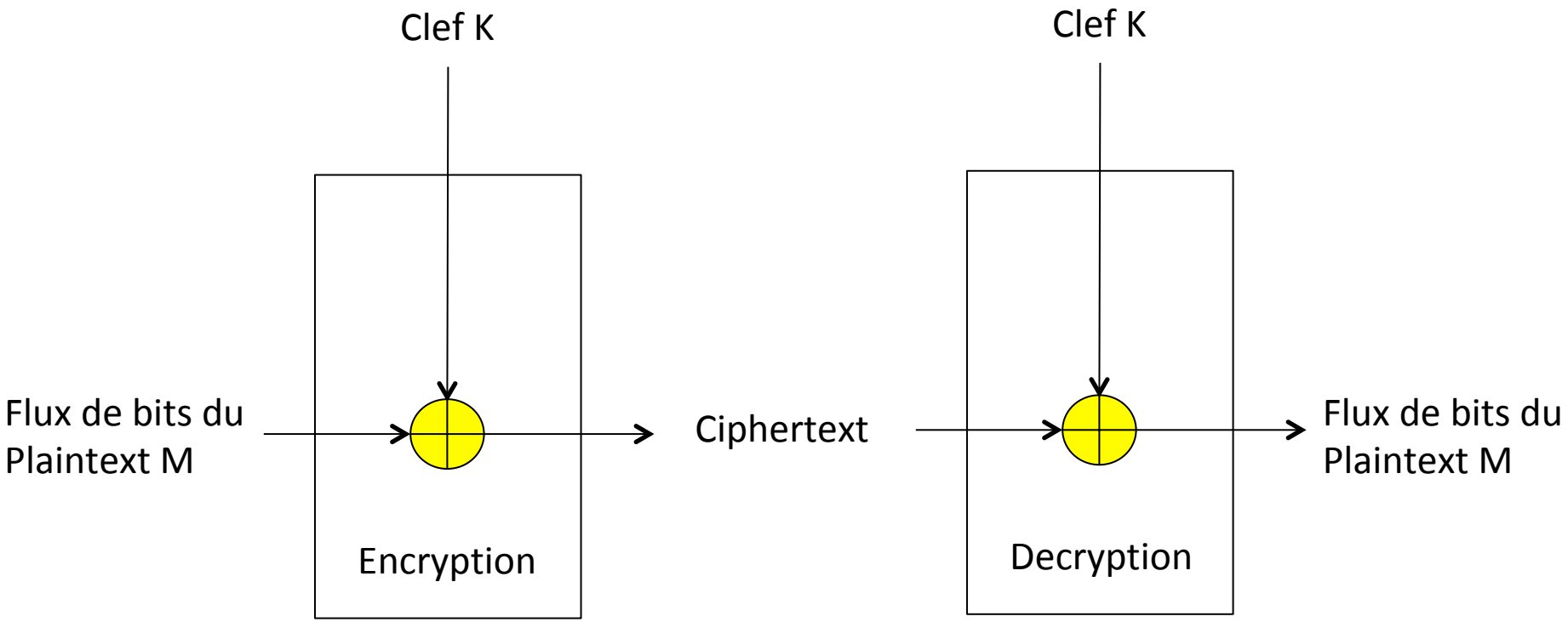
Chiffrement par flot (Stream cipher)
Générateurs Pseudo-aléatoires
et LFSRs

Stream cipher (chiffrement de flux)



Stream cipher (chiffrement de flux)

Chiffrement de Vernam (1918)

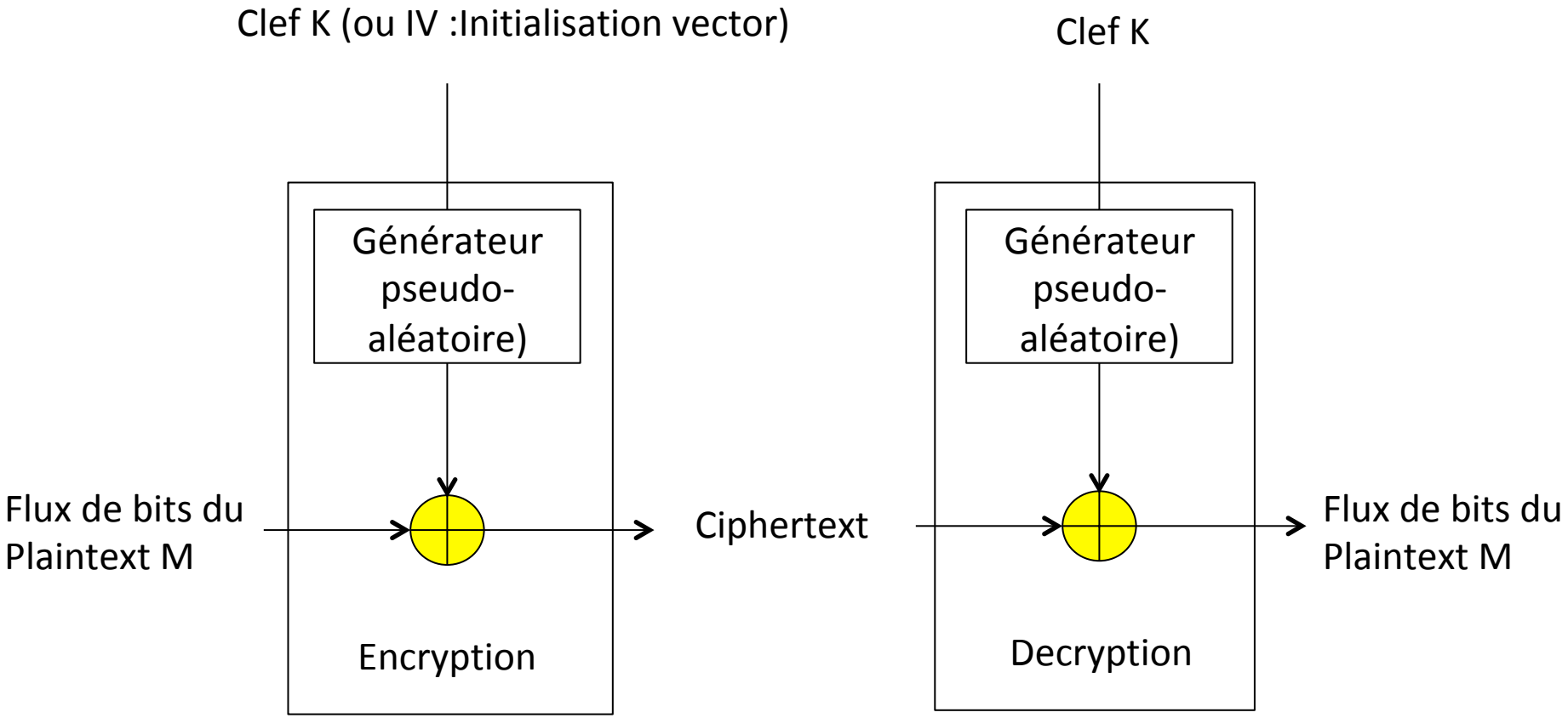
Clef = $k_1k_2k_3k_4$…… (aléatoire, utilisée une seule fois "One-time PAD")
Plaintext = $m_1m_2m_3m_4$……
Chiffré (cipher) : $c_1c_2c_3c_4$….. avec $c_i = m_i + k_i$ (+ : ou-exclusif)

Prouvé inconditionnellement sûr

Mais :
- la clef doit être aussi longue que le message
- partage de la clef (infinie) ?
- comment générer une telle clef ?

En pratique :
Key-stream : longue suite (pseudo aléatoire)
Key-stream : générée à partir d'une clef secrète courte.

Stream cipher (chiffrement de flux)



Pseudorandom Number Generators (PRNGs)

- On utilise des algorithmes déterministes pour générer des “nombres aléatoires
  - pas réellement aléatoires
  - mais satisfaisants aux tests "d'aléatorité"

- Connus sous le terme “Nombres Pseudo-aléatoires"

- Générés par des "générateurs pseudo-aléatoires" “Pseudorandom Number Generators (PRNGs)”

PRNGs et Stream Cipher

- Considérations de design :
  - Période longue sans répétitions
  - Statistiquement aléatoires
  - Dépendant d'une clef suffisamment grande
  - Complexité linéaire élevée
- Si bien conçus, (presque) aussi sûrs qu'un chiffrement par bloc de même taille de clef
- Bénéfice : habituellement plus simple et plus rapide

Conditions sur les PRNG

- Aléatoirité
  - uniformité, mise à l'échelle
- Imprévisibilité
  - Imprévisibilité (avant et arrière)
  - Utilisation des mêmes tests pour vérification
- Caractéristiques de la graine
  - si l'attaquant connait la graine -> toute la séquence
  - doit donc être un nombre aléatoire ou pseudo-aléatoire

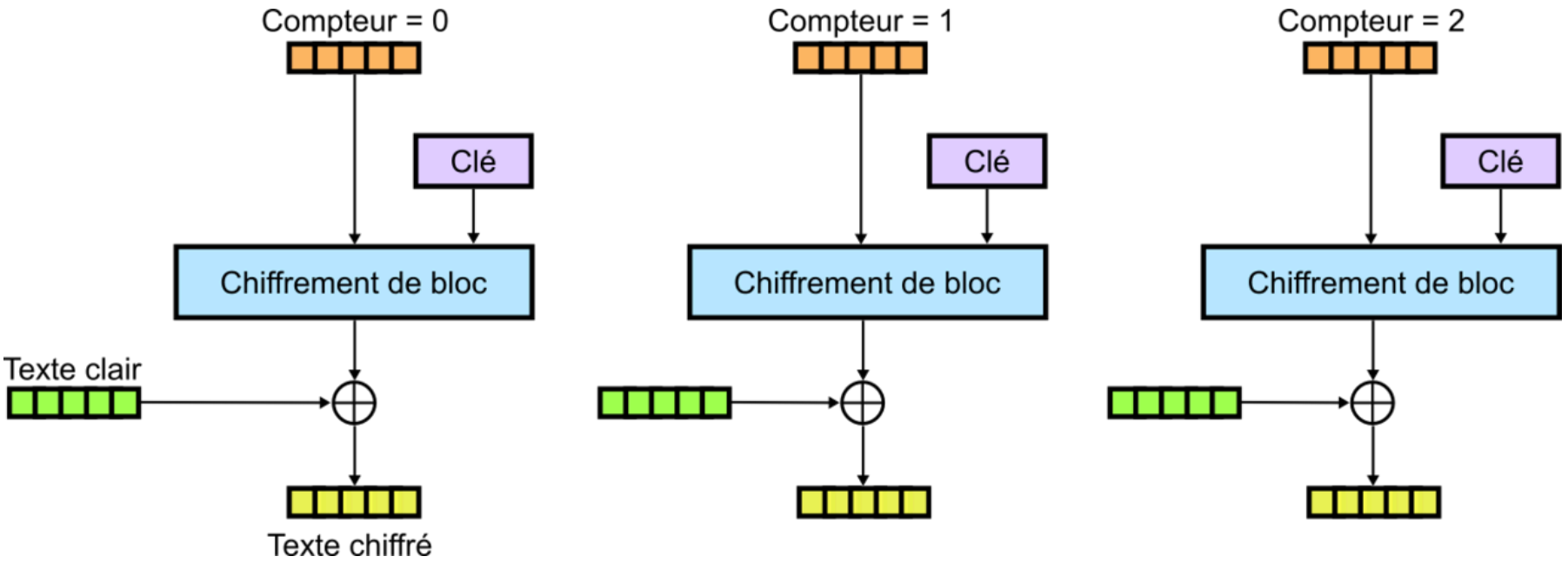
Générateurs Pseudo-Aléatoires

- Block cipher : DES, AES etc.. en mode CFB, OFB, CTR
- LFSR
- Autres

Générateurs Pseudo-Aléatoires : Block Ciphers

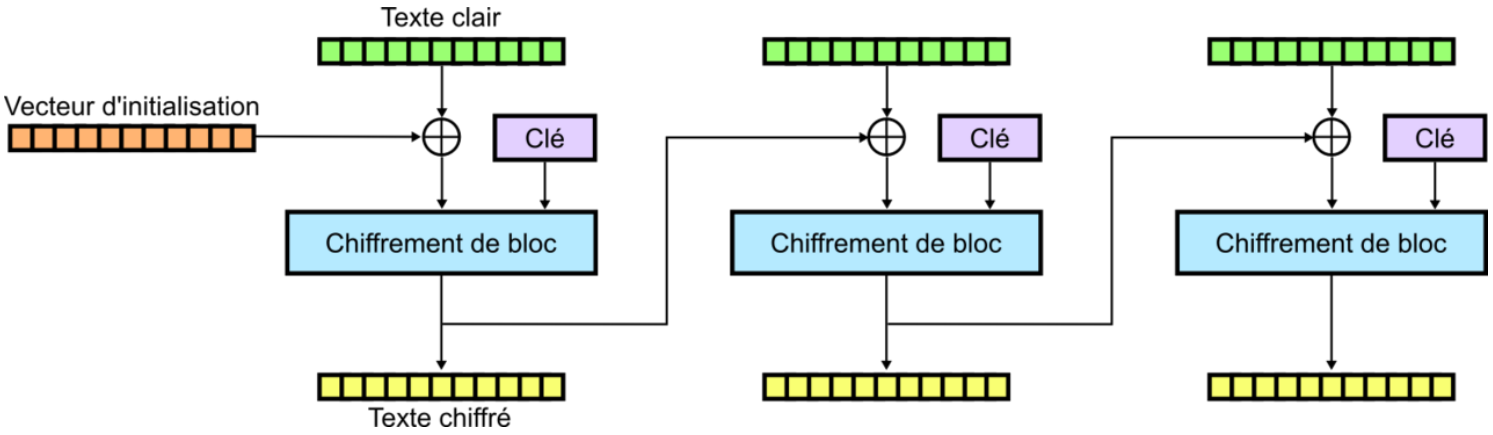
- Block cipher : DES, AES etc.. en mode CFB, OFB, CTR

- Exemple : mode CTR (counter)



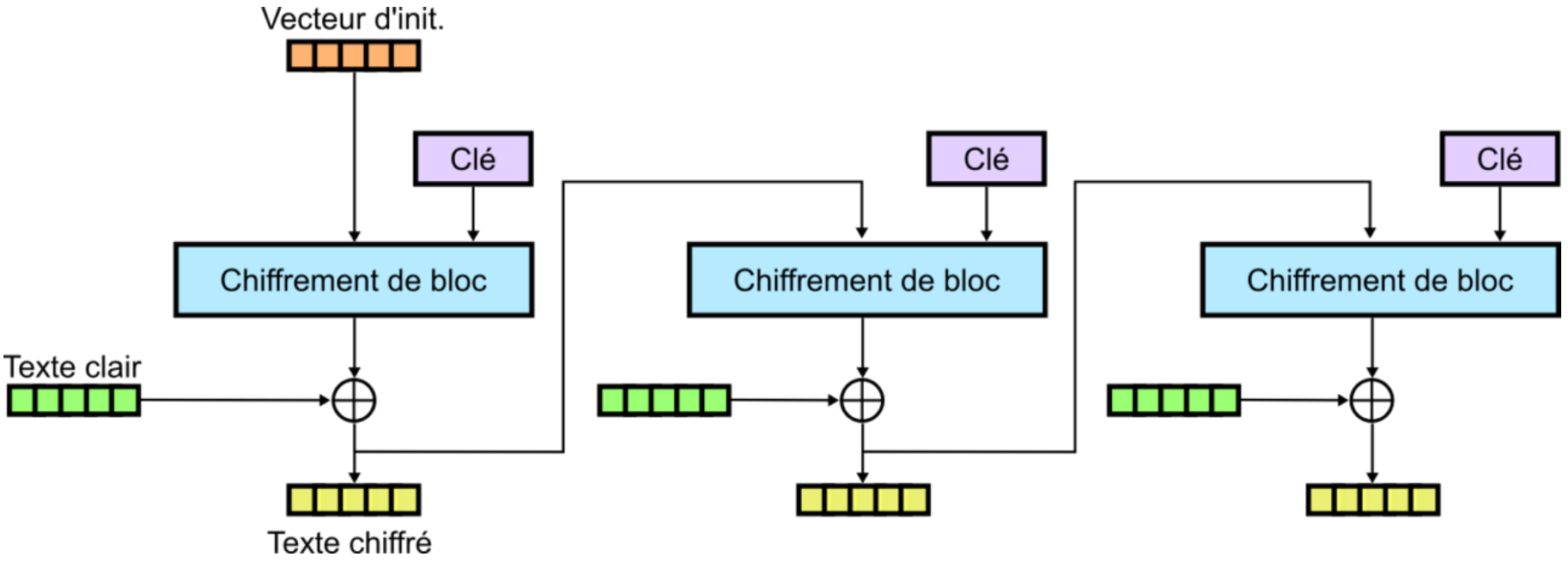
Générateurs Pseudo-Aléatoires : Block Ciphers

- Mode CBC (Cipher Block Chaining)



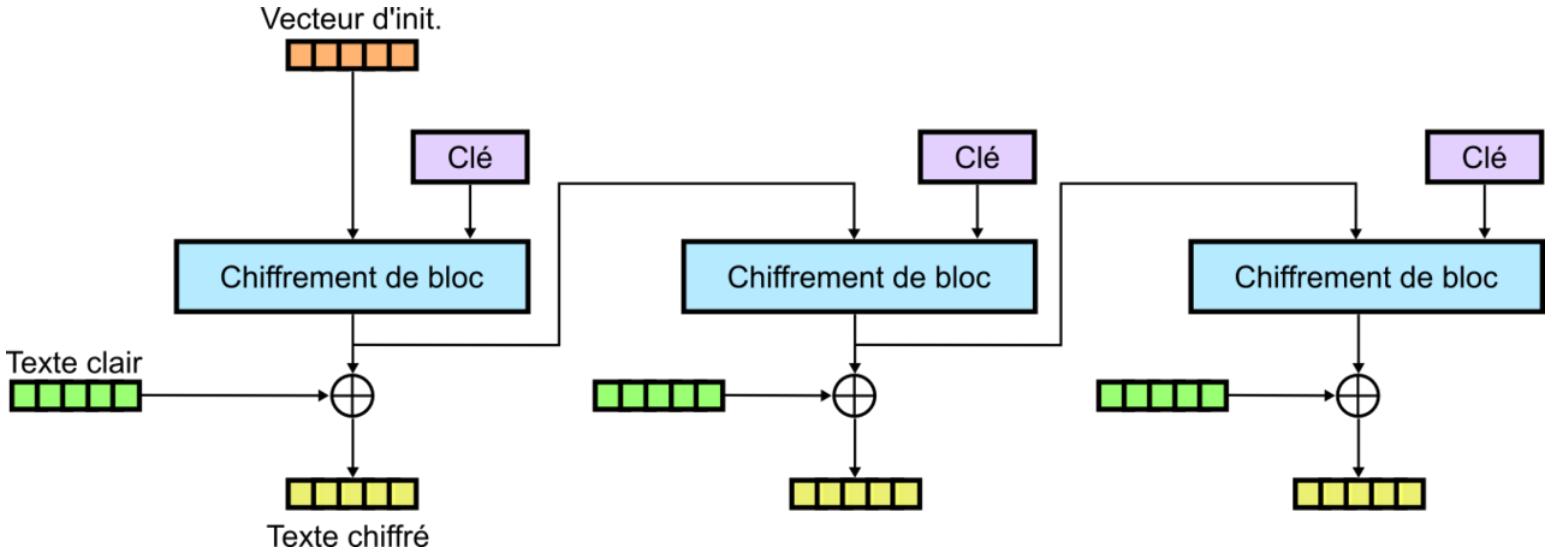
Générateurs Pseudo-Aléatoires : Block Ciphers

- Mode CFB (Cipher Feedback)
- CFB est un chiffrement par flot.
- intérêt : il ne nécessite que la fonction de chiffrement, ce qui le rend moins cher à câbler ou programmer pour les algorithmes ayant une fonction de chiffrement différente de la fonction de déchiffrement (exemple: AES).



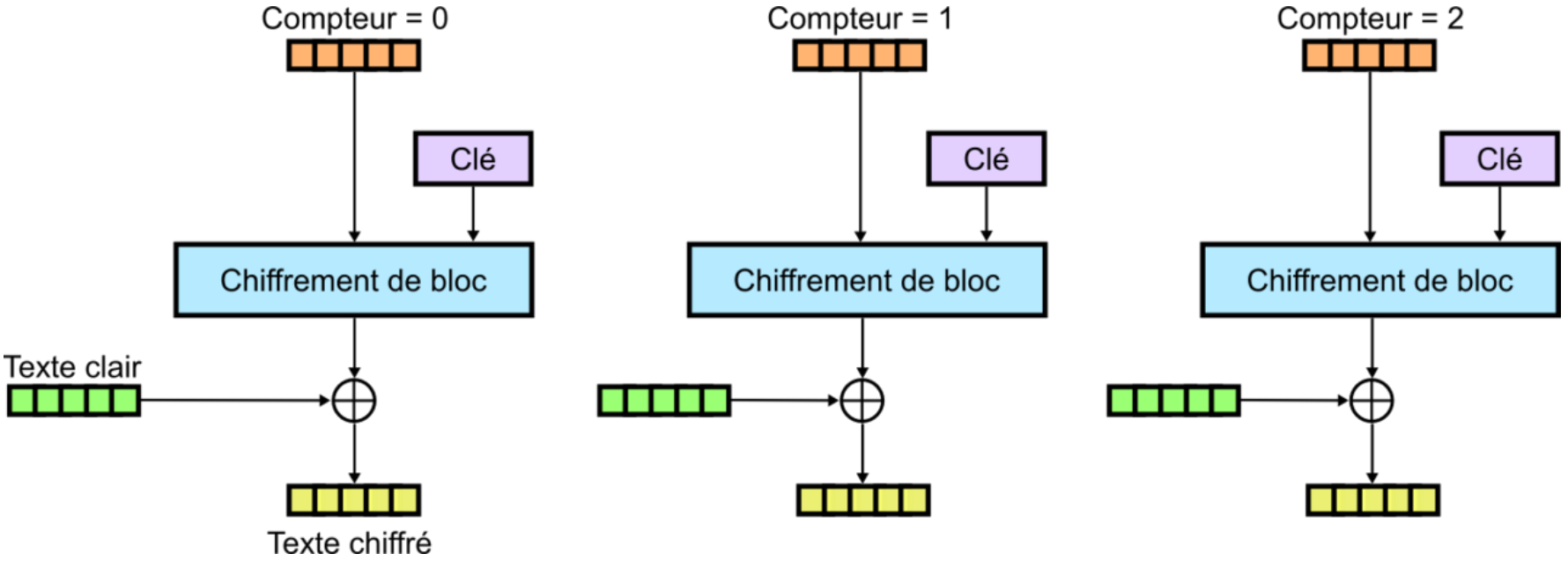
Générateurs Pseudo-Aléatoires : Block Ciphers

- Mode OFB (Output Feedback)
- C'est un mode de chiffrement de flot qui possède les mêmes avantages que CFB. De plus, il est possible de le précalculer en chiffrant successivement le vecteur d'initialisation. Il n'est donc sûr que si la fonction de chiffrement alliée à la clé forment une bonne suite pseudo-aléatoire.

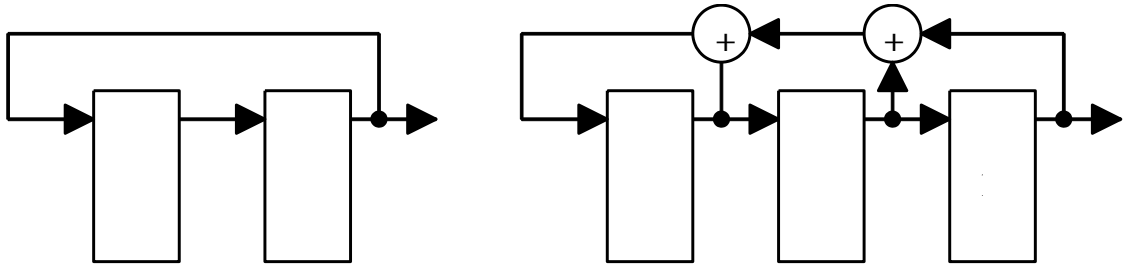


Générateurs Pseudo-Aléatoires : Block Ciphers

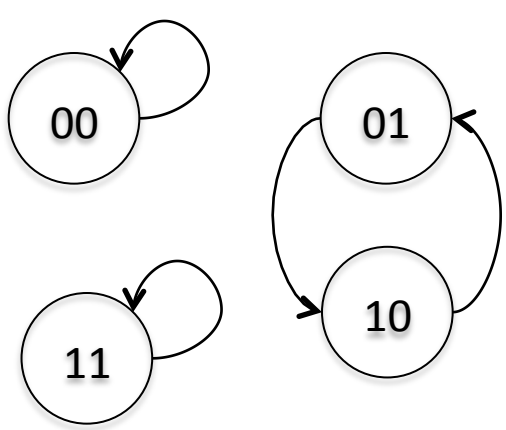
- Mode CTR (Counter)
- Dans ce mode, le flux de clé est obtenu en chiffrant les valeurs successives d'un compteur.
- Nombreux avantages :
  - chiffrement par flot
  - précalculable
  - accès aléatoire aux données
  - parallélisable

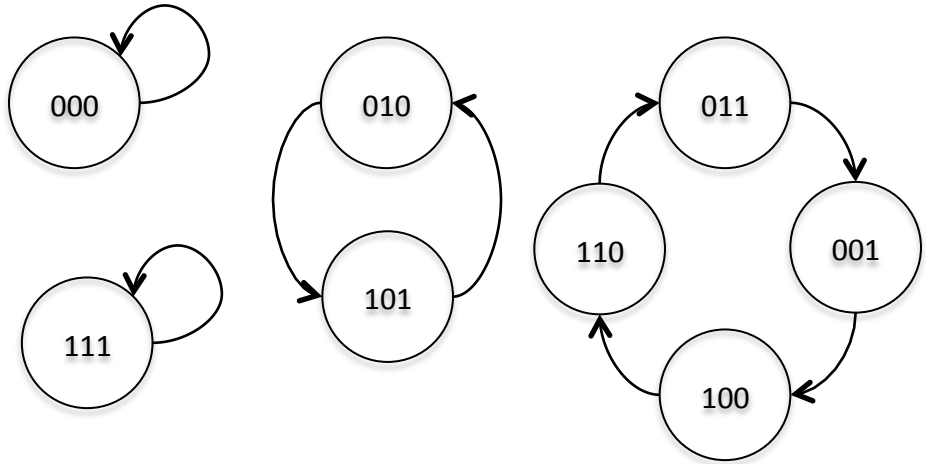


LFSR : Linear Feedback Shift Register
(registre à décalage rebouclé linéaire)

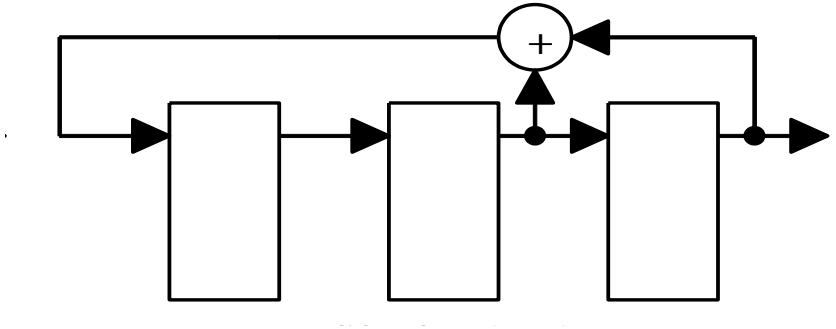


Circuits "cycliques" : parcours d'états cyclique (cf. compteurs)





Exemple



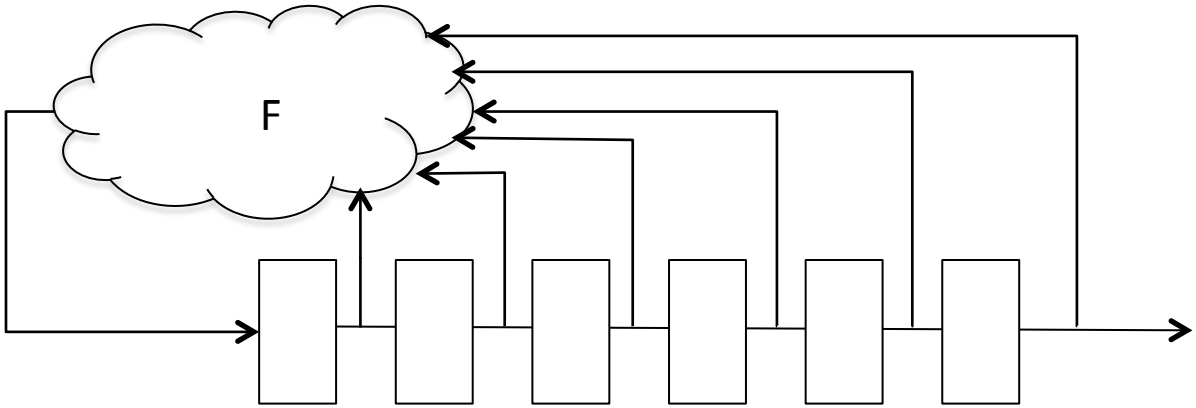
+ : Ouex

S0	0	1	1
S1	0	0	1
S2	1	0	0
S3	0	1	0
S4	1	0	1
S5	1	1	0
S6	1	1	1
S7	0	1	1

Longueur de la séquence = 2^3-1

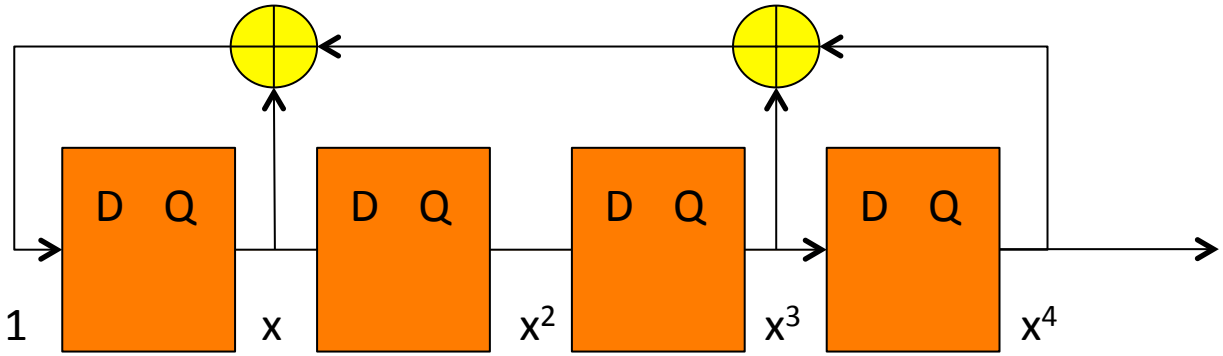
LFSR à séquence maximale

Forme générale

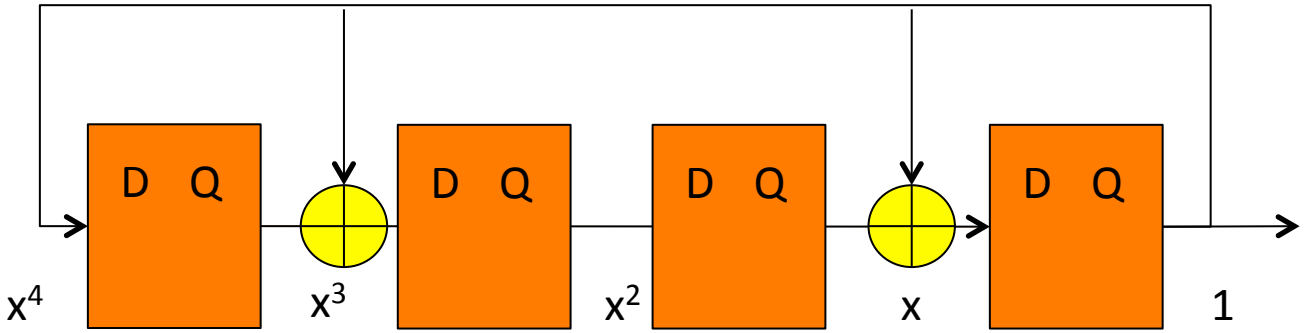


F = fonction linéaire de certaines FFs
= "OUEX"de certaines FFs

Types de LFSR



Fibonnaci



Galois

Polynôme caractéristique
– défini par la position des XOR
– **P**(**x**) = **x**^{4} + **x**^{3} + **x** + 1 pour les 2 exemples

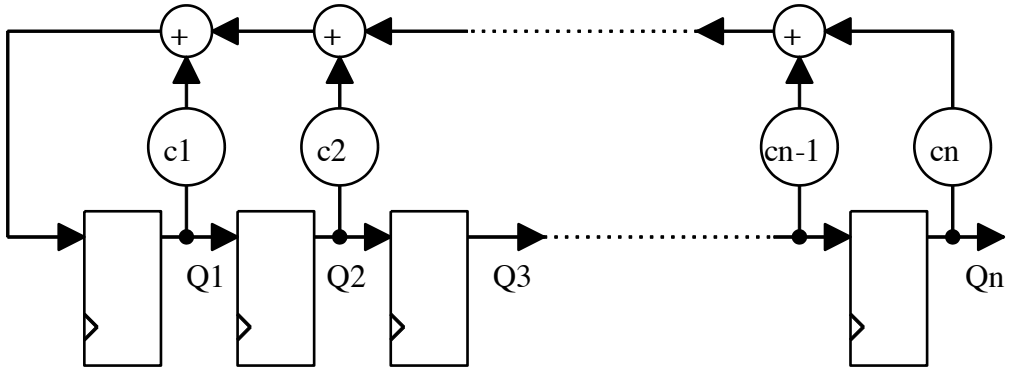
Forme générale

$$P(x) = 1 + \sum_{i=1}^{n} c_i . x^i$$

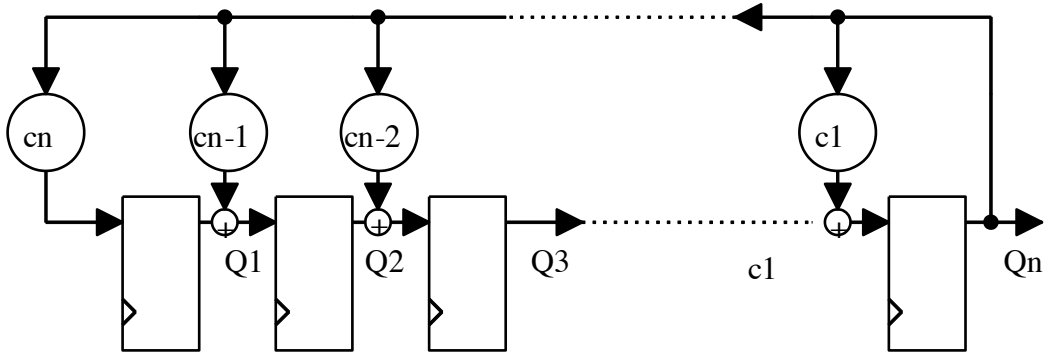
ci =1 : la connexion existe

ci = 0 : la connexion n'existe pas

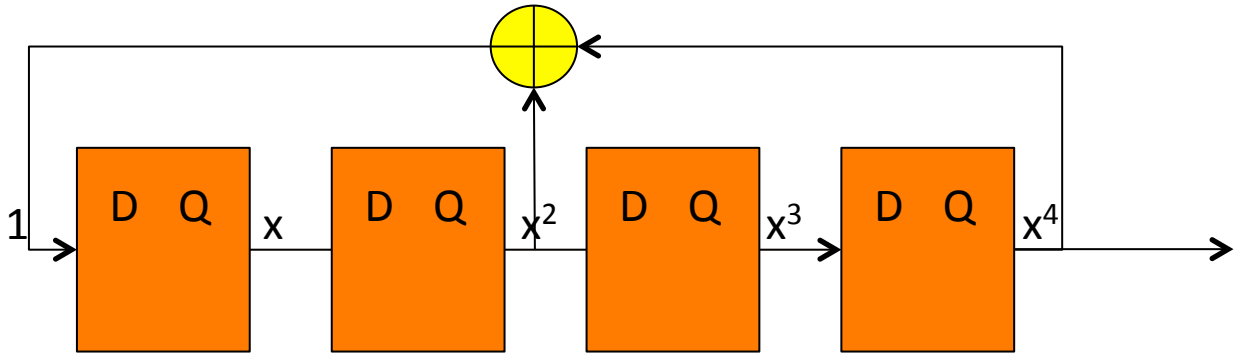
TYPE 1



TYPE 2

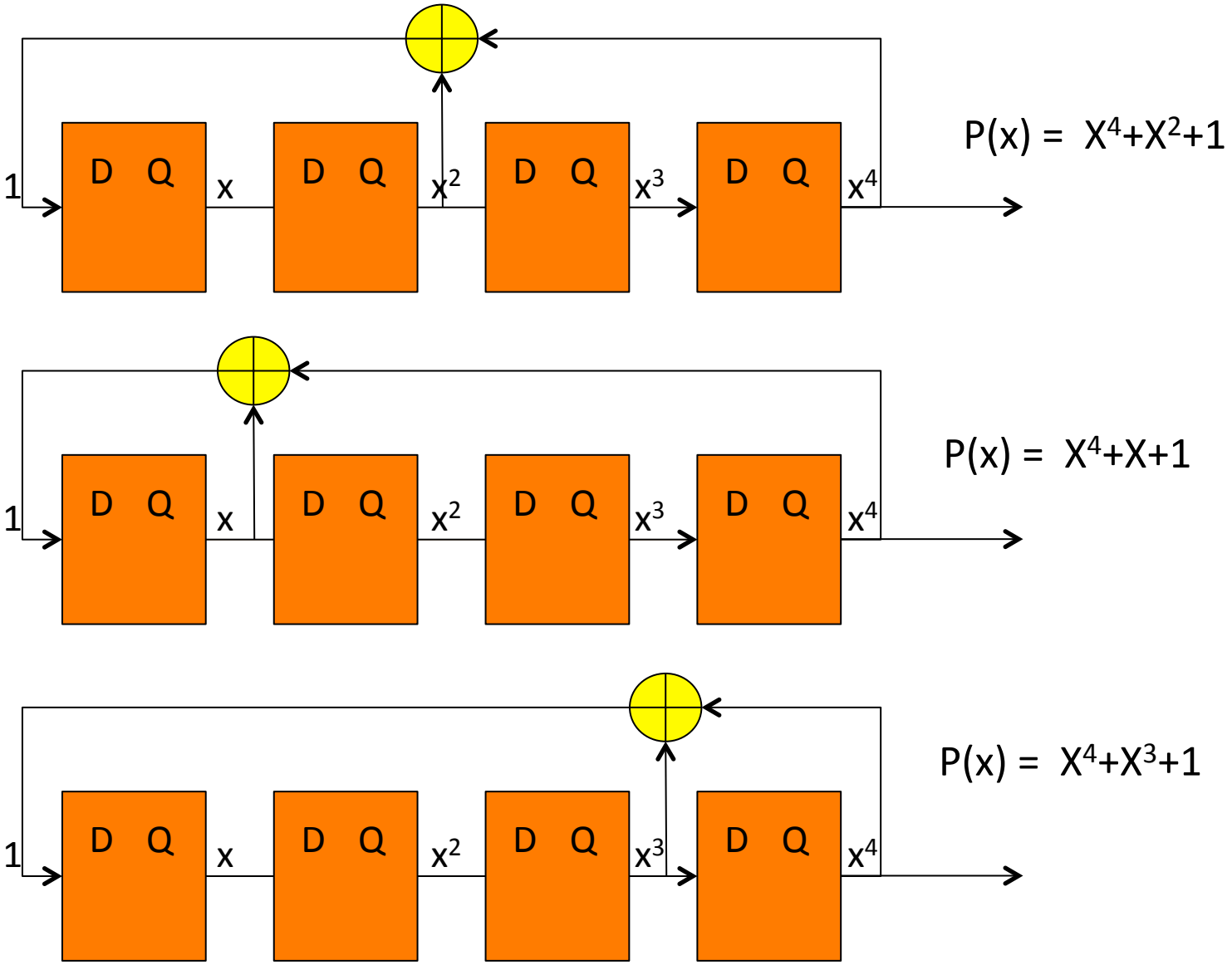


LFSR

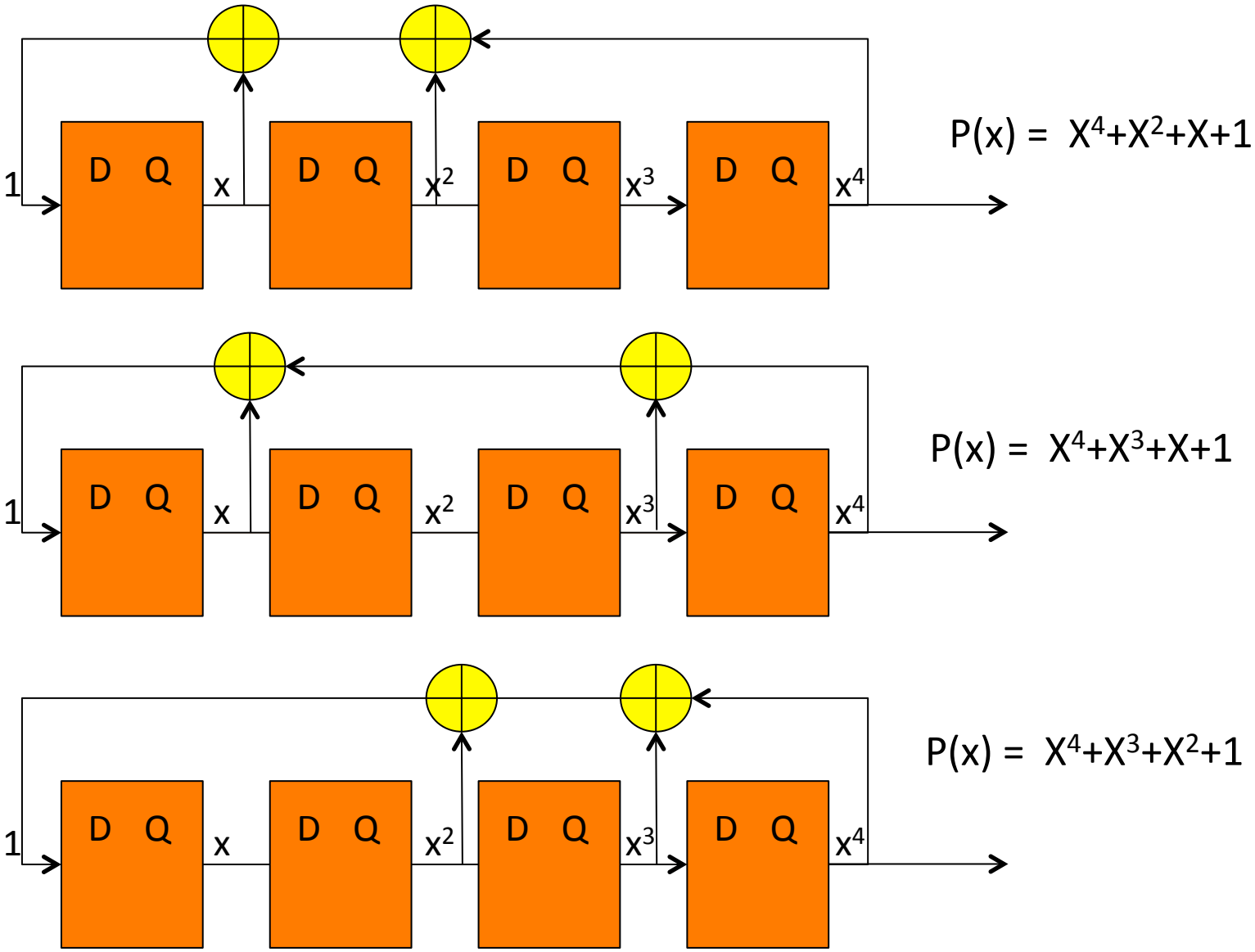


0	0	0	1
1	0	0	0
0	1	0	0
1	0	1	0
0	1	0	1
0	0	1	0
0	0	0	1

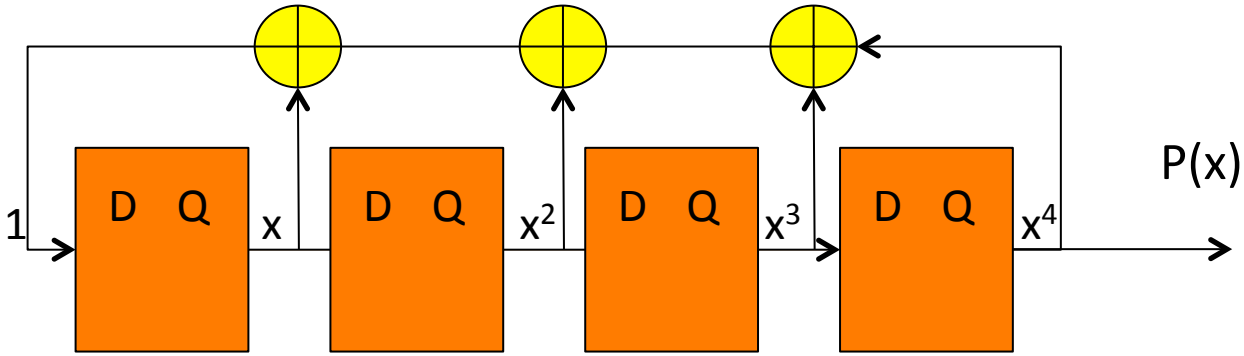
LFSR


1
D Q
x
D Q
x²
D Q
x³
D Q
x⁴
P(x) = X⁴+X²+1
1
D Q
x
D Q
x²
D Q
x³
D Q
x⁴
P(x) = X⁴+X+1
1
D Q
x
D Q
x²
D Q
x³
D Q
x⁴
P(x) = X⁴+X³+1

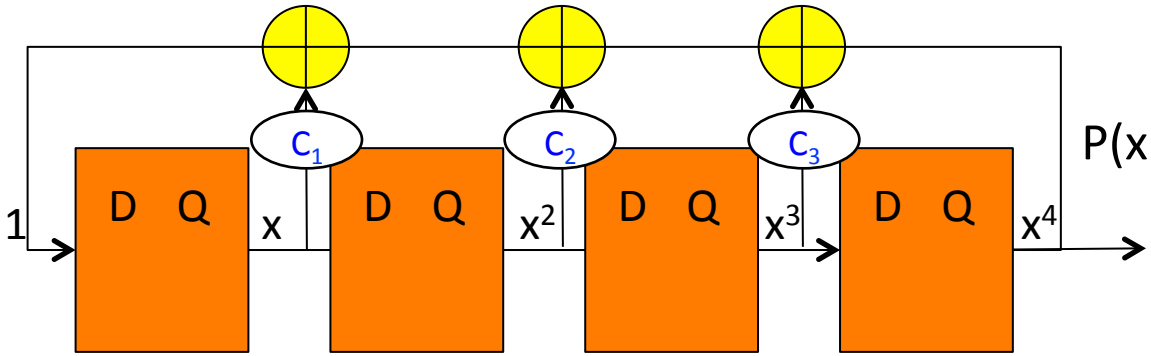
LFSR


1
D Q
x
D Q
x²
D Q
x³
D Q
x⁴
P(x) = X⁴+X²+X+1
1
D Q
x
D Q
x²
D Q
x³
D Q
x⁴
P(x) = X⁴+X³+X+1
1
D Q
x
D Q
x²
D Q
x³
D Q
x⁴
P(x) = X⁴+X³+X²+1

LFSR



$P(x) = X^4+X^3+X^2+X+1$

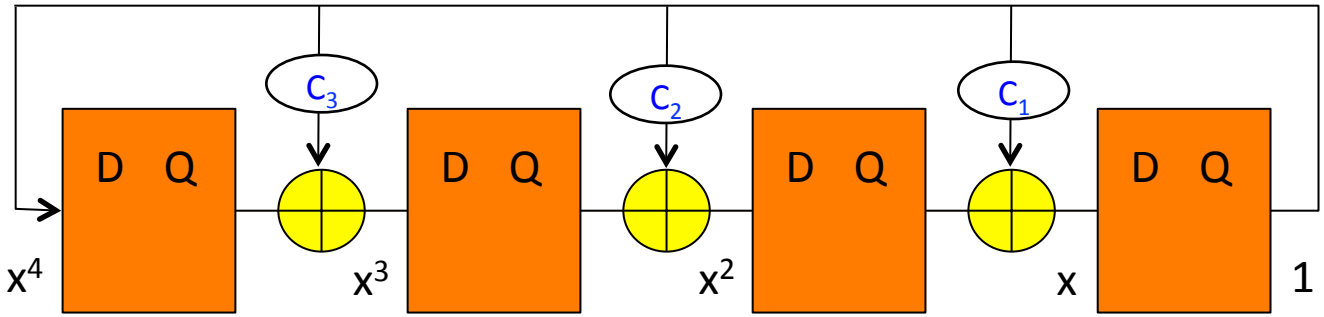


$P(x) = X^4+C_3X^3+C_2X^2+C_1X+1$

C_i=1 la connexion existe

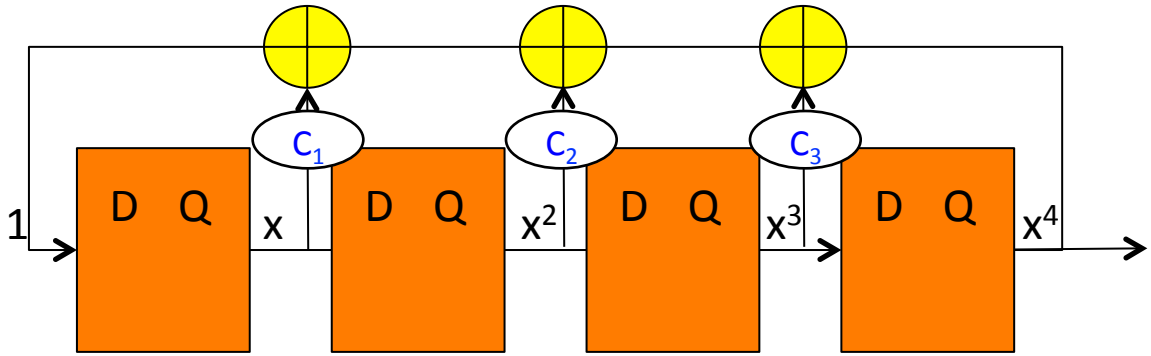
C_i=0 la connexion n'existe pas

LFSR de Fibonacci



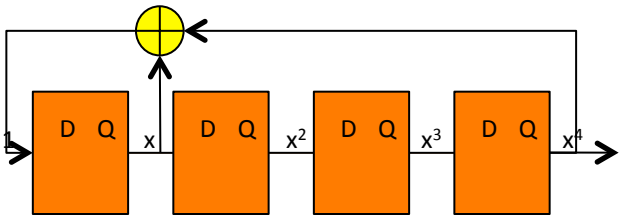
LFSR de Gallois équivalent

LFSR



n FFS => 2^{n-1}-1 LFSR possibles => P(x) = $X^4+C_3X^3+C_2X^2+C_1X+1$

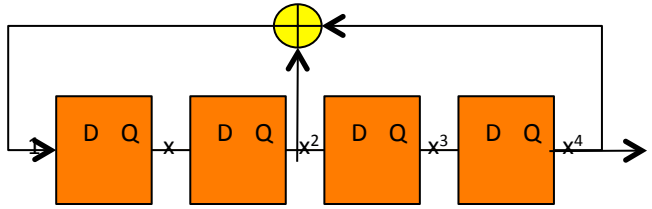
LFSR



$P(x) = X^4+X+1$

0	0	0	1
1	0	0	0
1	1	0	0
1	1	1	0
1	1	1	1
0	1	1	1
1	0	1	1
0	1	0	1
1	0	1	0
1	1	0	1
0	1	1	0
0	0	1	1
1	0	0	1
0	1	0	0
0	0	1	0
0	0	0	1

2^n-1 valeurs

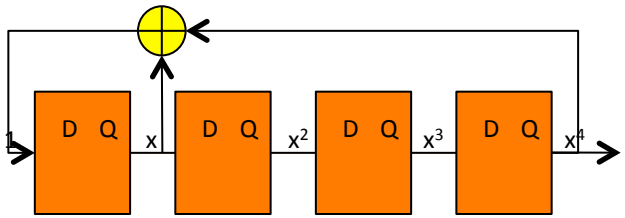


$P(x) = X^4+X^2+1$

0	0	0	1
1	0	0	0
0	1	0	0
1	0	1	0
0	1	0	1
0	0	1	0
0	0	0	1

6 valeurs

LFSR



0	0	0	1
1	0	0	0
1	1	0	0
1	1	1	0
1	1	1	1
0	1	1	1
1	0	1	1
0	1	0	1
1	0	1	0
1	1	0	1
0	1	1	0
0	0	1	1
1	0	0	1
0	1	0	0
0	0	1	0
0	0	0	1

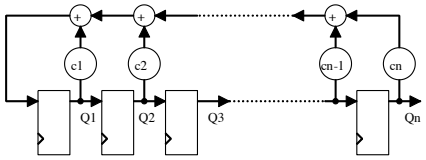
$P(x) = X^4+X+1$ => Polynôme primitif

2^n-1 valeurs

Séquence générée

$\{a_m\} = a_0, a_1, a_2, \ldots$ représente la séquence de sortie générée par le LFSR avec $a_i = 0$ ou 1

notée $G(x) = \sum_{m=0}^{\infty} a_m . x^m$ Par ailleurs, LFSR : $a_m = \sum_{i=1}^{n} c_i . a_{m-i}$



$$G(x) = \sum_{m=0}^{\infty} \sum_{i=1}^{n} c_i . a_{m-i} . x^m = \sum_{i=0}^{n} c_i . x^i \sum_{m=0}^{\infty} a_{m-i} . x^{m-i} = \sum_{i=0}^{n} c_i . x^i . \left[a_{-i} . x^{-i} + \ldots . + a_{-1} . x^{-1} + \sum_{m=0}^{\infty} a_{m-i} . x^{m-i} \right].$$

$$G(x) = \sum_{i=0}^{n} c_i . x^i \left[a_{-i} . x^{-i} + \ldots . + a_{-1} . x^{-1} + G(x) \right]$$

$$G(x) = \sum_{i=0}^{n} c_i . x^i . G(x) + \sum_{i=0}^{n} c_i . x^i \left[a_{-i} . x^{-i} + \ldots . + a_{-1} . x^{-1} \right]$$

$$G(x) = \frac{\sum_{i=0}^{n} c_i . x^i \left[a_{-i} . x^{-i} + \ldots . + a_{-1} . x^{-1} \right]}{1 + \sum_{i=0}^{n} c_i . x^i}$$

G(x) est donc fonction de l'état initial a_{-n}, …, a_{-1} et de la fonction de rebouclage.

Séquence générée

$$G(x) = \frac{\sum_{i=0}^{n} c_i.x^i \left[a_{-i}.x^{-i} + ... + a_{-1}.x^{-1} \right]}{1 + \sum_{i=0}^{n} c_i.x^i}$$

Polynôme caractéristique P(x)

G(x) est donc fonction de l'état initial a_{-n}, …, a_{-1} et de la fonction de rebouclage

Pour un LFSR à n étages, $c_n = 1$.
Si on suppose que l'état initial est $a_{-1} = a_{-2} = ... = a_{-(n-1)} = 0$ et $a_{-n} = 1$ alors on obtient

$$G(x) = \frac{1}{P(x)}$$

Séquence générée

$$G(x) = \frac{1}{P(x)} = \sum_{m=0}^{\infty} a_m . x^m$$

Puisque la séquence est cyclique de <u>période p,</u> on obtient :

$$\frac{1}{P(x)} = \sum_{m=0}^{\infty} a_m . x^m = (a_0 + a_1 x + \cdots + a_{p-1} x^{p-1}) + x^p (a_0 + a_1 x + \cdots + a_{p-1} x^{p-1}) + x^{2p} (a_0 + a_1 x + \cdots + a_{p-1} x^{p-1}) + \cdots$$

$$\frac{1}{P(x)} = \frac{(a_0 + a_1 x + \cdots + a_{p-1} x^{p-1})}{1 - x^p}$$

Donc P(x) divise $1-x^p$

Polynômes primitifs

- Théorème : Si l'état initial d'un LFSR est $a_{-1} = a_{-2} = ... = a_{-(n-1)} = 0$ et $a_{-n} = 1$, alors la séquence $\{a_m\}$ du LFSR est périodique avec une période égale au plus petit entier k tel que P(x) divise $(1-x^k)$.

- Définition: Si la séquence générée par un LFSR à n étages est de longueur 2^n-1, elle est appelée séquence de longueur maximum

- Définition: Le polynôme caractéristique associé à une séquence de longueur maximum est appelé polynôme primitif

- Définition: Un polynôme irréductible est un polynôme qui ne peut être factorisé c'est à dire qu'il n'est divisible que par 1 et lui-même.

Polynômes primitifs

- Théorème: Un polynôme irréductible satisfait les deux conditions suivantes :
  - il a un nombre impair de termes (le terme 1 inclus)
  - si il est de degré supérieur à 3 alors P(x) doit diviser $(1+x^k)$, avec $k=2^n-1$
- Dem :
  - s'il ne contient pas le 1, 0 est racine (ex : $x^8 + x^4 + x^3 + x^2)$
  - si il a un nombre pair de termes, 1 est racine (ex : $x^8 + x^4 + x^3 + 1)$

- Théorème: Un polynôme irréductible est primitif si le plus petit entier positif k tel que P(x) divise $1+x^k$ est tel que $k=2^n-1$ avec n degré du polynôme P(x).

Tableau de polynômes primitifs

degré					degré					degré				
1	0				13	4	3	1	0	25	3	0		
2	1	0			14	12	11	1	0	26	8	7	1	0
3	1	0			15	1	0			27	8	7	1	0
4	1	0			16	5	3	2	0	28	3	0		
5	2	0			17	3	0			29	2	0		
6	1	0			18	7	0			30	16	15	1	0
7	1	0			19	6	5	1	0	31	3	0		
8	6	5	1	0	20	3	0			32	28	27	1	0
9	4	0			21	2	0			33	13	0		
10	3	0			22	1	0			34	15	14	1	0
11	2	0			23	5	0			35	2	0		
12	7	4	3	0	24	4	3	1	0	36	11	0		

P(x) = $\mathbf{x}^2 + \mathbf{x} + 1$

- 3 termes (impair)
- P(x) divise $(1+ \mathbf{x}^{4-1}) : (1+ \mathbf{x}^3) = (1+ \mathbf{x}) (\mathbf{x}^2 + x + 1)$
- P(x) ne divise ni $(\mathbf{x}^2+1)$ ni $(x+1)$

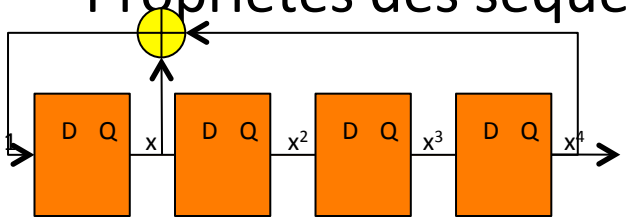
Tableau de polynômes primitifs

degré					degré					degré				
1	0				13	4	3	1	0	25	3	0		
2	1	0			14	12	11	1	0	26	8	7	1	0
3	1	0			15	1	0			27	8	7	1	0
4	1	0			16	5	3	2	0	28	3	0		
5	2	0			17	3	0			29	2	0		
6	1	0			18	7	0			30	16	15	1	0
7	1	0			19	6	5	1	0	31	3	0		
8	6	5	1	0	20	3	0			32	28	27	1	0
9	4	0			21	2	0			33	13	0		
10	3	0			22	1	0			34	15	14	1	0
11	2	0			23	5	0			35	2	0		
12	7	4	3	0	24	4	3	1	0	36	11	0		

P(x) = $\mathbf{x}^3 + \mathbf{x} + 1$

- 3 termes (impair)
- P(x) divise $(1+ \mathbf{x}^{8-1})$ et ne divise ni $(1+ \mathbf{x}^6)$ ni $(1+ \mathbf{x}^5)$ ni $(1+ \mathbf{x}^4)$ ni $(1+ \mathbf{x}^3)$ ni $(1+ \mathbf{x}^2)$ ni $(1+ \mathbf{x})$

Propriétés des séquences générées par un LFSR (m-séquences)



0	0	0	1
1	0	0	0
1	1	0	0
1	1	1	0
1	1	1	1
0	1	1	1
1	0	1	1
0	1	0	1
1	0	1	0
1	1	0	1
0	1	1	0
0	0	1	1
1	0	0	1
0	1	0	0
0	0	1	0

- Sur les 15 bits de sortie : 8 '1' et 7 '0'
- Le nombre de 1s d'une m-séquence diffère du nombre de 0s d'une unité
- Une m-séquence produit un nombre égal de salves de 1 et de 0
- Dans toute m-séquence, la moitié des salves sont de longueur 1, le quart de longueur 2, le huitième de longueur 3 et ainsi de suite tant que la fraction est un nombre entier.
- => Propriétés d'aléatoirité => LFSR : source (pseudo) aléatoire

A comparer avec un compteur

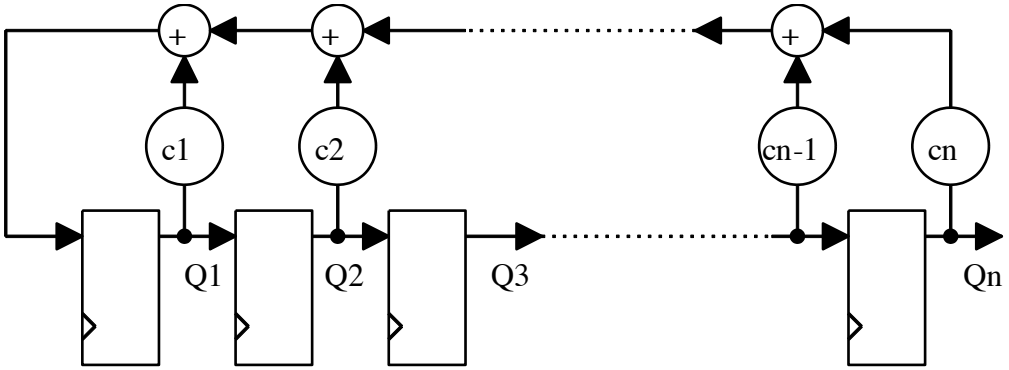
Générateurs pseudo-aléatoires : LFSRs

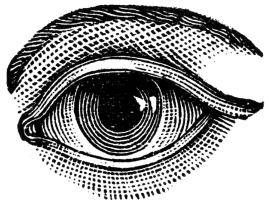
- Pour un LFSR de n bits : clef primaire (IV de n bits)
- Séquence de 2^n-1 bits aléatoires
- Rapide, peu coûteux en matériel

Mais, mais …. (attention)

Mais …

- Algorithme de Berlekamp-Massey
  - Soit un LFSR de n étages (poly de degré n)
  - Soit $\{s_0,s_1,\ldots..\}$ la séquence de sortie de période 2^n-1 (s_0 le dernier sorti) .
  - Connaissant les N=2n premières valeurs de la séquence, l'algo donne le poly de rétroaction g(x) …….





$g(x) = 1 + c_1 x + c_2 x^2 + \ldots\ldots + c_n x^n$

Idée : Algorithme de Berlekamp-Massey

- Etant donné un LFSR de 4 étages, on sait que:

  $s_4=s_3c_3+s_2c_2+s_1c_1+s_0c_0 \bmod 2$

  $s_5=s_4c_3+s_3c_2+s_2c_1+s_1c_0 \bmod 2$

  $s_6=s_5c_3+s_4c_2+s_3c_1+s_2c_0 \bmod 2$

  $s_7=s_6c_3+s_5c_2+s_4c_1+s_3c_0 \bmod 2$

  …….

- Connaissant $s_0,s_1,\ldots,s_7$, on peut calculer c_0,c_1,c_2,c_4. (4 équations, 4 inconnues)
- En général, connaissant 2n bits de sortie, on retrouve un LFSR à n étages

Algorithme de Berlekamp-Massey

- Entrée : $\{s_0, s_1, \ldots, s_{N-1}\}$ une suite de bits de longueur N
- Sortie : LFSR de longueur n = N/2 (construction de g(x))
- $g(x) = 1 + c_1 x + c_2 x^2 + \ldots\ldots + c_n x^n$ à trouver

Algo :

g(x)=1 ; m=-1 ; L=0; h(x)=1

Pour k = 0 -> N-1

Calculer : $d = s_k$ + Somme pour i de 1 à L de $c_i . s_{k-i}$ mod 2

Si d = 1

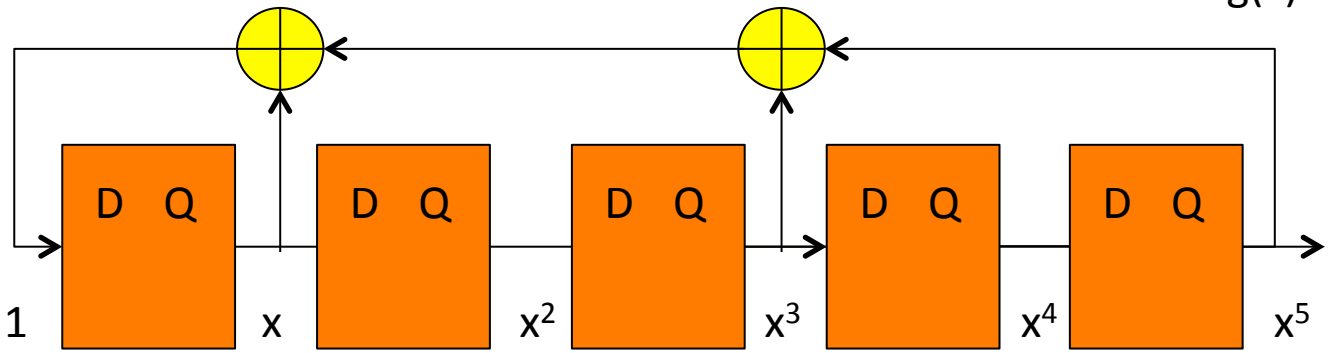
$t(x) = g(x)$

$g(x) = g(x) + h(x) . x^{k-m}$

Si $2L \leq k$ alors L=k+1-L, m = k, h(x)=t(x)

Exemple

$g(x) = X^5 + X^3 + X + 1$



0	1	1	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	0	1
0	0	1	1	0
1	0	1	1	1
…	…	…	…	…

Soit donc la suite ….1001101110 sortant du LFR

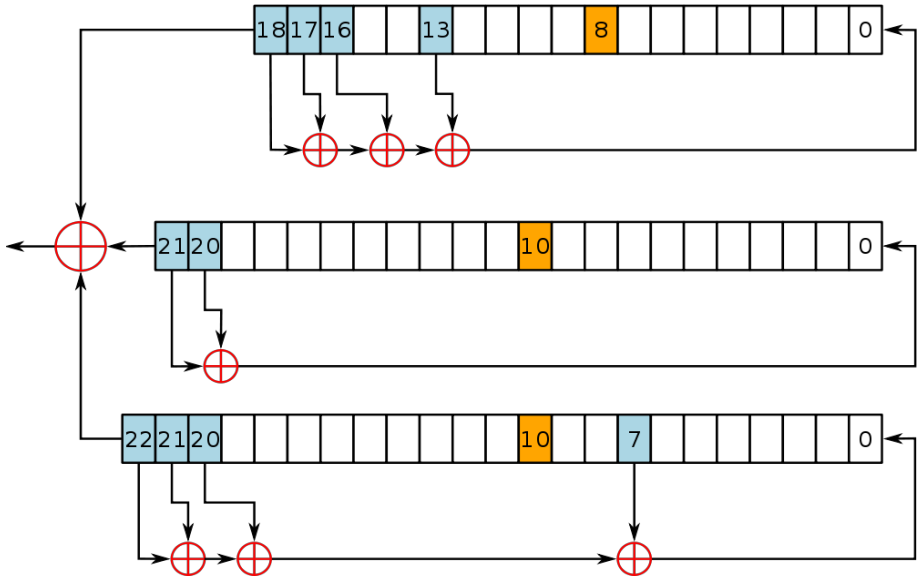
Retrouver le LFSR

Générateurs de clef

- Plusieurs LFSRs
- Fonctions non linéaire

GSM A5/1

Conversation GSM : division en blocs temporels de 4,6 millisecondes et contient 2×114 bits.
Une clé de session K est utilisée pour la communication.
Pour chaque bloc, K est mélangée à un compteur de blocs et le résultat est utilisé comme état initial d'un générateur de nombres pseudo-aléatoires qui produit 228 bits.



De plus, chaque registre est clocké si son "clocking bit" (orange) a la même valeur que la majorité des trois bits oranges

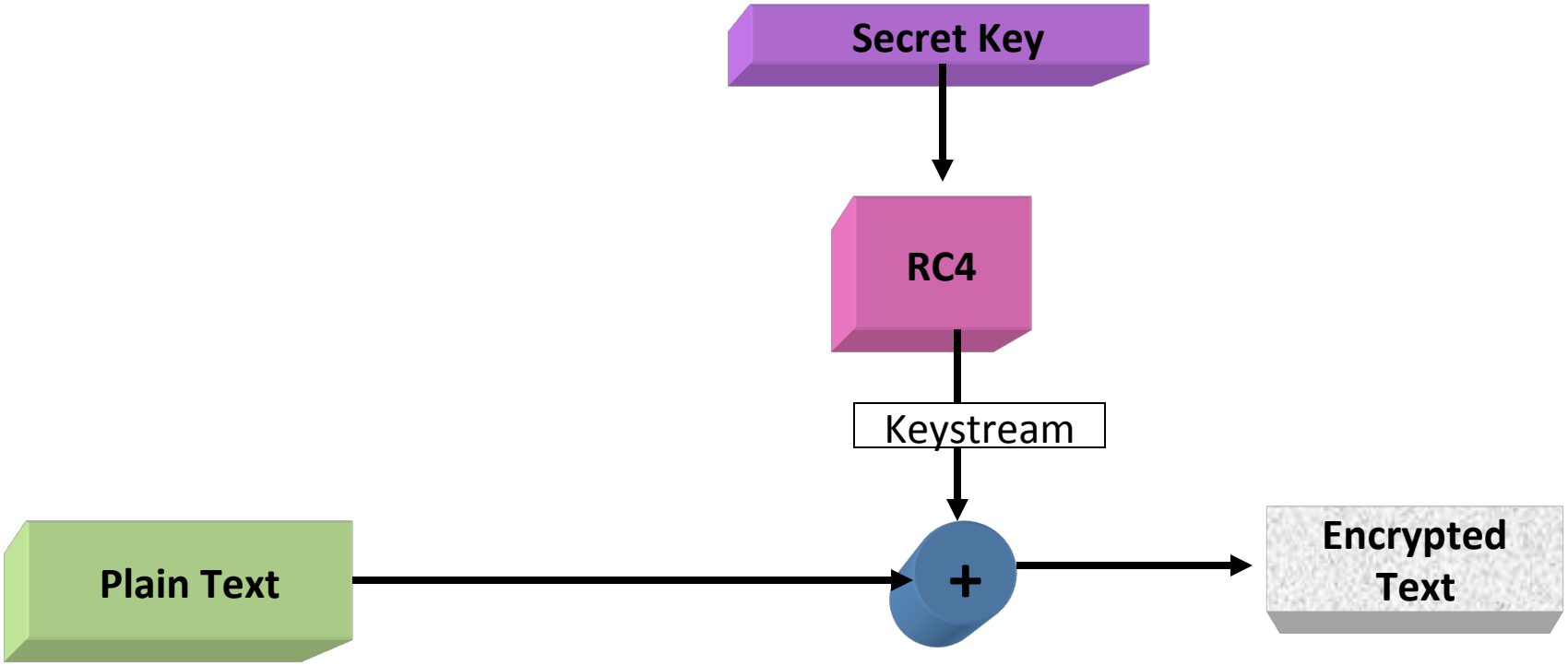
RC4 : Principes

- Algorithme Clef Symétrique inventé Ron Rivest
  - Cipher propriétaire de RSA, gardé secret
  - Code divulgué anonymement dans Cyberpunks mailing list en 1994
  - Plus tard posté sur sci.crypt newsgroup
- **Taille des clefs variable**, stream cipher sur **octets**
  - Normalement utilise des clefs de 64 ou 128 bits.
- Utilisé dans :
  - SSL/TLS (Secure socket, transport layer security) entre les navigateurs web et les serveurs.
  - IEEE 802.11 wireless LAN std: WEP (Wired Equivalent Privacy), WPA (WiFi Protocol Access)

Utilisations basées sur RC4

- WEP
- WPA default
- Bit Torrent Protocol Encryption
- Microsoft Point-to-Point Encryption
- SSL (optionally)
- SSH (optionally)
- Remote Desktop Protocol
- Kerberos (optionally)

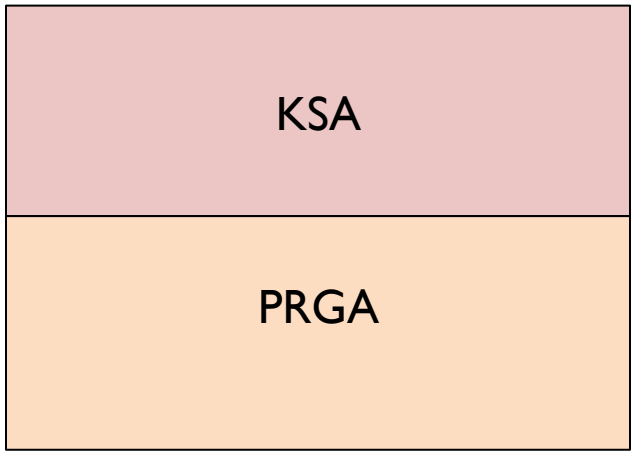
RC4 Block Diagram



Cryptographiquement très robuste et facile à implanter en soft

RC4 …l'intérieur

- 2 parties:
  - Key Scheduling Algorithm (KSA)
  - Pseudo-Random Generation Algorithm (PRGA)



- KSA
  - Génère un tableau d'états
- PRGA sur le KSA
  - Génère le keystream
  - XOR du keystream avec les données pour générer le flux crypté

KSA (key Schedule Algorithm)

- Clef secrète K utilisée pour initialiser et permuter la vecteur d'états S, fait en 2 pas
- Utilise 2 index 8-bits **i** et j qui donnent les cases de S à permuter

1

```
for i = 0 to 255 do
   S[i] = i;
   T[i] = K[i mod(|K|)]);
```

2

```
j = 0;
for i = 0 to 255 do
 j = (j+S[i]+T[i])(mod 256)
 permute (S[i], S[j])
```

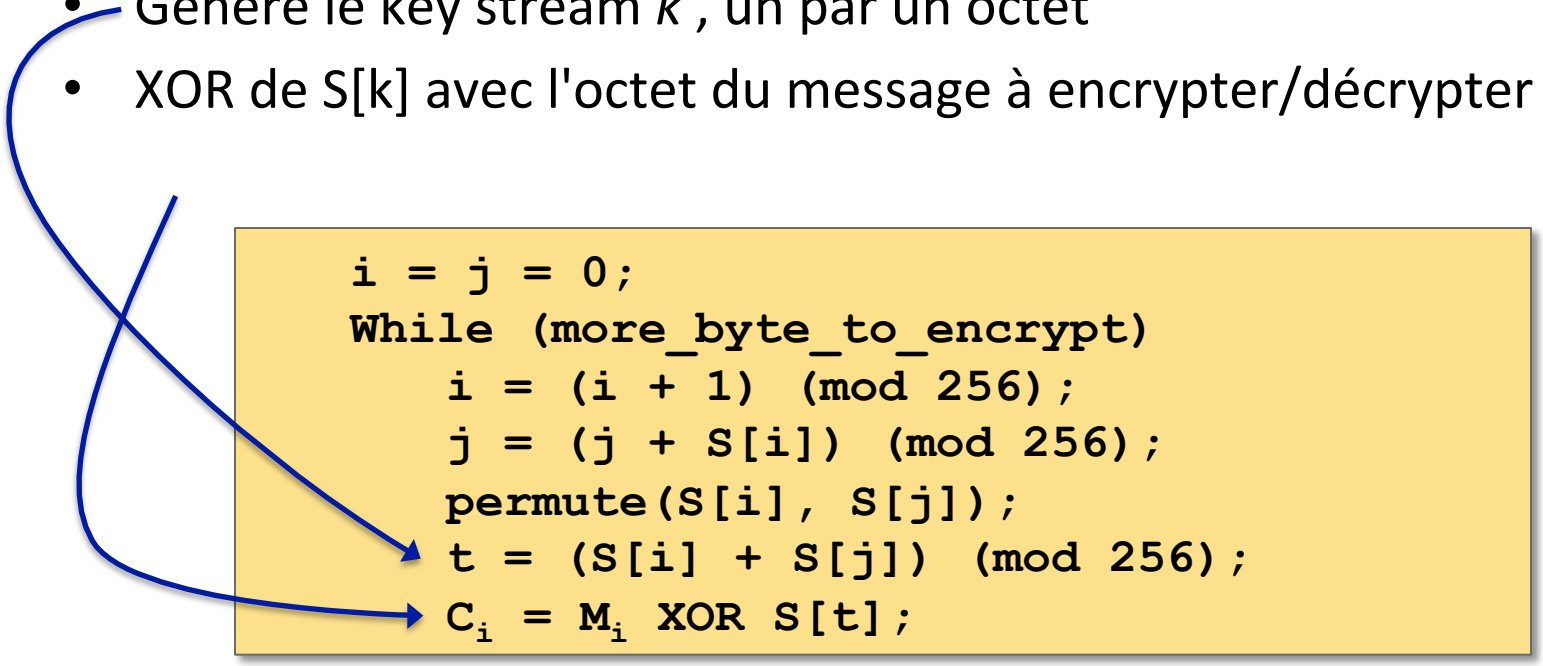
[S], S est l'ensemble des valeurs de 0 à 255
S[0]=0, S[1]=1,…, S[255]=255
[T], Un vecteur temporaire
[K], Tableau d'octets de la clef secrète
|K| = Longueur de (K)

- On utilise T pour produire une permutation initiale de S
- La seule opération sur S est une permutation
- S contient toujours les valeurs de 0 à 255 (mais en vrac)

Après KSA, la clef primaire K et le vecteur temporaire T ne sont plus utilisés

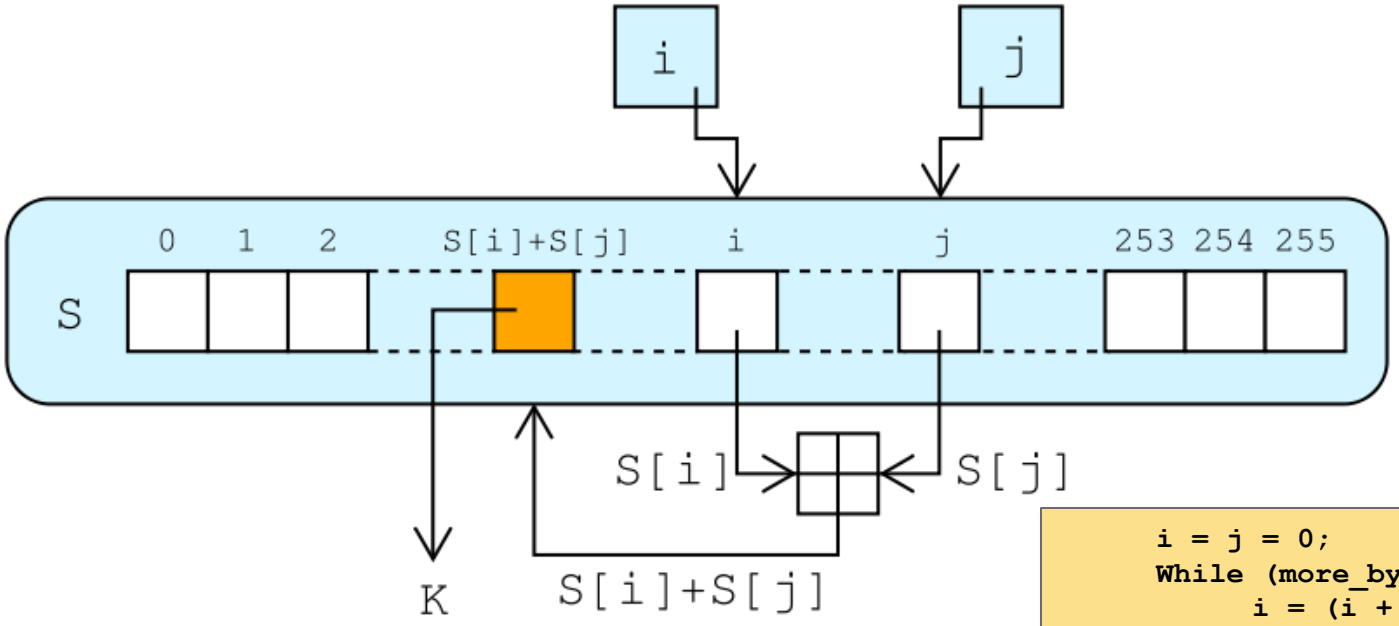
PRGA (Pseudo-Random Generation Algorithm)

- Génère le key stream *k* , un par un octet
- XOR de S[k] avec l'octet du message à encrypter/décrypter



RC4 Lookup Stage

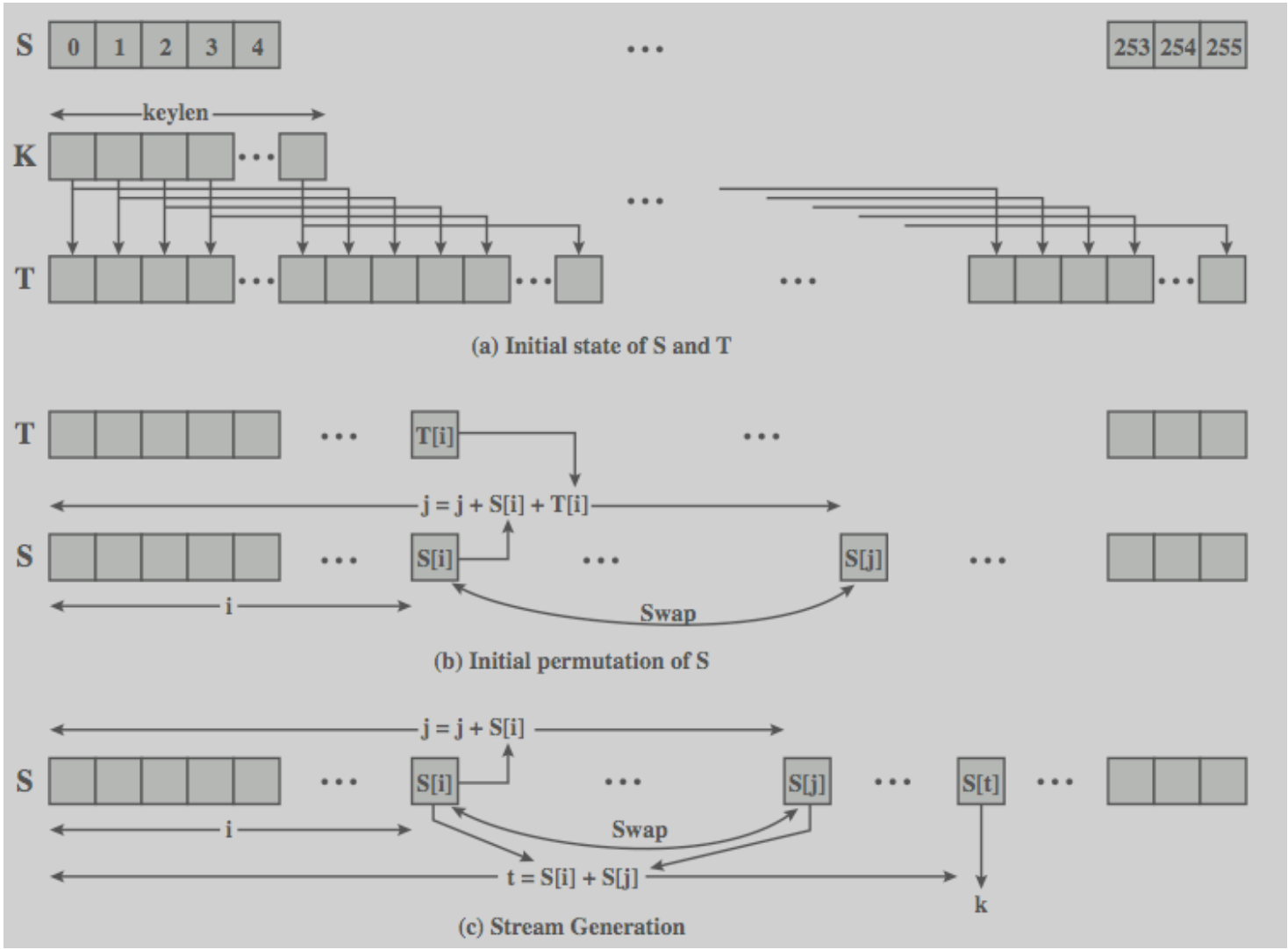
- L'octet de sortie est sélectionné en prenant les valeurs de S[i] and S[j], les additionnant modulo 256, et en regardant la somme dans S
- S [S[i] + S[j]] est utilisé comme l'octet de keystream, *K*



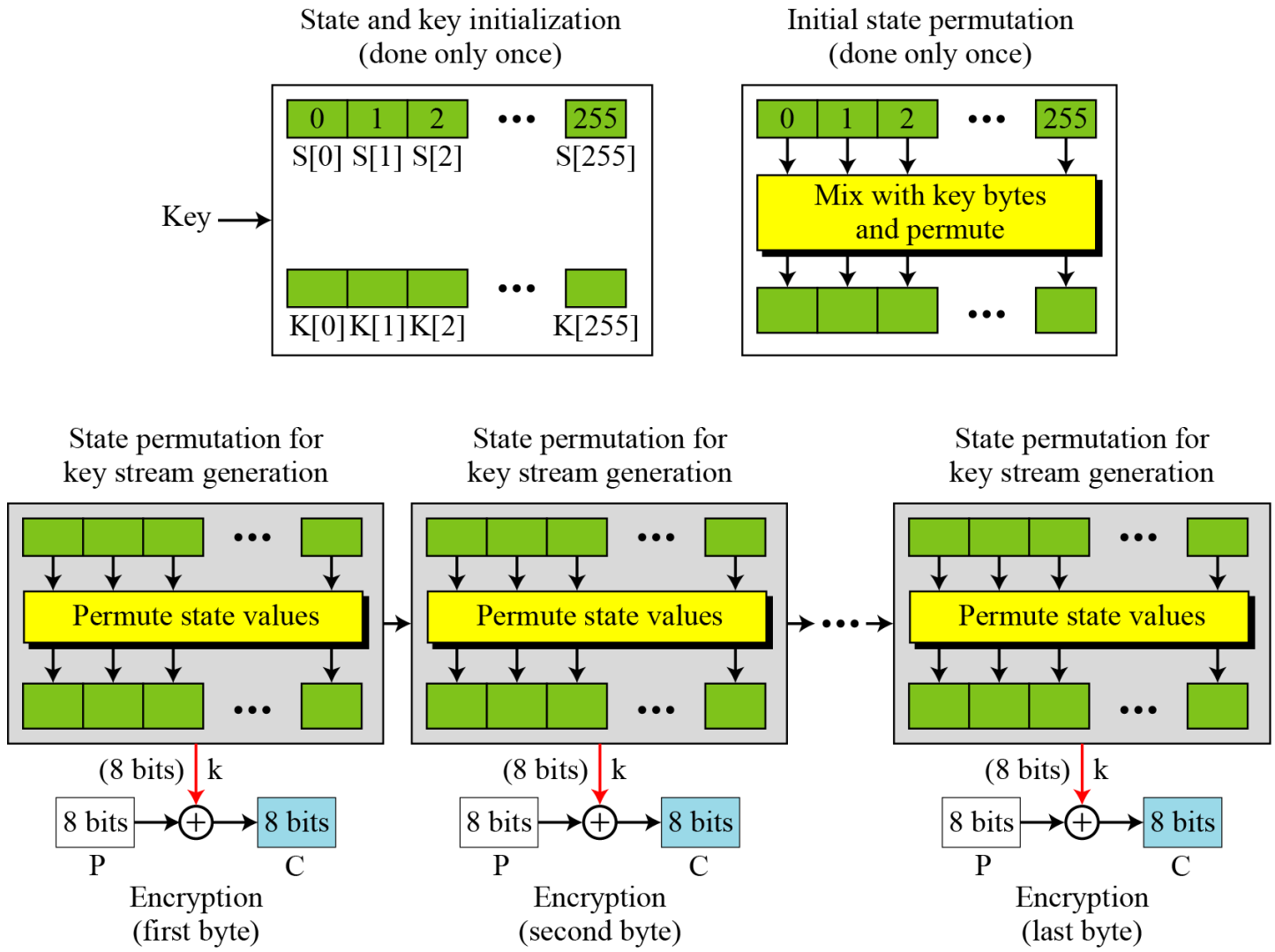
http://en.wikipedia.org/wiki/File:RC4.svg

```
i = j = 0;
While (more_byte_to_encrypt)
      i = (i + 1) (mod 256);
      j = (j + S[i]) (mod 256);
      swap(S[i], S[j]);
      k = (S[i] + S[j]) (mod 256);
      C_i = M_i XOR S[k];
```

Diagramme détaillé



Overall Operation of RC4



Decryption using RC4

- Use the same secret key as during the encryption phase.
- Generate keystream by running the KSA and PRGA.
- XOR keystream with the encrypted text to generate the plain text.
- Logic is simple :

$$(A \text{ xor } B) \text{ xor } B = A$$

A = Plain Text or Data
B = KeyStream

Topics

- One-Time-Pad
- Random Number Generator
- Stream Cipher
- RC4
- **RC4 and WEP**

RC4 and WEP

- WEP is a protocol using RC4 to encrypt packets for transmission over IEEE 802.11 wireless LAN.
- WEP requires each packet to be encrypted with a separate RC4 key.
- The RC4 key for each packet is a concatenation of a 24-bit IV (initialisation vector) and a 40 or 104-bit long-term key.

Exemple

Exemple : suite binaire de longueur 9, s = 001101110 (le premier sorti, s_{N-1})

001101110.

g(x)=1 ; m=-1 ; L=0; h(x)=1
Pour k = 0 -> N-1
  Calculer : d = s_k + Somme pour i de 1 à L de $c_i.s_{k-i}$ mod 2
  Si d = 1
    t(x) = g(x) et g(x) = g(x)+h(x). x^{k-m}
    Si 2L ≤ k alors L=k+1-L, m = k, h(x)=t(x)

k	S_k	d	L	g(x)	m	h(x)
			0	1	-1	1
0	0	0	0	1	-1	1
1	0	0	0	1	-1	1
2	1	1	3	x^3+1	2	1
3	1	1	3	x^3+x+1	2	1
4	0	1	3	x^3+x^2+x+1	2	1
5	1	1	3	x^2+x+1	2	1
6	1	0	3	x^2+x+1	2	1
7	1	1	5	x^5+x^2+x+1	7	x^2+x+1
8	0	1	5	x^5+x^3+1	7	x^2+x+1

Le LFSR de longueur 5 ayant pour $f(x) = X^5 + X^3 + 1$ génère donc la suite donnée.