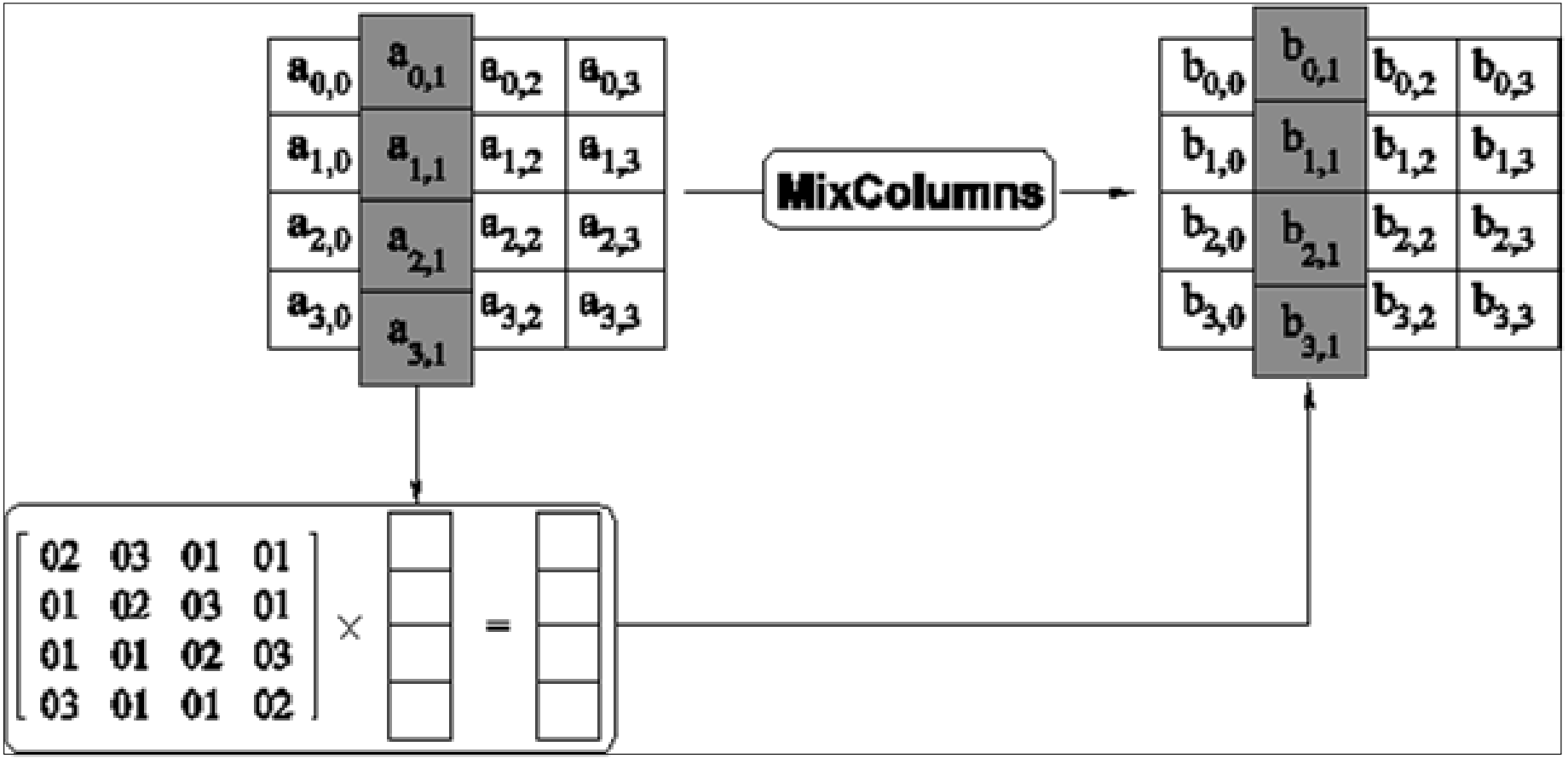
- Chaque colonne C, polynôme de GF(256)[Y], est multipliée par G = ([03] Y^3 + Y^2 + Y + [02]) modulo (Y^4-[01])
- Multiplication de Matrices : A_G C
- A_G=

[2]	[3]	[1]	[1]
[1]	[2]	[3]	[1]
[1]	[1]	[2]	[3]
[3]	[1]	[1]	[2]

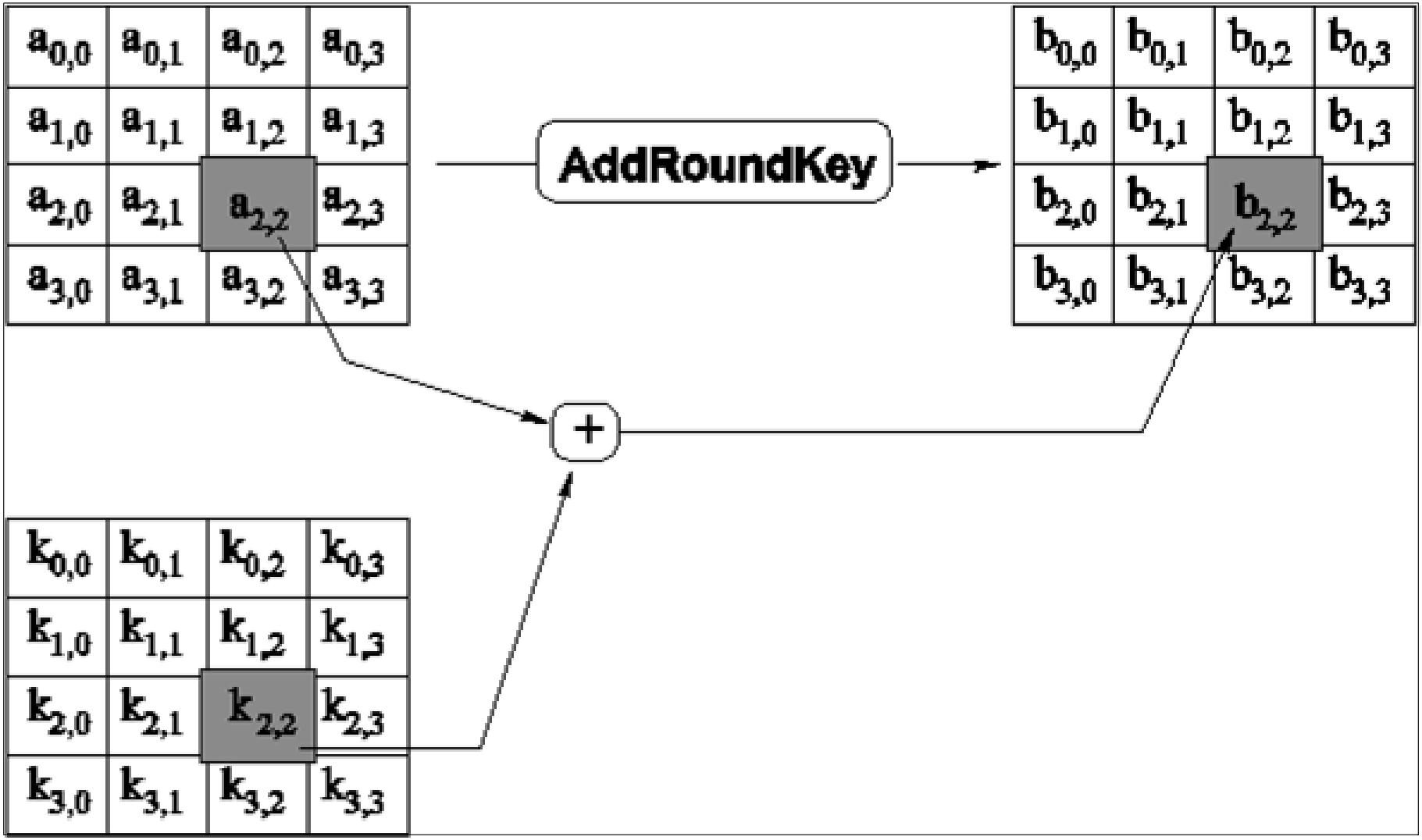
avec les monômes rangés
de haut en bas
par degré croissant

- M = C $\times$ G mod Y^8-[1] : code 2-correcteur d'erreur cyclique sur les colonnes :
  - Détection jusqu'à 4 octets erronés dans M
  - Correction automatique si 2 octets de M sont modifiés
    - Bonne propriétés de diffusion cryptographique
    - M mod Y^4+[1] : addition des 8 octets deux à deux : conservation de la diffusion

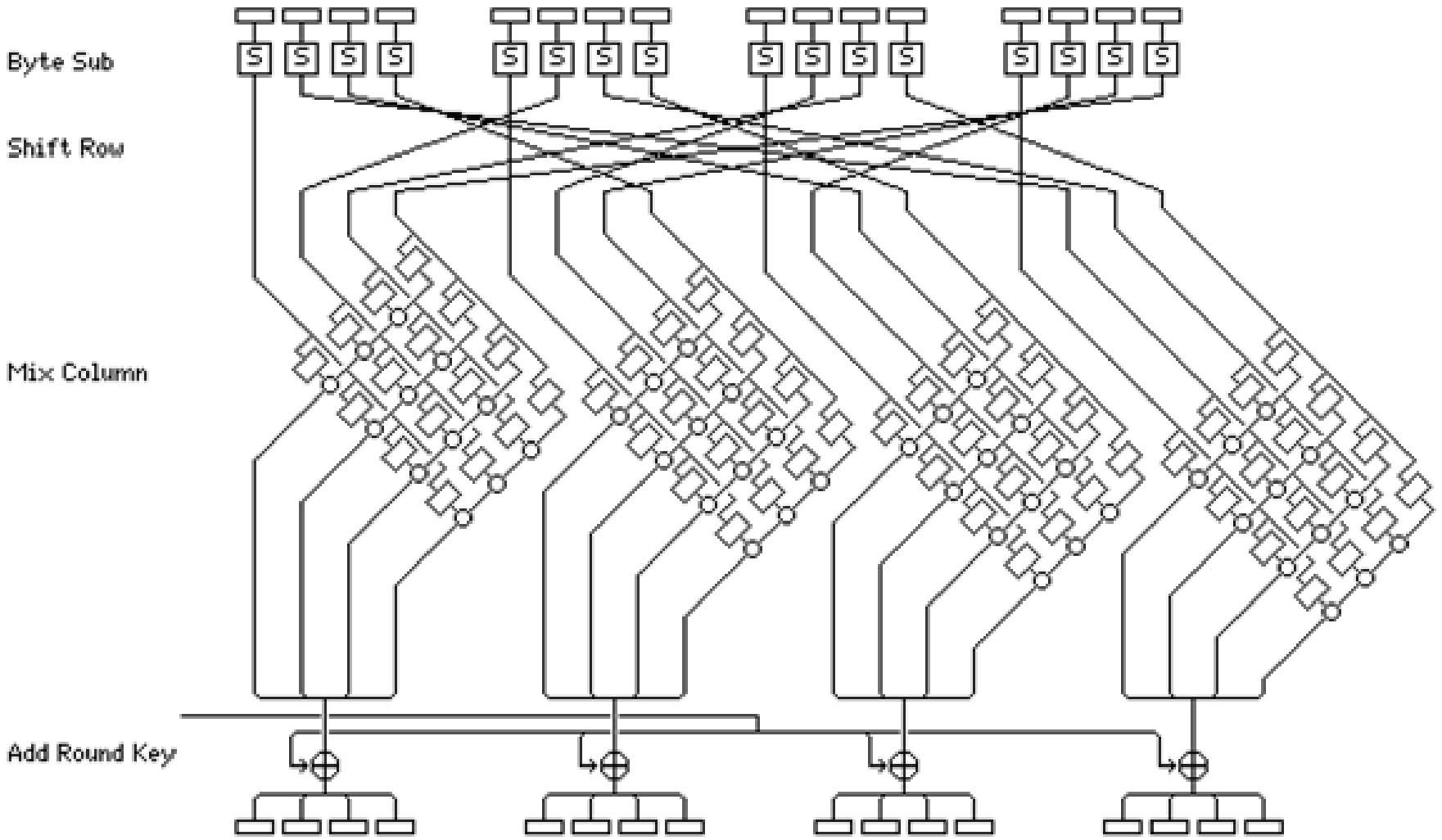
MixColumns



AddRoundKey



Ronde de Rijndaël-128



Pseudo code AES

Cryptage A.E.S. $E_K(M)$

```
A:=M;
KeySchedule(K);
A:=AddRoundKey(A,K_0);
Pour i = 1 à 9 Faire {
    A:=SubBytes(A);
    A:=ShiftRows(A);
    A:=MixColumns(A);
    A:=AddRoundKey(A,K_i);
}
A:=SubBytes(A);
A:=ShiftRows(A);
A:=AddRoundKey(A,K_10);
Return C:=A;
```

Décryptage A.E.S. $D_K(C)$

```
A:=C;
KeySchedule(K);
A:=AddRoundKey(A,K_10);
A:=InverseShiftRows(A);
A:=InverseSubBytes(A);
Pour i = 1 à 9 Faire {
    A:=AddRoundKey(A,K_10−i);
    A:=InverseMixColumns(A);
    A:=InverseShiftRows(A);
    A:=InverseSubBytes(A);
}
A:=AddRoundKey(A,K_0);
Return M:=A;
```

Pseudo code KeySchedule

KeySchedule(K)

$W_0||W_1||W_2||W_3 := K_0$
Pour $i = 1$ à 10 Faire {
    T:=$W_{4i-1} <<< 8$; // décalage circulaire de 8 bits vers la gauche
    T:=SubBytes(T); // 4 appels SubBytes
    T:=T $\oplus(X^i \mod P_8)$; // addition sur chaque octet
    W_{4i}:=$W_{4i-4} \oplus T$;
    W_{4i+1}:=$W_{4i-3} \oplus W_{4i}$;
    W_{4i+2}:=$W_{4i-2} \oplus W_{4i+1}$;
    W_{4i+3}:=$W_{4i-1} \oplus W_{4i+2}$;
    $K_i = W_{4i}||W_{4i+1}||W_{4i+2}||W_{4i+3}$;
}

Sécurité de l'AES

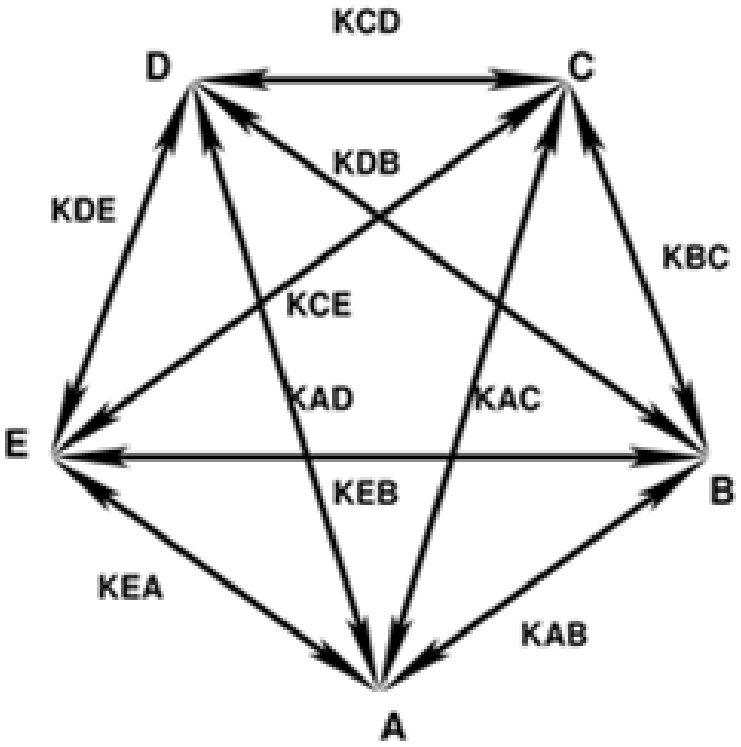
- Propriétés cryptanalytiques
  - S : sans point fixe ni opposé, ni inverse
  - ShiftRow diffuse les données en séparant les consécutifs
  - +MixColumn : chaque bit de sortie dépend de tous les bits en entrée (code correcteur linéaire sur chaque colonne)
- Implémentations FPGA
  - Jusqu'à 21.54 Go/s pour le cryptage *[Hodjat-Verbauwhede 2004]*
- Sécurité
  - Aucune attaque significative révélée
  - [Courtois, Pieprzyk, 02, 05] attaques en $\approx O(2^{100})$ sur AES-128
  - …

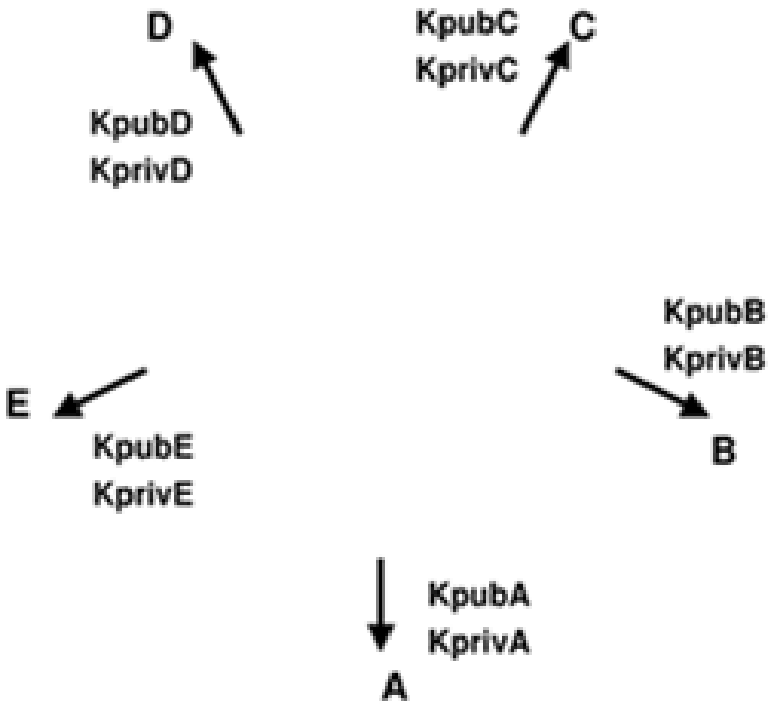
Applications utilisant Rijndaël

- SONET (Synchronous Optical NETwork)
- Routeurs Internet
- Switch Ethernet ATM (Asynchronous Transfert Mode)
- Communications Sattelites
- VPN (Réseaux privés virtuels)
- Téléphonie mobile
- Transactions électroniques

Clef publique / clef privée

- Tout le monde peut envoyer (avec la clef publique)
- Seul le destinataire peut déchiffrer (avec la clef privée)





Entropie et clefs publiques ?

- K_d est secrète, K_e est publique
- Un cryptosystème à clef publique parfait devrait vérifier

$$H(M \mid C=EK_e(M)\ ) = H(M)$$

- Dans un cryptosystème à clef secrète les fonctions de chiffrement et déchiffrement sont inconnues car la clef est secrète
- Or dans un cryptosystème à clef publique le déchiffrement est l'inverse du chiffrement, et le chiffrement est explicite

E_{Ke} est explicite car K_e publique

et $D_{Kd} = E_{Ke}^{-1}$

☹ Du point de vue de la théorie de l'information, on a donc

- ☹ $H(M|C) = 0$ et même $H(M)=0$ si le système pouvait être parfait
- ☹ $H(K_d \mid K_e) = 0$ dans tous les cas

Il n'y a aucun secret sur la clef privée ni sur le message !!!

⇨ La théorie de l'information est insuffisante, l'incertitude sur la clef privée n'est pas pertinente pour mesurer la fiabilité d'un cryptosystème à clef publique, la caractérisation doit plutôt se faire par la *théorie de la complexité*

Théorie de la complexité

- Méthodologie pour analyser la complexité de calcul des algorithmes
  - Complexité en temps (ou en nombre d'opérations)
  - Complexité en mémoire (espace de stockage nécessaire)

- Complexité exprimée comme fonction de la taille du paramètre d'entrée

- Complexité d'un problème : complexité de l'algorithme permettant de résoudre l'instance la plus difficile

- Classification :
  - Problèmes solubles (en temps polynomial) : classe P
  - Problèmes difficiles (solubles en temps exponentiel) : classe NP
  - Problèmes indécidables

Petit théorème de Fermat

- Indicateur d'Euler : $\phi(n)$ nombre de premiers avec n
  → c'est le cardinal de Z/nZ*
- Algorithme :
  - p premier, $\phi(p) = p - 1$, $\phi(p^k) = (p-1)p^{k-1}$
  - m, n premiers entre eux $\phi(mn) = \phi(m)\phi(n)$

- Théorème d'Euler (1707-1783) : $a^{\phi(n)} \equiv 1$ [n], pour a inversible modulo n
  - $G_a = \{y, y=ax, x \in Z/nZ^*\}$ Preuve de Lagrange (1736-1813)
  ➔ $G_a = Z/nZ^*$, car a, x et y inversibles
  ➔ $\prod_{x \in Z/nZ^*} x = \prod_{y \in Ga} y = \prod_{x \in Z/nZ^*} ax$
  ➔ $a^{|Z/nZ^*|} = 1$

- Théorème de Fermat (1601-1665) : $a^P \equiv a$ [p] pour p premier

Théorème Chinois (Qin Jiu-Shao XIII^{e} siècle)

- Soient m_1, …, m_n positifs et premiers entre eux
- M leur produit
- Alors, pour tous résidus a_1, …, a_n il existe un unique x < M tel que les restes de la division de x par m_i coïncident avec les a_i :

$$\forall a_1, \ldots, a_k, \exists! x < M, x \equiv a_i [m_i]$$

- Construction (par Lagrange (1736-1813)) :
  - Posons $M_i = M/m_i$,
  - D'après l'AEE, $\exists$ y_i tel que $y_i M_i \equiv 1 [m_i]$
  - Alors $x \equiv \sum a_i y_i M_i [M]$ convient !

$\exists$ Construction itérée par Newton (1643-1727) (différences multipliées)

Le théorème RSA

- Théorème *[Rivest-Shamir-Adleman 1978]* : pour p et q premiers, on pose n=pq et on a

$$\forall a \in Z/_n Z, a^{1+k(p-1)(q-1)} = a \mod n$$

- Preuve : petit théorème de Fermat + théorème chinois

- Choix d'un entier e premier avec $\varphi(pq) = (p-1)(q-1)$

- Algorithme d'Euclide étendu calcule les coefficients de Bézout d et k tels que $ed - (p-1)(q-1)k = 1$

Le code RSA

- Les entiers n et e choisis sont la clef PUBLIQUE
- L'entier d associé est la clef PRIVÉE secrète

  p et q sont également secrets

- Le chiffrement d'un message M s'effectue par découpage en blocs $m_i < pq$, puis exponentiation rapide avec la clef publique : $c_i = m_i^e \bmod n$

- Le déchiffrement d'un chiffré C s'effectue également par exponentiation rapide avec la clef privée :

  $m_i = c_i^d \bmod n$

Casser RSA

- Déchiffrer un message secret sans la clef privée
  - Connaissant c_i, calculer m_i directement :
  - $c_i \equiv m_i^e \bmod n$ se transforme en passant au logarithme en :
  - $\log_g(c_i) \equiv e \log_g(m_i) \bmod \varphi(n)$
  - ☹ Il est déjà difficile de calculer $\log_g(c_i)$
- C'est plus simple de trouver la clef privée
  - ⇨ Il faut calculer les coefficients de Bézout de e et $\varphi(n)$

⇨ Il faut calculer $\varphi(n)=(p-1)(q-1)$ sans connaitre ni p ni q

⇨ Factoriser n

Paramètres intéressants

- Chiffrement et déchiffrement faciles
  - Exponentiation rapide récursive par carrés :
    - $5^{11}\,[7] \equiv ((5^2)^2\,5)^2\,5 \equiv ((4)^2\,5)^2\,5 \equiv (2\times5)^2\,5 \equiv 2\times5 \equiv 3$
    - Complexité O(log(n)) multiplications de grands entiers soit $O(\log^3(n))$
    - En pratique une exponentiation à 60 000 chiffres (200 000 bits) en 1 seconde
- Cassage trop difficile
  - Meilleure factorisation connue en complexité $\approx n^{1/6}$
  - En pratique plusieurs mois pour casser du RSA à 200 chiffres

Fonction de hachage : résumé

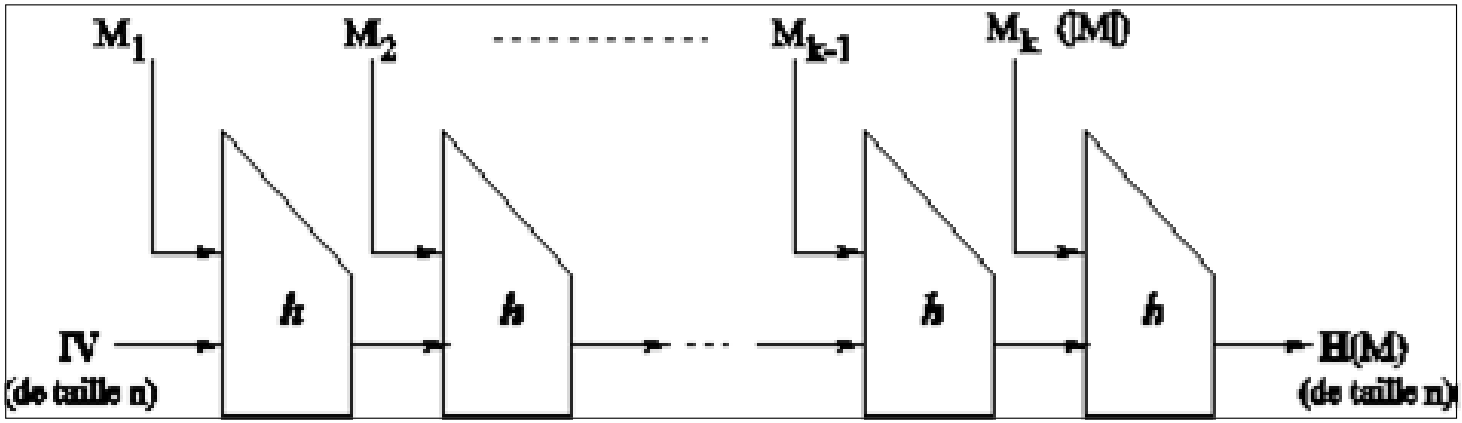
- H : $\{0,1\}^* \rightarrow \{0,1\}^n$
- Opère sur un message de longueur quelconque et fo urnit une valeur de hachage (le résumé) de longueur fixe n
  - Uniformité : Probabilité(H(M)=i) == $1/2^n$
  - Sécurité : à partir du résumé il est impossible de retrouver le texte

- Niveaux de sécurité
  - Résistance à la préimage : dûr de trouver x tq y=H(x)
  - Résistance à la 2nde préimage : ayant x, dûr de trouver x' tq H(x)=H(x')
  - Résistance aux collisions : dûr de trouver x et x' tq H(x)=H(x')

- MD5 (128 bits *[collisions 2004]*), SHA-1 160 bits *[collisions en théoriquement 2^{51} opérations]*, etc.

Calcul d'empreinte

- Itération d'une ronde h : $\{0,1\}^b x \{0,1\}^n \rightarrow \{0,1\}^n$



- h résiste aux collisions => H aussi *[Merkle-Damgård]*
  - Sinon : collision sur h à la première différence

- Attaque par force brute
  - Combien de x doit-on générer pour trouver une collision ?
  - ⇨ Entre 2 et 2^n+1

Signatures numériques

- Authentique : convainc le destinataire que le signataire a délibérément signé un document
- Infalsifiable : preuve que le signataire a délibérément signé
- Non réutilisable : attachée à un document
- Inaltérable : aucune modification du document signé
- Non reniable : le signataire ne peut répudier le document

Intégrité + Authentification + Non-répudiation

Signatures

- MAC sont symétriques
- Signatures sont asymétriques

Signatures RSA

- Hachage + Clef publique
  - Crypter H(M) avec la clef privée de l'envoyeur
  - Vérifier en décryptant et recalculant le hash

+ Interdire les rejeux : Crypter plutôt H(M || Date)

DSS/DSA : SHA-1 + ElGamal/Log discret

- Choisir q premier de 160 bits
- Trouver p = kq+1 premier de 512 à 1024 bits
- Tant que (g==1) choisir a et calculer $g \equiv a^k$ [p]
- Choisir x de 160 bits : clef privée
- $y \equiv g^x$ [p] : p, q, g, y publics

1. Alice choisit k aléatoire inférieur à q
2. Alice calcule et envoie
   - $r = (g^k \bmod p) \bmod q$
   - $s = (k^{-1} (SHA1(Message)+xr)) \bmod q$
3. Bob vérifie la signature ssi (v == r)
   - $w \equiv s^{-1}$ [q]
   - $u \equiv SHA1(Message)\ w$ [q]
   - $t \equiv r\ w$ [q]
   - $v = (g^u\ y^t \bmod p) \bmod q$

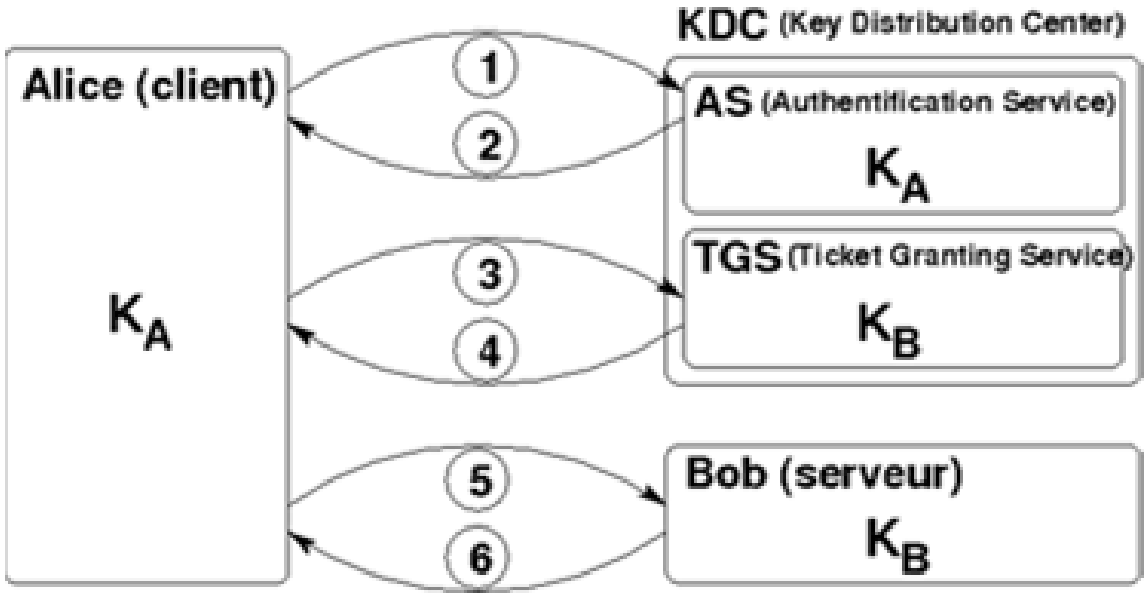
Architectures et protocoles

- Échanger des clefs
  - Diffie-Hellman : man-in-the-middle
  - Échange « out-of-band »
  - Tiers de confiance
    - Clefs secrètes : Kerberos [MIT]
    - Clefs publiques : certificats, PKI
- Sessions cryptées : systèmes hybrides de chiffrement
  - PGP
  - SSH

Échange de clef

- Challenge
  - KDC envoie un « challenge », nombre aléatoire que le client doit crypter avec sa clef secrète : authentification
- Tiers de confiance
  - Pas besoin d'échanger entre tous les utilisateurs possibles, juste avec le KDC
  - Le KDC peut alors distribuer des clefs de session ciblées
- Horodatage
  - Empêche les re-jeux

Kerberos



- Kerberos = Challenge + Tiers + Horodatage
  1. Bonjour
  2. $K_{K,A}$ || challenge
  3. Je veux parler à Bob || réponse challenge datée
  4. $K_{A,B}$ || Ticket daté pour Bob contenant $E_{KB}(K_{A,B})$
  5. Ticket pour Bob || Je suis Alice daté
  6. Bonjour Alice daté

Certificats Numériques

- Comment rattacher une clef publique à son propriétaire ?
  - Certificat numérique : association identité + clef
- Comment authentifier un certificat ?
  - Les certificats sont signés par des Autorités
- Comment récupérer un certificat ?
  - Des annuaires sont maintenus
- Comment gérér / sécuriser ce mécanisme
  - Auto signatures des Autorités
  - Architectures à clef publiques : PKI …

Système hybride : clef secrète / clef publique

- Initiée par PGP [Zimmermann]
  - Cryptographie secrète : rapide mais problème du tiers
  - Cryptographie publique : lente

- SSH
  1. Client contacte le serveur
  2. Serveur envoie ses clefs publiques H fixe et S changée régulièrement + challenge
  3. Client vérifie H correspond à sa BD
  4. Client double crypte clef secrète de session par H et S
  5. Confirmation du serveur

  ⇨ Connexion sécurisée établie