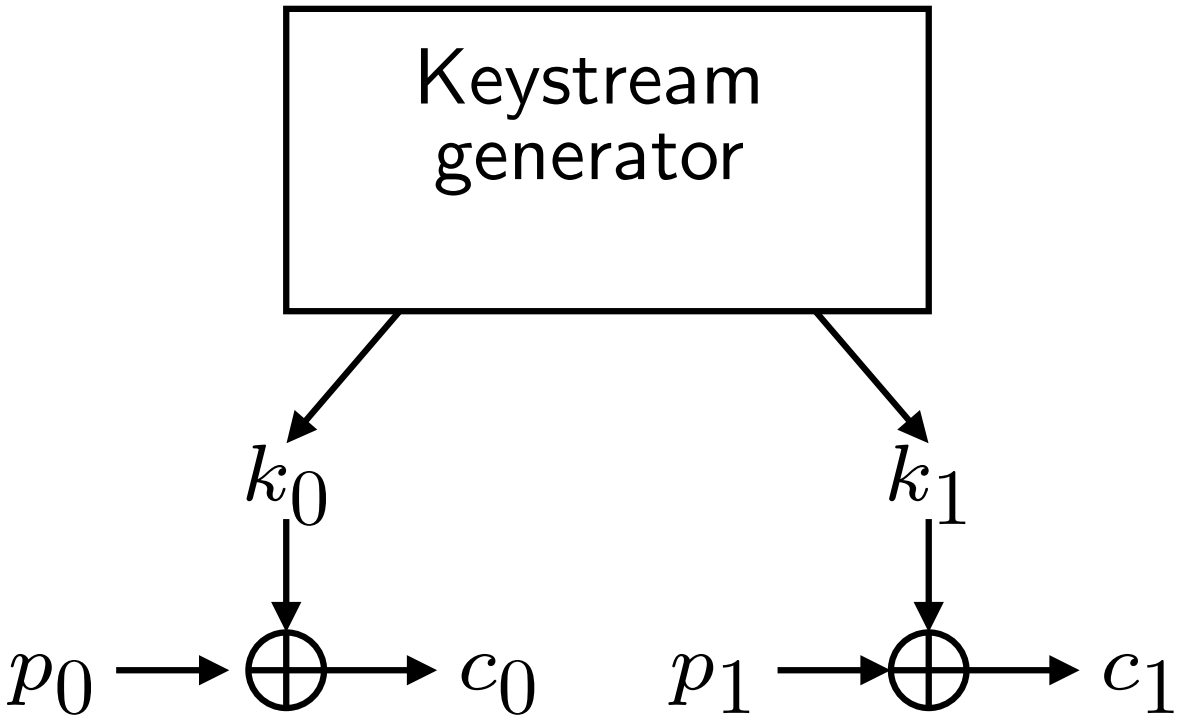
One-Time-Pad



One-Time-Pad

Perfect Secrecy

- $\ell(k) \geq \ell(p)$ implies
  - ▷ knowing $\ell(p)$ to generate k,
  - ▷ or transmit k after p,
  - ▷ the bandwidth of the secure channel (key) is equal to the bandwidth of the insecure channel (message).
- No repetition for k :
  - ▷ k is chosen randomly
  - ▷ then tested to eliminate repetition ($O(\log N)$).
- The One-Time-Pad is not practical !

Perfect security

- Whatever the power of the adversary, it is not possible to recover the plaintext or the key from the ciphertext.

- **Conditions to achieve perfect secrecy :**
  - $I(M;C) = 0$
  - $H(K) \geq H(M)$

- $H(K) \geq H(M)$ is the condition that make the system not practical.

- We need to change the security model !

From perfect secrecy. . .

to computational security

- An encryption scheme achieves computational security of the best cryptanalysis requires no less that n operations such that it can not be executed in practice.
- We limit the adversary power.
- This is **computational security**.

Computational security

Warning

- There is no system such that we have a mathematical proof that the best cryptanalysis costs only n opérations (lower bound).
- **However**, there is scheme for which the best known best known cryptanalysis **connue** cost n operations
- En practice, designers give *a security level* (*complexity for the best cryptanalysis*).

Examples

- GSM : A5/1
- IS95 : Long code generator
- CDMA and TDMA : Oryx
- Bluetooth : E0
- WEP : RC4
- DVDCSS : summation generator
- DVB : Common Scrambling Algorithm
- XBOX : RC4
- ECRYPT ESTREAM

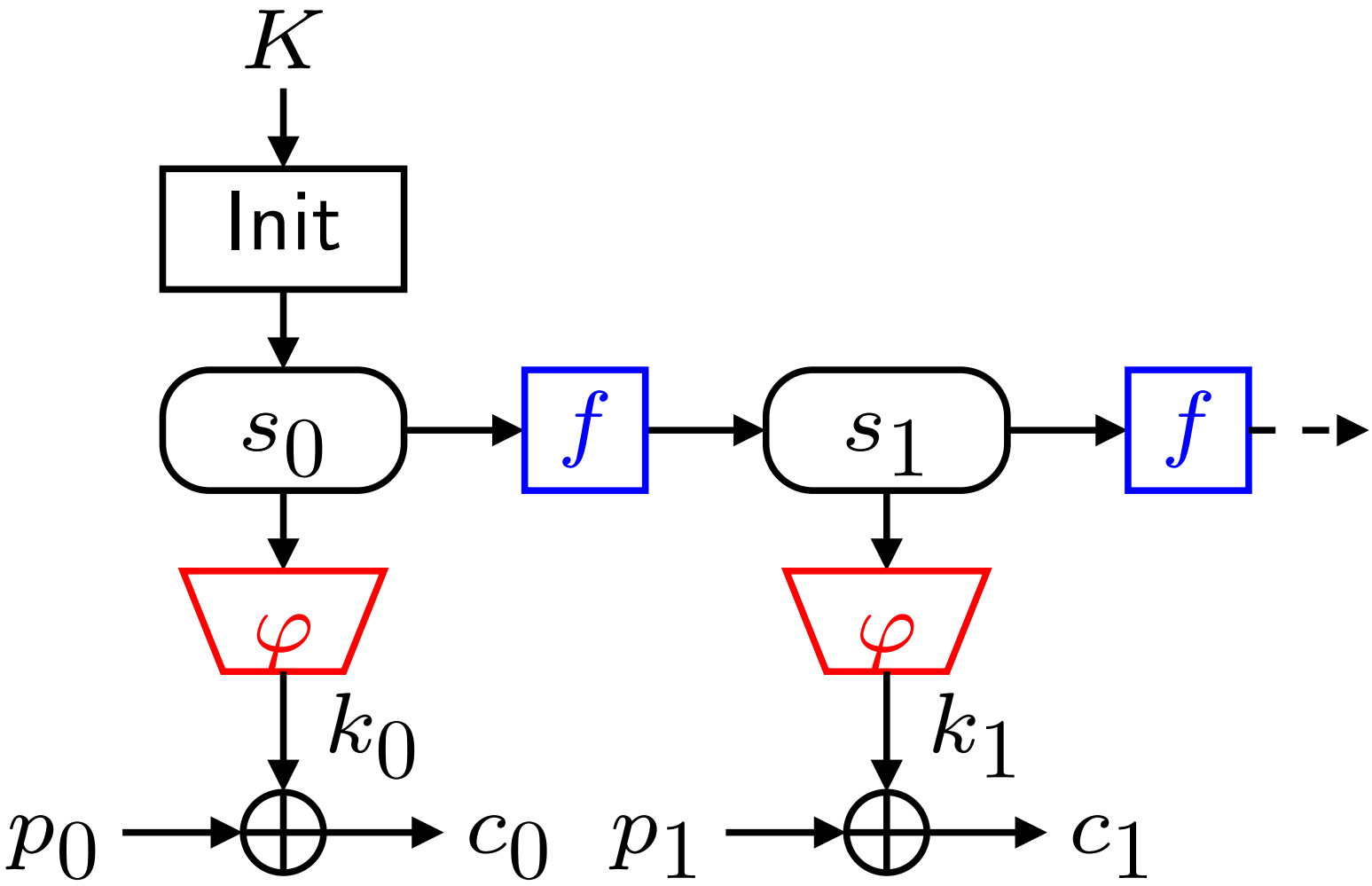
Towards stream ciphers

- We want :
  - a key independent of the size of the message,
  - gkeep the good properties of the one-time-pad,
  - (fast encryption (complexity in $O(1)$).

- How to transform a fixed size key in a arbitrary length keystream ?

how to do it ?

Pseudo-random generators



Pseudo-random number generators (PRNG)

Vocabulary

- A PRNG is finite state machine :
  - the internal state s (ℓ bits),
  - the seed s_0,
  - update function f,
  - filtering function φ (optional).

- A PRNG is deterministic : we have at most 2^ℓ states possible.

Combinatorics

Boolean functions

- A Boolean function is a function

$$g : \{0,1\}^\ell \to \{0,1\}.$$

- A vectorial Boolean function is a function

$$h : \{0,1\}^\ell \to \{0,1\}^m.$$

- **True table** :

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

Combinatorics

- Let φ be a Boolean function : $\varphi : \{0,1\}^\ell \to \{0,1\}$.
- *How many* φ *?*
- Let f be a vectorial Boolean function $f : \{0,1\}^\ell \to \{0,1\}^\ell$.
- *How many* f *?*
- *How do you conclude ?*

Period

- A PRNG produces an infine sequence of values.
- Let $(s_n)_{n\in\mathbb{N}}$ be a sequence of elements in E. $(s_n)_{n\in\mathbb{N}}$ is $periodic$ if there is $p \in \mathbb{N}$ Such that $x_{n+p} = x_n$ for any n.
- ULet $(s_n)_{n\in\mathbb{N}}$ be a sequence of elements in E. $(s_n)_{n\in\mathbb{N}}$ is $ultimately\ periodic$ if there is $n_0 \in \mathbb{N}$ and $p \in \mathbb{N}$ such that $x_{n+p} = x_n$ for any $n \geq n_0$.
- p is $a\ period$ of the sequence.

Period

- The *period of a PRNG* is the smallest $T > 0$ such that :

$$s_{t+T} = s_t.$$

- For an n-bit internal state, we have $T \geq 2^\ell$.

- The period is a problem if we want to encipher long message. . .

Floyd Algorithm (1)

Require: s_0

$\mu \leftarrow 1$

$y_0 \leftarrow s_0$

$s_1 \leftarrow f(s_0)$

$y_1 \leftarrow f(f(y_0))$

while $s_\mu \neq y_\mu$ **do**

$s_\mu \leftarrow f(s_{\mu-1})$ {turtle}

$y_\mu \leftarrow f(f(y_{\mu-1}))$ {hare}

$\mu \leftarrow \mu + 1$

end while

return μ

Floyd Algorithm (2)

- After this step, we are on the **main cycle** of the sequence.
- Moreover, μ is a multiple of T : $\mu = k \cdot T$.
- **Complexity :** $O(\mu)$.

Floyd Algorithm (3)

- To find T, we search $\lambda \neq \mu$ such that

$$s_\lambda = s_{2\lambda}.$$

On aura alors :

$$\begin{cases} \lambda \leq & \mu + T, \\ T = k \cdot (\lambda - \mu). \end{cases}$$

- We have $T = \lambda - \mu$.
- **Complexity :** $O(\lambda)$.

Floyd Algorithm (4)

$\lambda \leftarrow 0$

$s_0 \leftarrow f(s_\mu)$

$y_0 \leftarrow s_0$

$s_1 \leftarrow f(s_0)$

$y_1 \leftarrow f(f(y_0))$

while $s_\lambda \neq y_\lambda$ **do**

$s_\lambda \leftarrow f(s_{\lambda-1})$ {Tortue}

$y_\lambda \leftarrow f(f(y_{\lambda-1}))$ {Lièvre}

$\lambda \leftarrow \lambda + 1$

end while

return $T \leftarrow \mu - \lambda$

Conclusion

- The complexity of the Floyd algorithm is $O(T)$.
- It is therefore impossible to work on f by looking to random generators.
- We need to find generators with provable period.

Linear congruential generators

- Proposed by Lehmer :

$$s_{t+1} = (a \cdot s_t) + c \bmod m$$

- His first choice for the parameters :
  — m prime,
  — $c = 0$.

- We need to choose a.

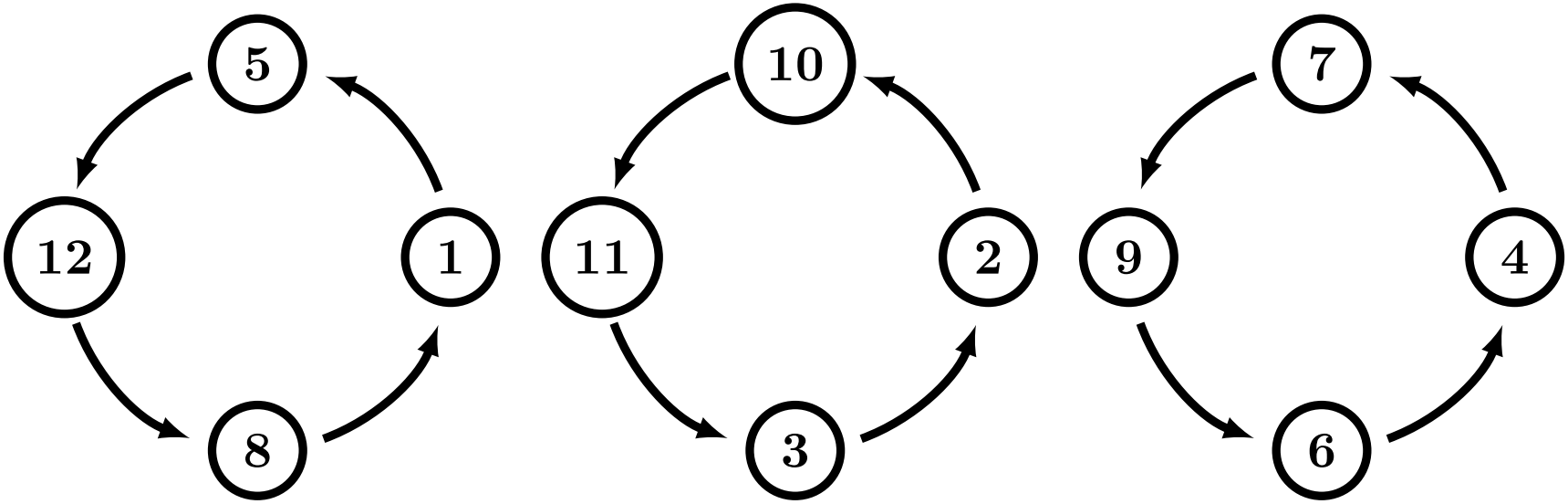
Fixed Points

- $a = 0 \rightarrow \forall t,\ f(s_t) = 0 \rightarrow$ **fixed point**.
- $a = 1 \rightarrow \forall t,\ f(s_t) = s_0 \rightarrow$ **fixed point**.
- $a = m - 1$, then we have :

$$s_1 = s_0 \cdot (m-1) \bmod m$$

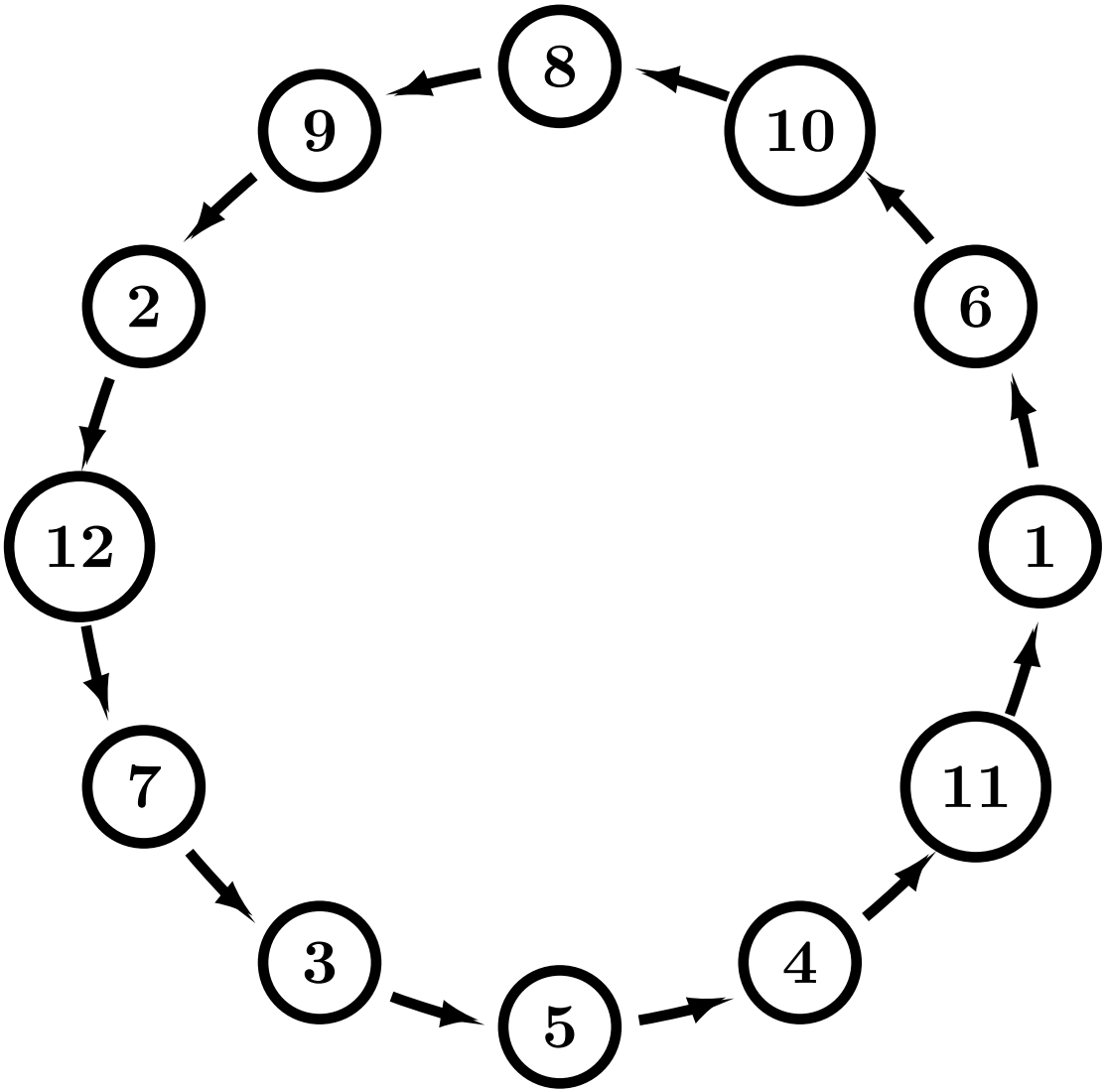
- $s_0 = 0$ est une $bad\ seed$.

$$s_{t+1} = 5 \cdot s_t \bmod 13$$



- We have **3 cycles of length 4.**

$s_{t+1} = 6 \cdot s_t \bmod 13,$ **and** $(x_0 = 1)$



Park-Miller generator

- Park and Miller have proposed to use :

$$x_{t+1} = 16807 \times x_t \bmod 2^{31} - 1.$$

- Still used today :
  — bibliothèque *Apple Carbon*,
  — Contiki.

Implementation

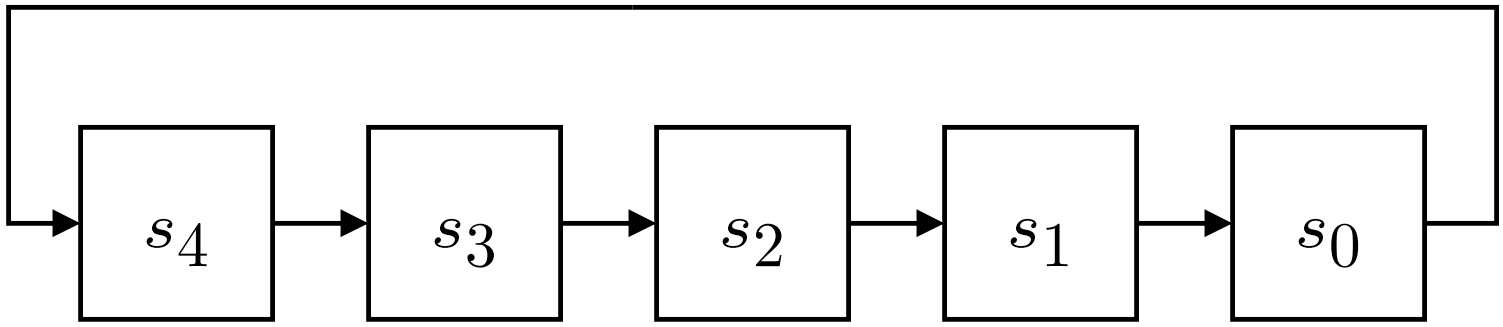
- Be careful :

$$x_{t+1} = 16807 \times x_t \bmod 2^{31} - 1.$$

- **Overflow**. the result is on $2m$ bits !

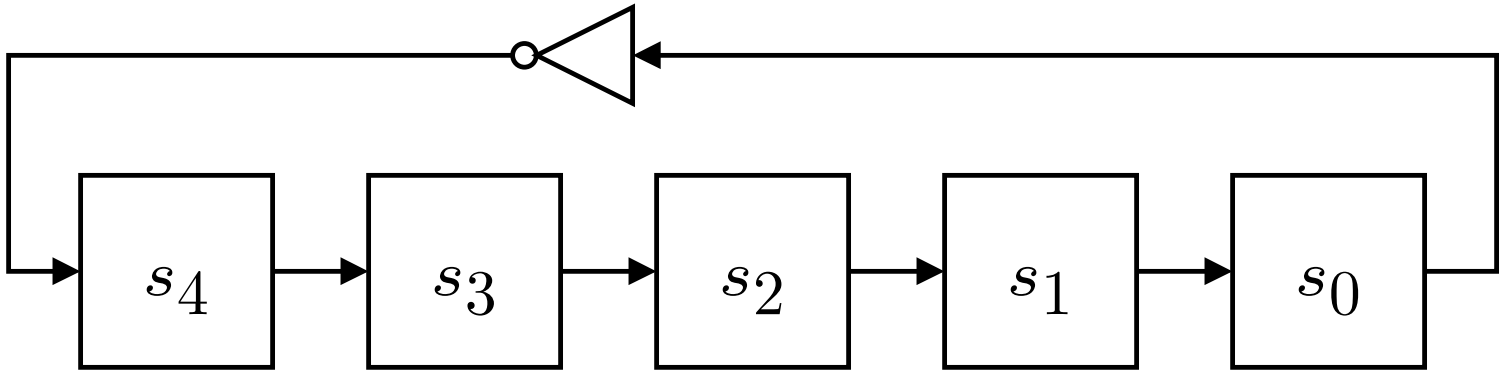
$$x_{t+1} = 16807 \times x_t \bmod 2^{31} - 1$$
$$\neq$$
$$x_{t+1} = (16807 \times x_t \bmod 2^k) \bmod 2^{31} - 1$$
$$\text{if } k \leq 46$$

Ring register



	s_4	s_3	s_2	s_1	s_0
0	0	0	0	0	1
1	1	0	0	0	0
2	0	1	0	0	0
3	0	0	1	0	0
4	0	0	0	1	0

Johnson counter



	s_0	s_1	s_2	s_3	s_4
0	0	0	0	0	0
1					
2					
3					
4					

	s_0	s_1	s_2	s_3	s_4
5					
6					
7					
8					
9					

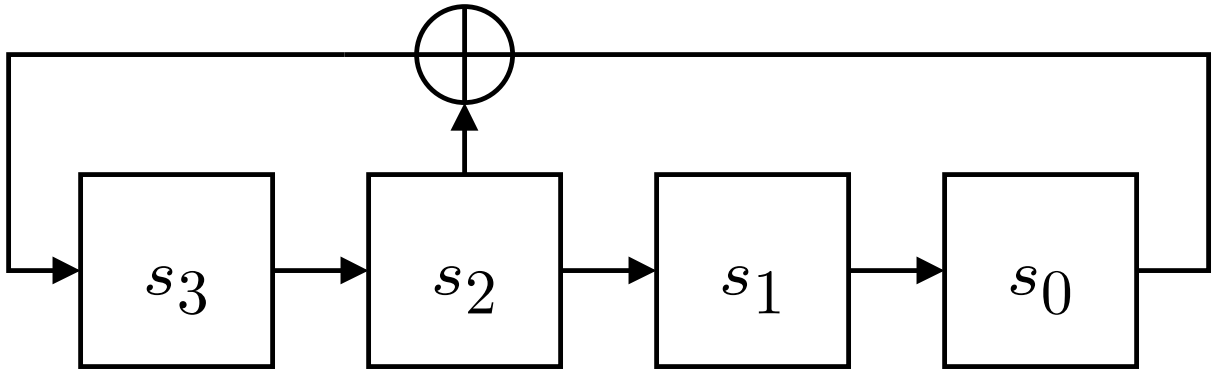
Golomb's theorem

- The cycle of a PRNG is without branch pointif and only if it can be written :

$$f(s_m, s_{m-1}, \cdots, s_{m-\ell+1}) = f_1(s_{m-1}, \cdots, s_{m-\ell+1}) \oplus s_m.$$

- We reduce the space from 2^{2^n} to $2^{2^{n-1}}$ update functions.

Linear feedback shift register



	s_3	s_2	s_1	s_0
0	0	0	0	1
1				
2				
3				
4				
5				
6				

Linear feedback shift register

- We denote : $(s_0, \cdots, s_{\ell-1})$ lthe internal state of the register.
- $(a_1, \cdots, a_\ell)$ are the feedback coefficients.
- $a_i \in \mathbb{F}_2$ et $s_i \in \mathbb{F}_2$ pour $i \in [1, n]$
- The feedback function f is defined by :

$$s_{m+\ell} = a_1 s_{m+\ell-1} \oplus a_2 s_{m+\ell-2} \oplus \cdots \oplus a_\ell s_m$$

Feedback polynomial

- We call feedback polynomial of an LFSR, the polynomial defined by :

$$p(X) = 1 + \sum_{i=1}^{\ell} a_i X^i \ \in \mathbb{F}_2[X].$$

- Let $(s_n)_{n\geq 0}$ the ultimately periodic sequence of the register. The formal sum associated to the sequence :

$$s(X) = \sum_{n\geq 0} s_n X^n \ \in \mathbb{F}_2[X]$$

Feedback polynomial

- $s(X)$ can be expressed as :

$$s(X) = \frac{g(X)}{p(X)},$$

with $g(X) \in \mathbb{F}_2[X]$ and deg $g <$ deg p

Proof

- We have $g(X) = s(X) \cdot p(X)$ with :

$$g(X) = \left(\sum_{n\geq 0} s_n X^n\right) \cdot \left(1 + \sum_{i=1}^{\ell} a_i X^i\right)$$

$$= \sum_{i=0}^{\infty} \left(\sum_{j=0}^{i} a_{i-j} s_j\right) X^i$$

We have $\sum_{j=0}^{i} a_{i-j} s_j = 0$ for $i \geq \ell$, we obtain $g(X)$:

$$g(X) = \sum_{i=0}^{\ell-1} \left(\sum_{j=0}^{i} a_{i-j} s_j\right) X^i$$

Minimal Feedback polynomial

- The minimal feedback polynomial of $(s_n)_{n\geq 0}$ is the feedback polynomial with the lowest possible degree which can generate $(s_n)_{n\geq 0}$.

- The linear complexity of sequence is the degree of the *minimal feedback polynomial*.

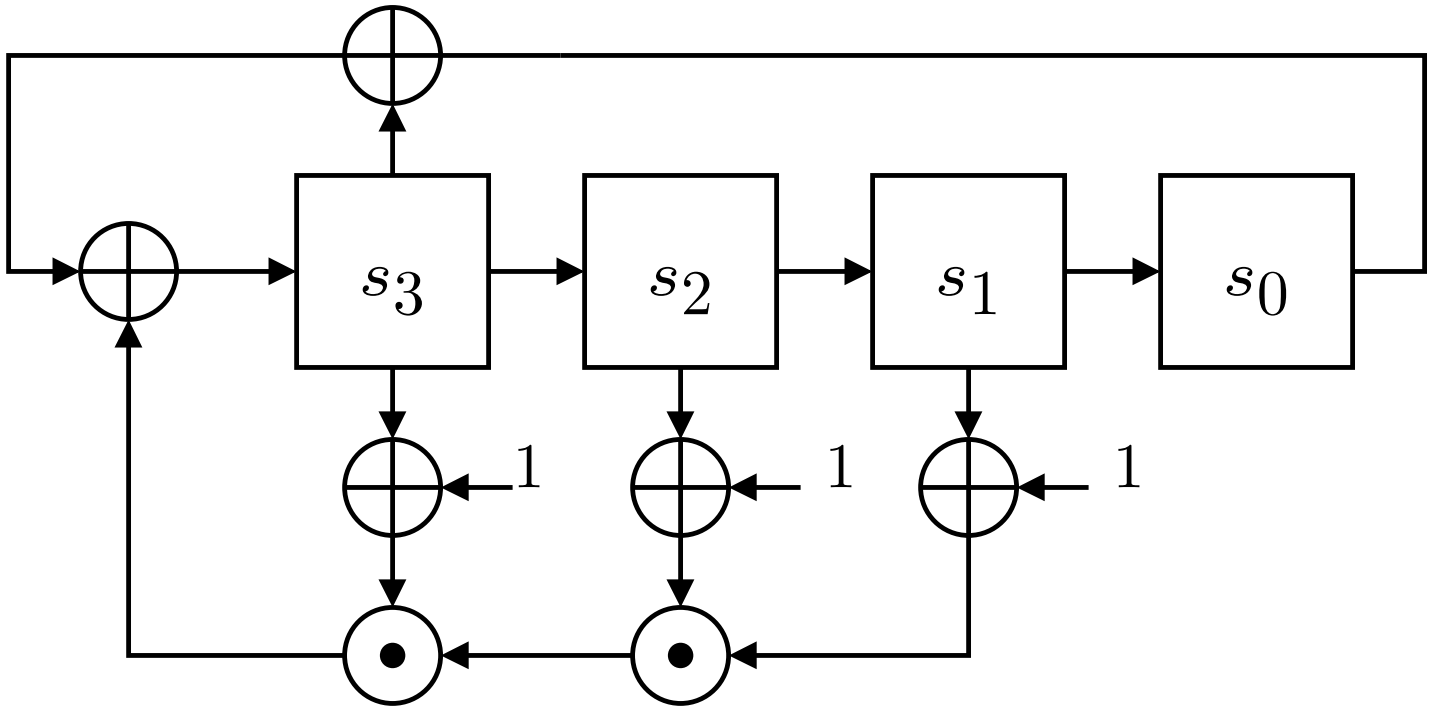
Minimal Feedback polynomial

- A polynomial $p(X)$ of degree $\ell > 0$ is *irréductible* if all the divisors of p are a constant or a product of p by a constant.
- A polynomial $p(X)$ of degree $\ell > 0$ over $\mathbb{F}_2$ is *primitive* if it is *irreducible* and

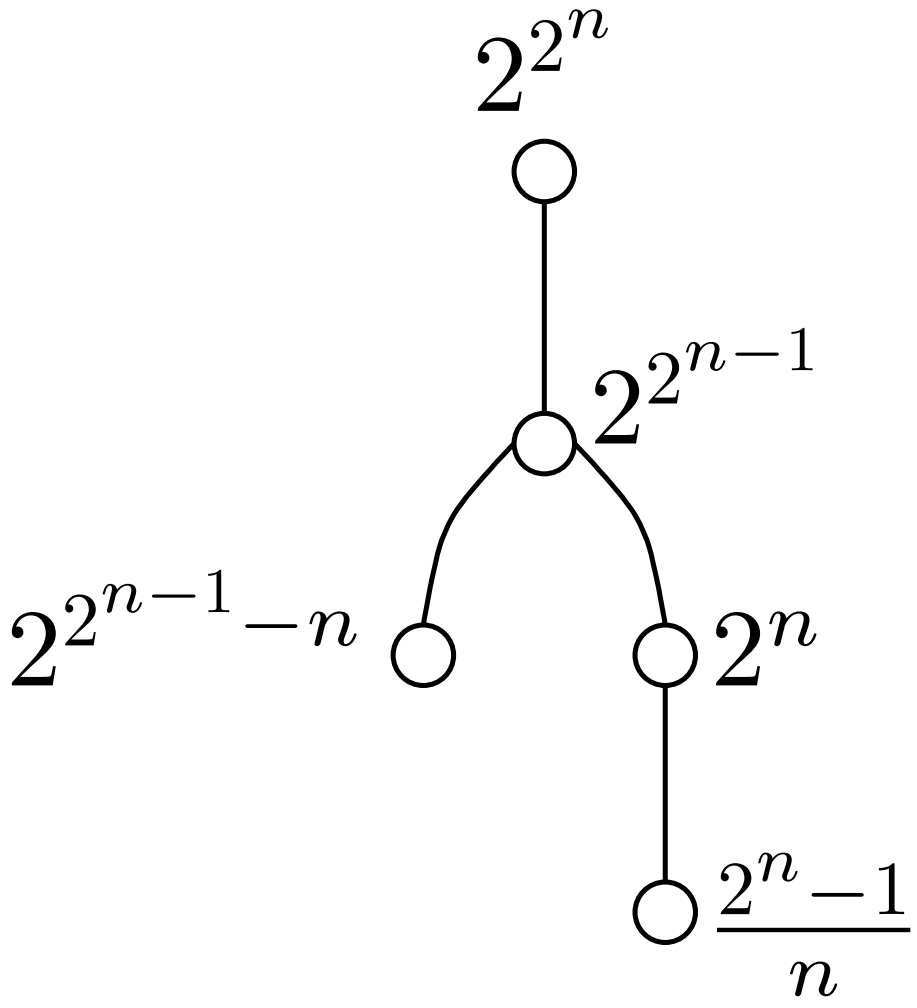
$$\min \{i > 0 | X^i = 1 \bmod p\} = 2^\ell - 1.$$

- An LFSR of length ℓ has period $2^\ell - 1$ if and only if its feedback polynomial is primitive and its initial state is not 0.

de Brujin Sequences



Sequence



Matrix view

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 1 \\ a_\ell & a_{\ell-1} & \cdots & a_3 & a_2 & a_1 \end{pmatrix}$$

- For the register, we have :

$$R_{i+1} = AR_i.$$

Matrix view

- We observe that :

$$R_{i+n} = A^n R_i.$$

- The characteristic polynomial of A is defined by :

$$\tilde{p}(X) = \det(Id - AX).$$

- We have : $\tilde{p}(X) = X^n p(X^{-1})$.

Statistical Properties

- Golomb 1982 :
  — the sequence of an LFSR is balanced ;
  — les séries sont équitables réparties ;
  — la fonction d'autocorrélation prend 2 valeurs.

- Attention ces propriétés ne garantissent aucunement la sécurité de la séquence !

Equilibre

- Dans chaque période, le nombre de 0 est approximativement égal au nombre de 1 :

$$\left| \sum_{i=0}^{T-1} (-1)^{s_i} \leq 1 \right| .$$

- Dans une suite de longueur maximale $2^\ell - 1$, toute suite $(s_i, s_{i+1}, \cdots, s_{i+\ell-1})$ de n éléments non tous nuls apparaît **une et une seule fois par période**.

Equilibre

- **La démonstration est simple :** un registre donné $R_i = (s_i, s_{i+1}, \cdots, s_{i+\ell-1})$ ne peut apparaître qu'une fois par période. Or il y a $2^\ell - 1$ états par période et les registres prennent au plus $2^\ell - 1$ valeurs. Donc ils prennent toutes les valeurs une fois.

- **Question :** combien y a t'il d'états ayant la valeur s_i à 1. Même question pour 0. Conclusion ?

Série

- Une série (de 0 ou de 1) est une succession de bits identiques, maximale (i.e. encadrée par des bits opposés).

- Soit S l'ensemble des séries. Dans chaque période, si $2^k \leq |S| < 2^{k+1}$, on trouve $|S|/2$ séries de longueur 1, $|S|/4$ séries de longueur 2, . . . , $|S|/2^k$ séries de longueur k, et pour chaque longueur, autant de séries de 0 que de séries de 1.

Série

- **Démonstration.** Comptons le nombre d'occurence d'une série d'exactement k zéros. Cela revient à compter les occurences de $(1, \underbrace{0, \cdots, 0}_{k}, 1)$. Comme tous les états $R_i = (s_i, s_{i+1}, \cdots, s_{i+\ell-1})$ apparaissent une et une seule fois, un registre composé de

$$(1, \underbrace{0, \cdots, 0}_{k}, 1, s_{i+k+2} \cdots, s_{i+\ell-1}),$$

apparaît $2^{\ell-k-2}$.

Cryptanalysis

- The goal of a **key recovery attack**, is to recover **the secret key** from the knowledge of the keystream.
- The most powerful attack !

Cryptanalysis

- The goal of a **state recovery attack**, is to recover the seed of the stream cipher.
- Knowing the key implies to know the seed, but the opposite is not true.

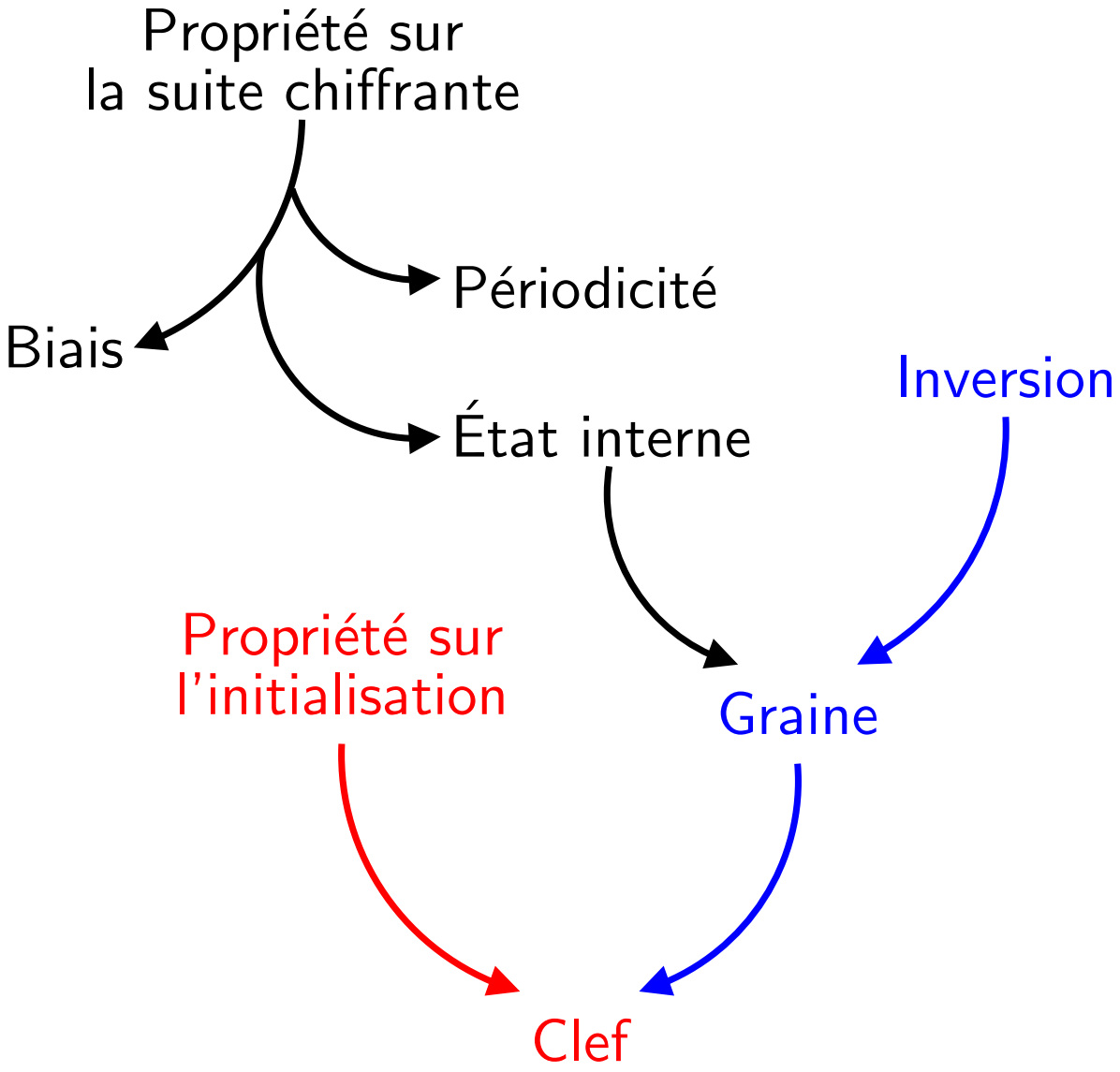
Cryptanalysis

- A **next-bit prediction attack** consists from the observation of the keystream to produce new bits of the keystream.

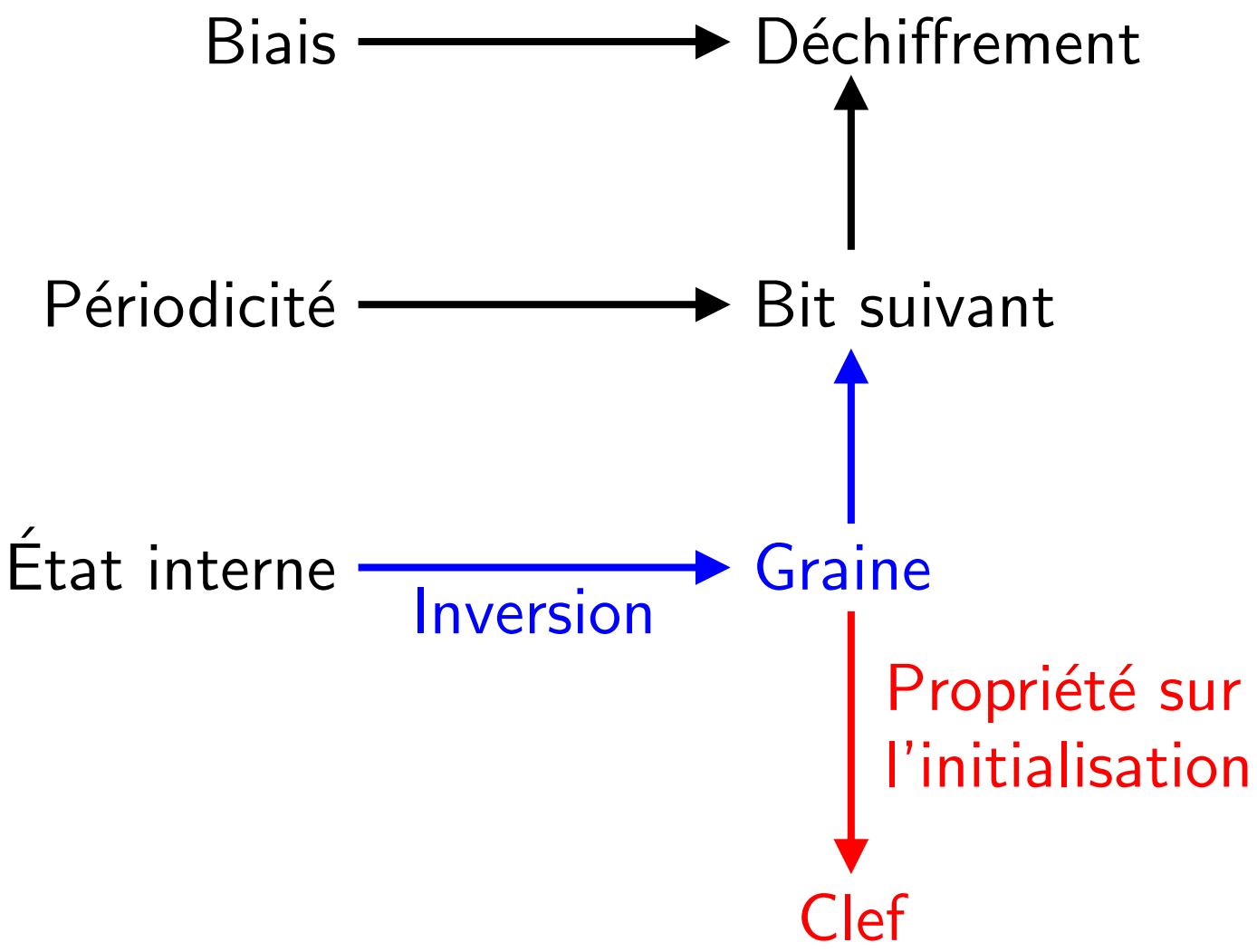
Cryptanalysis

- A distinguishing attack determine if a sequence of the stream cipher can be distinguished from a true random sequence.

Attack on stream ciphers



Impact


Biais
Déchiffrement
Périodicité
Bit suivant
État interne
Graine
Inversion
Propriété sur
l'initialisation
Clef

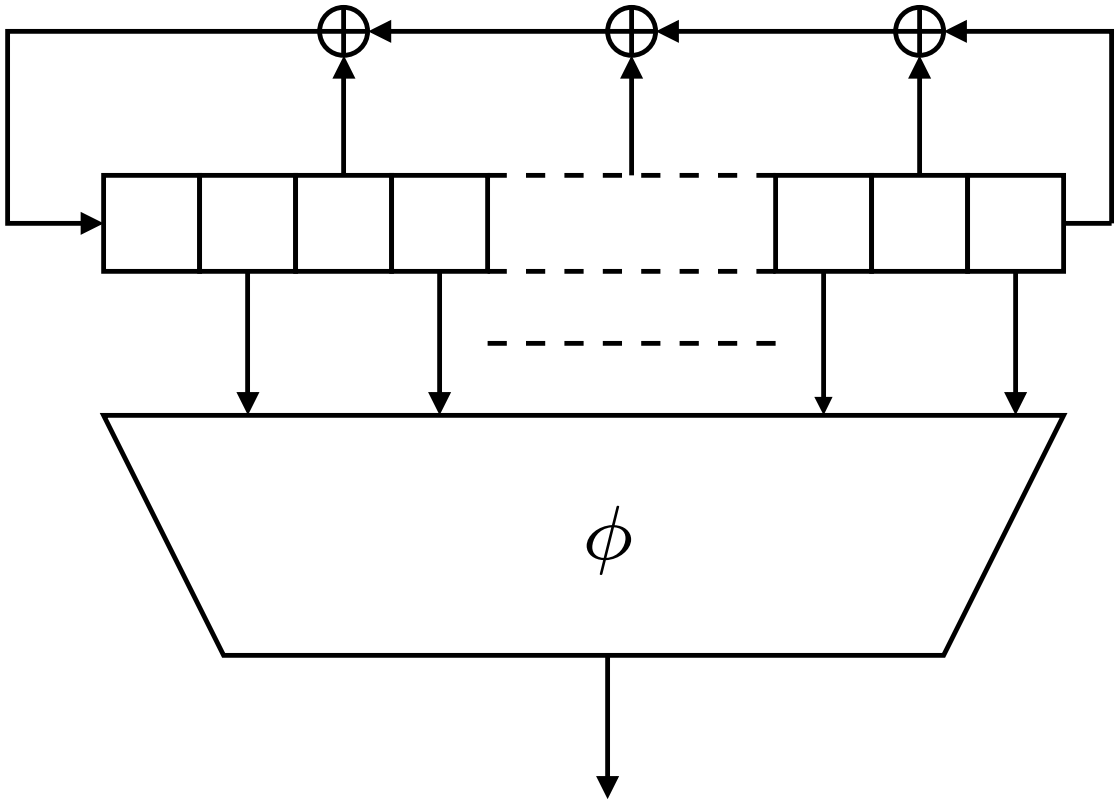
Reconstruction by Gaussian Elimination

- We have the following set of equations :

$$\begin{aligned}
s_\ell &= c_1 s_{\ell-1} \oplus c_2 s_{\ell-2} \oplus \cdots \oplus c_\ell s_0 \\
s_{\ell+1} &= c_1 s_\ell \oplus c_2 s_{\ell-1} \oplus \cdots \oplus c_\ell s_1 \\
s_{\ell+2} &= c_1 s_{\ell+1} \oplus c_2 s_\ell \oplus \cdots \oplus c_\ell s_2 \\
&\vdots \quad \vdots \quad \vdots \\
s_{2\ell-1} &= c_1 s_{2\ell-2} \oplus c_2 s_{2\ell-1} \oplus \cdots \oplus c_\ell s_{\ell-1}
\end{aligned}$$

- **Complexity :** $O(\ell^3)$.
- Faster : Berlekemp-Massey $O(\ell^2)$.

Filtered generator



- How to choose φ ?

Boolean Functions

Elementary

not

a	$\neg a$
0	1
1	0

$a \vee b$

a/b	0	1
0	0	1
1	1	1

$a \wedge b$

a/b	0	1
0	0	0
1	0	1

$a \oplus b$

a/b	0	1
0	0	1
1	1	0

Boolean Functions

Properties

Associativity : $(a \wedge b) \wedge c = a \wedge (b \wedge c),$

$(a \vee b) \vee c = a \vee (b \vee c).$

Commutativity : $a \wedge b = b \wedge a,$

$a \vee b = b \vee a.$

Distributivity :

$a \wedge (b \vee c) = (a \wedge c) \vee (a \wedge b)$

$a \vee (b \wedge c) = (a \vee c) \wedge (a \vee b)$

idempotent : $a \wedge a = a$

$a \vee a = a$

Boolean Functions

Properties

Double negation : $\neg\neg a = a$

DeMorgan's Law : $\neg(a \wedge b) = \neg a \vee \neg b,$

$\neg(a \vee b) = \neg a \wedge \neg b.$

Absorption : $a \vee (a \wedge b) = a,$

$a \wedge (a \vee b) = a.$

Constants : $a \wedge 0 = 0,$

$a \vee 1 = a.$

Opposite : $a \wedge \neg a = 0,$

$a \vee \neg a = 1.$

Polynomials in $\mathbb{F}_2[X_1, \cdots, X_n]$

- A polynomial of $\mathbb{F}_2[X_1, \cdots, X_n]$ is a finite sum of monomials :

$$X_1^{a_1} X_2^{a_2} \cdots X_n^{a_n} \text{ avec } a_1, \cdots, a_n \geq 0.$$

- **Example :** $X_1^3 + X_1^2 X_2 + X_1^5 X_3^7$ is an element of $\mathbb{F}_2[X_1, X_2, X_3]$.

Polynomials in $\mathbb{F}_2[X_1, \cdots, X_n]$

- The degree of a monomial $X_1^{a_1} X_2^{a_2} \cdots X_n^{a_n}$ is defined by :

$$\deg(X_1^{a_1} X_2^{a_2} \cdots X_n^{a_n}) = \sum_{i=1}^{n} a_i.$$

- The **degree of a polynomial** is the maximum degree of his monomials.
- Determine the degree of $X_1^3 + X_1^2 X_2 + X_1^5 X_3^7$.

Polynomials in $\mathbb{F}_2[X_1, \cdots, X_n]$

- Polynomials in $\mathbb{F}_2[X_1, \cdots, X_n]$ can be used to construct Boolean function $\{0,1\}^n$ (0 and 1 are the element of $\mathbb{F}_2$).

Algebraic Normal Form

- A reduced polynomial has all its monomials such that :

$$X_1^{a_1} X_2^{a_2} \cdots X_n^{a_n} \text{ avec } a_1, \cdots, a_n = 0 \text{ or } 1.$$

- The following polynomial is reduced :

$$X_1 X_3 + X_2 X_3 + X_3.$$

- Any Boolean function can be written as a reduced polynomial.

Algebraic normal form (ANF)

- The *algebraic normal form* of a Boolean function f is a polynomial of the form :

$$f(x_0, \cdots x_{n-1}) = \sum_{j \in \mathbb{F}_2^n} a_i \prod_{i=0}^{n} x_i^{j_i},$$

with

— $a_i \in \mathbb{F}_2$,

— $j = (j_0, \cdots, j_{n-1})$ and $\forall i < n, j_i \in \mathbb{F}_2$.

Algebraic normal form (ANF)

- Let $\mathbf{A} = (a_0, \cdots, a_{n-1})$, with $a_i \in \mathbb{F}_2$ the vector defining the monomials of the ANF.

Require: f the true table of f.

$f_{0,b} \leftarrow f(b), \forall b \in \mathbb{F}_2^n$

for $k = 0$ **to** n **do**

for $c = 0$ **to** $2^{n-k-1} - 1$ **do**

$f_{k+1,c} = (f_{k,2c} f_{k,2c+1} + f_{k,2c})$

end for

end for

return $\mathbf{A} = f_{n,0}$

Hamming Weight

- The *Hamming weight* w_H of $x = (x_0, \cdots, x_{n-1})$ is defined by :

$$w_H(x) = \sum_{i=0}^{n-1} x_i.$$

- La *distance de Hamming* d entre 2 vecteurs x et y est définie par :

$$d(x, y) = w_h(x \oplus y).$$

Support

- The support de x is the set of the position for which the coordinates are not 0 :

$$\mathrm{supp}(x) = \{i \in \{0, \cdots, n-1\}, x_i \neq 0\}.$$

- All the previous definitions can be applied to functions too.

Application to functions

- The support of f is the set of the position for which the function is not 0 :

$$\mathrm{supp}(x) = \{x \in \mathbb{F}_2^n, f(x) \neq 0\}.$$

- The *distance* d between 2 functions f and g is defined by :

$$d(f, g) = w_h(f + g).$$

Example

- Let f be a Boolean function with 3 variables and its algebraic normal form is given by :

$$f(x) = f(x_0, x_1, x_2) = x_1 + x_2 + x_2x_3 + x_1x_2x_3.$$

- We have the following properties :
  — $deg(f) = 3$,
  — $w_H(f) = 3$,
  — $\mathrm{supp}(f) = \{(1,0,0),(0,1,0),(1,0,1)\}$.

Weight of a function

- The weight $\mathcal{F}$ of a Boolean function is defined by :

$$\begin{aligned} \mathcal{F}(f) &= \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x)} \\ &= 2^n - 2w_H(f). \end{aligned}$$

Walsh-Hadamard Transform

- The **Walsh-Hadamard transform** of f is denoted $\mathcal{F}(f+\varphi(\cdot))$ and is defined by :

$$\begin{aligned}\mathcal{F}(f+\varphi(\cdot)):\ \mathbb{F}_2^n &\to\ \mathbb{Z}\\ a &\to\ \sum_{x\in\mathbb{F}_2^n}(-1)^{f(x)\oplus a\cdot x}\end{aligned}$$

- $\mathcal{F}(f+\varphi_a)$ is the *Walsh coefficient* of f in a. The set $\{\mathcal{F}(f+\varphi_a), a\in\mathbb{F}_2^n\}$ is called the *Walsh spectrum* of f.

Distinguishing attacks

- if the true table of φ has more 0 than 1 then we have a bias and we can mount a **distinguishing attack**.
- If φ has a **bias** ϵ then an adversary can distinguish the keystream from a genuine random sequence.
- $\rightarrow$ notion of balancedness pour φ.

Balanced

- A function f is said **balanced** if and only if $\mathcal{F}(f) = 0$.

- The linear functions are all balanced :

$$f(s_0, \cdots, x_{\ell-1}) = a_0 s_0 \oplus \cdots \oplus a_{\ell-1} x_{\ell-1}.$$

Algebraic attacks

Shannon

- From the keystream, we can build the following system of equations :

$$\begin{aligned} k_1 &= \varphi(s_0, \cdots s_{\ell-1}) \\ k_2 &= \varphi(f(s_0, \cdots s_{\ell-1})) \\ k_3 &= \varphi(f(f(s_0, \cdots s_{\ell-1}))) \\ &\vdots \qquad\quad \vdots \end{aligned}$$

- What can we do ?

Linearization

- The degree of the monomials is greater than 1, thus we can not apply Gaussian elimination directly.
- For monomial of degree greater than 1, we can associate a new variable and re-write the system !
- How many monomials do we have to re-write ?

Example

- We consider $\mathbb{F}_2[x_1, x_2]$.
  We have the following system :

$$\begin{aligned} x_1 \cdot x_2 &= 0 \\ x_1 \cdot x_2 + x_1 &= 0 \\ x_2 + x_1 &= 1 \end{aligned}$$

- We rewrite :

$$(w_1, w_2, w_3) = (x_1, x_2, x_1 \cdot x_2)$$

Exemple

- We obtain :

$$\begin{aligned} w_3 &= 0 \\ w_3 + w_1 &= 0 \\ w_2 + w_1 &= 1 \end{aligned}$$

$$\begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

Linearization

Complexity

- Time complexity is about

$$\left(\sum_{i=0}^{d}\binom{\ell}{i}\right)^{3}.$$

- Data Complexity ?
- Memory complexity ?

Linearization

- Linéarization attacks are more efficient than exhaustive search if

$$\deg(\varphi) \leq 0.42 \left(\frac{|K|}{1 + \log_2 |K|} \right)$$

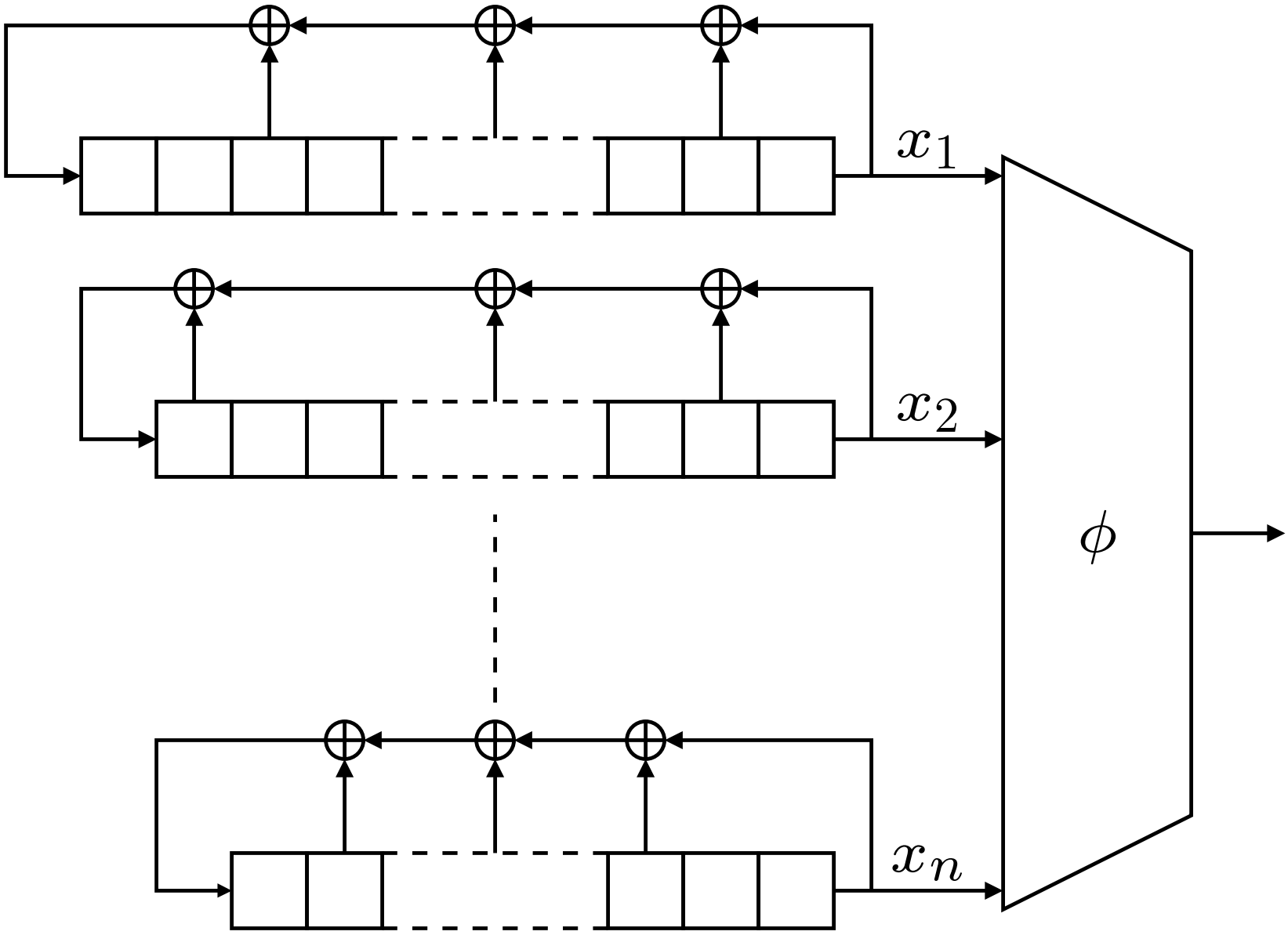
Non-linearity

- The non-linearity of f is defined by :

$$\begin{aligned}\mathcal{N}(f) &= \min_{a\in\mathbb{F}_2^n}(w_H(f+\varphi_a), w_H(f+\varphi_a+1)) \\ &= 2^{n-1} - \frac{\mathcal{L}(f)}{2}.\end{aligned}$$

- with $\mathcal{L}(f) = \max_{a\in\mathbb{F}_2^n}|\mathcal{F}(f+\varphi_a)|$

Combined generator



Correlation attacks

- **Divide-and-conquer** Attack.
- The state of the generator at time t can be divided in x_t and y_t of respective size ℓ and $(n - \ell)$.
- We attempt to recover the x_t individually.

Correlation attacks

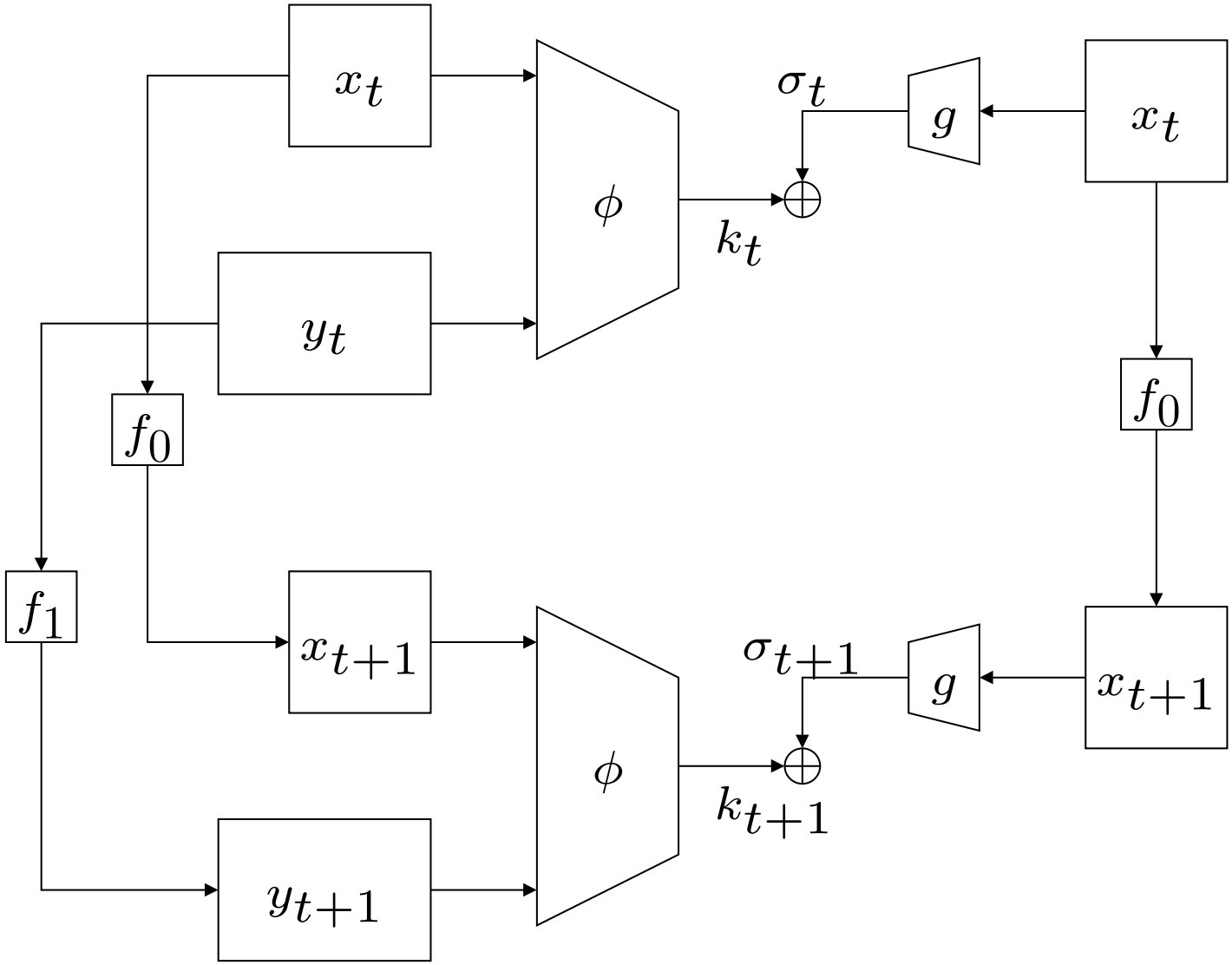
- L'attaque s'applique s'il existe une fonction g à ℓ variables qui coïncide avec la sortie de f dans plus de la moitié des cas :

$$p_g = Pr_{XY}[\varphi(X,Y) = g(X)] > 1/2$$

- De plus, on a :

$$Pr[k_t = \sigma_t] = p_g > 1/2$$

Correlation attacks



Correlation attacks

- For any $x_0 \in \{0,1\}^\ell$, we compute $\sigma_t = g \circ f_0(x_0)$
- Then, $c(k_t, \sigma_t) = \sum_{i=0}^{N-1} (-1)^{\sigma_i + k_i}$
- We need to find x_0 to maximize $c(k_t, \sigma_t)$.

Correlation attacks

Complexity

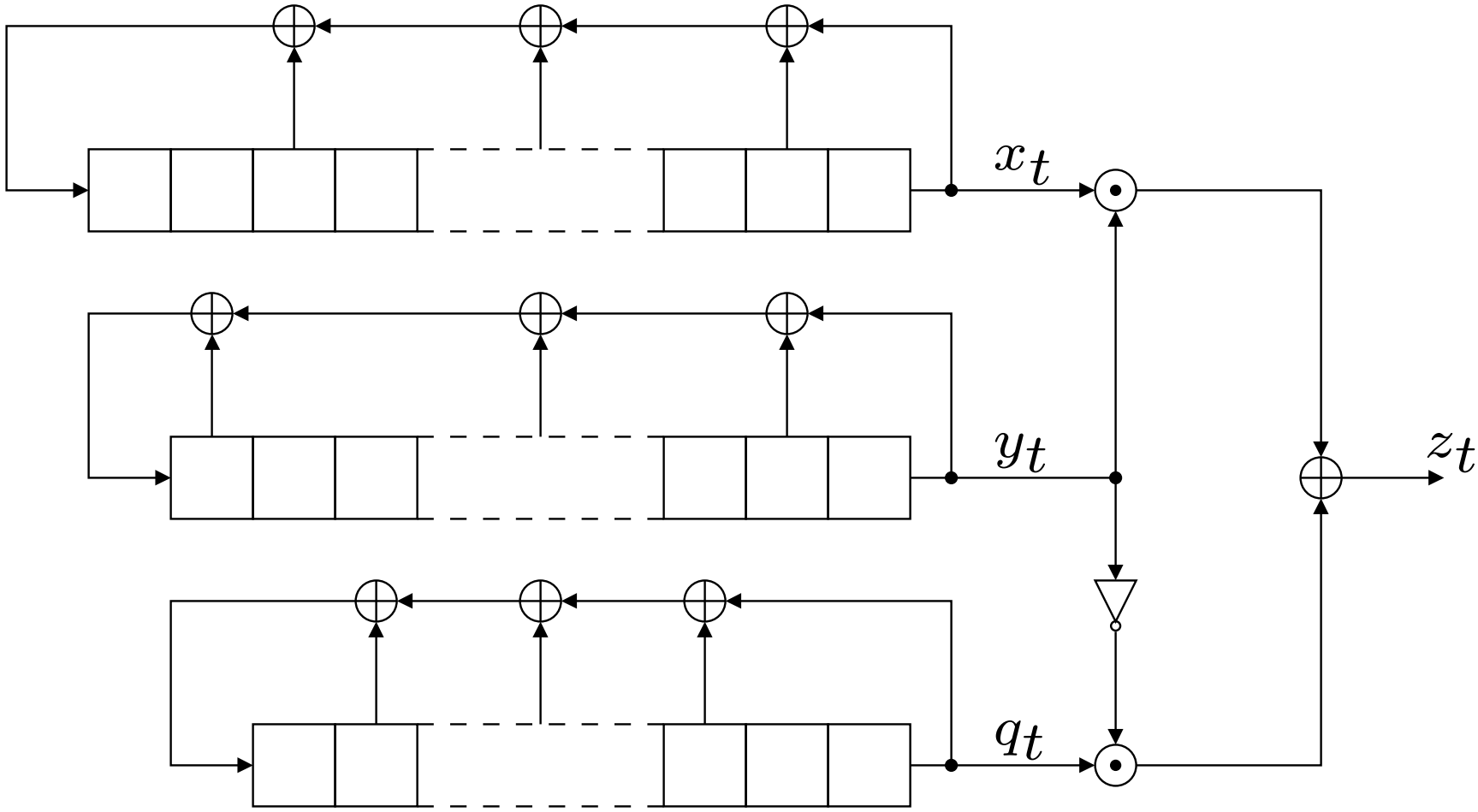
- The number of bits N of the keystream needed to recover x_0 is :

$$\left(\frac{1}{p_g - 0.5}\right)^2$$

- The complexity to recover the ℓ bits x_0 est $\ell 2^\ell$.

Combined generator

Geffe (example)



Correlations immunity

- f is without correlation of order t if the distribution of its output is unchanged after fixing any subset of t input variables and if the other $(n-t)$ input variables form a set of independent and uniformly distributed variables.

t-résilience

- f is t-resilient if f est without correlation of order t and balanced.

- A Boolean function f is t-resilient if and only if

$$\forall a \in \mathbb{F}_2^n,\ 0 \leq w_H(a) \leq t, \mathcal{F}(f + \varphi + a) = 0$$

Siegenthaler's bound

- Let f be without correlation of order t and algebraic degree d, then we have :

$$d + t \leq n.$$

If f is balanced and if $t < n - 1$ then

$$d + t \leq n - 1.$$

RC4 : Rivest Cipher 4

Generalities

System	Type	Creation
RC2	Bloc	1987
RC4	**Flot**	**1987**
RC5	Bloc	1994
RC6	Bloc	1997

- RC4 is the most popular stream ciphers : WEP, WPA, XBOX, Skype. . .
- **However, RC4 is very weak !**

RC4 : characteristics

- **Key :** lenght $40 \leq \ell \leq 128$ bits.
- **Internal state :**
  — RC4 work on a state S of 256 bytes.
  — 2 counters.
- **Initialization function** which depends on the key K.
- **Keystream generation functions :**
  — Update function f.
  — Filtering function ϕ.

- *To simplify the description of the algorithm, we reduce the state of RC4 to an array of 8 bytes (TinyRC4).*

TinyRC4

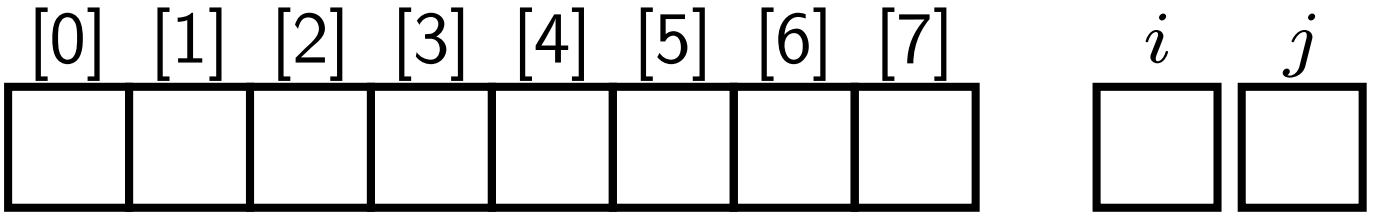
Initialization

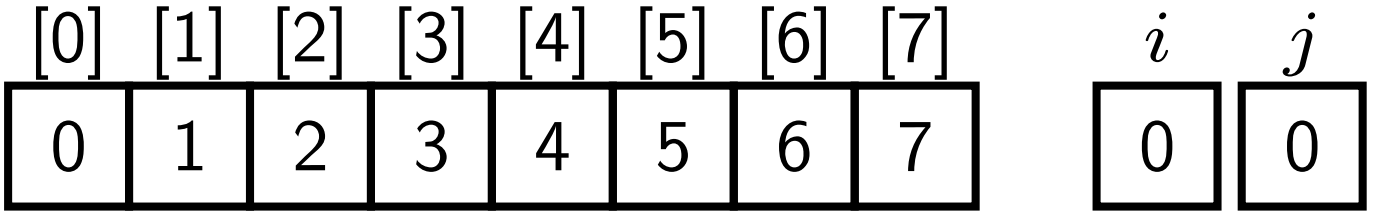
- The goal of the initialization is to obtain a pseudo-random permutation over $[0, 7]$ dependent from the key.

- Expected result (example) : $\{0, 1, 3, 7, 4, 6, 5, 2\}$

TinyRC4

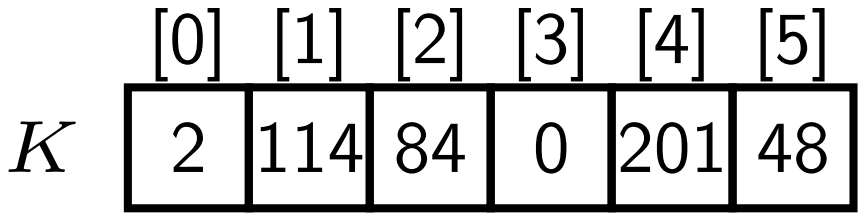
Initialization

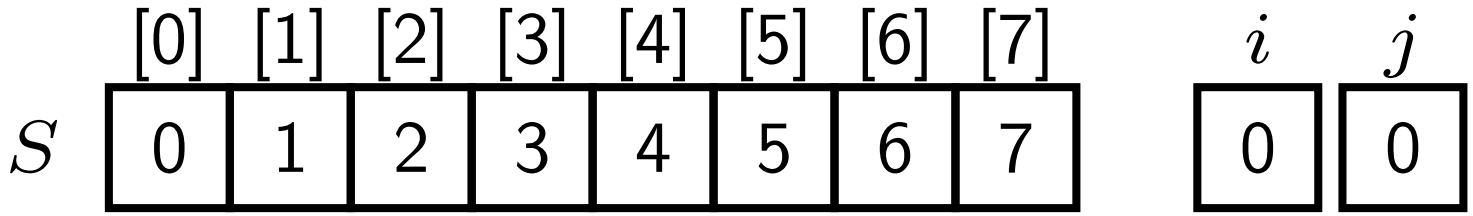




TinyRC4

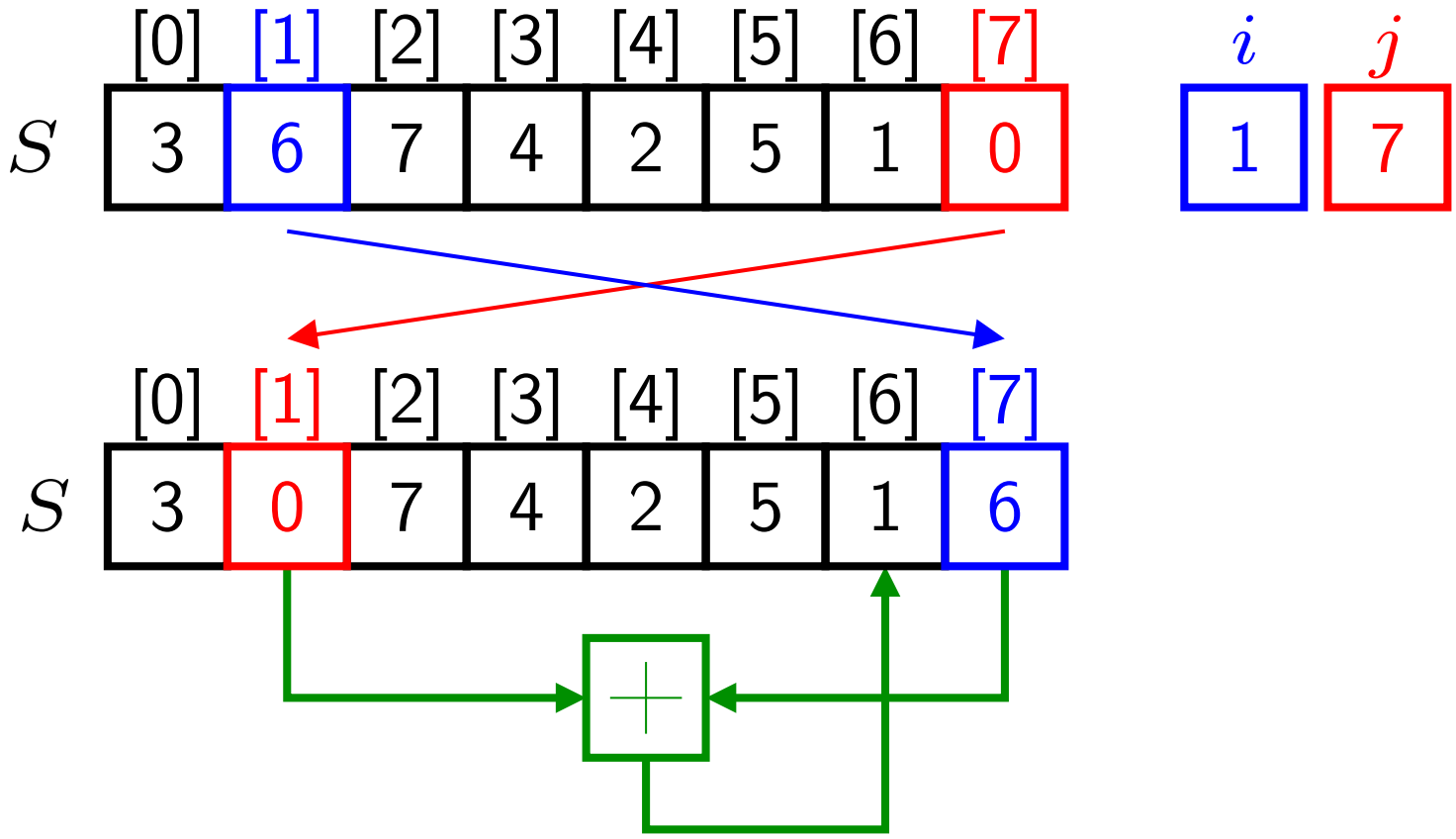
Initialization





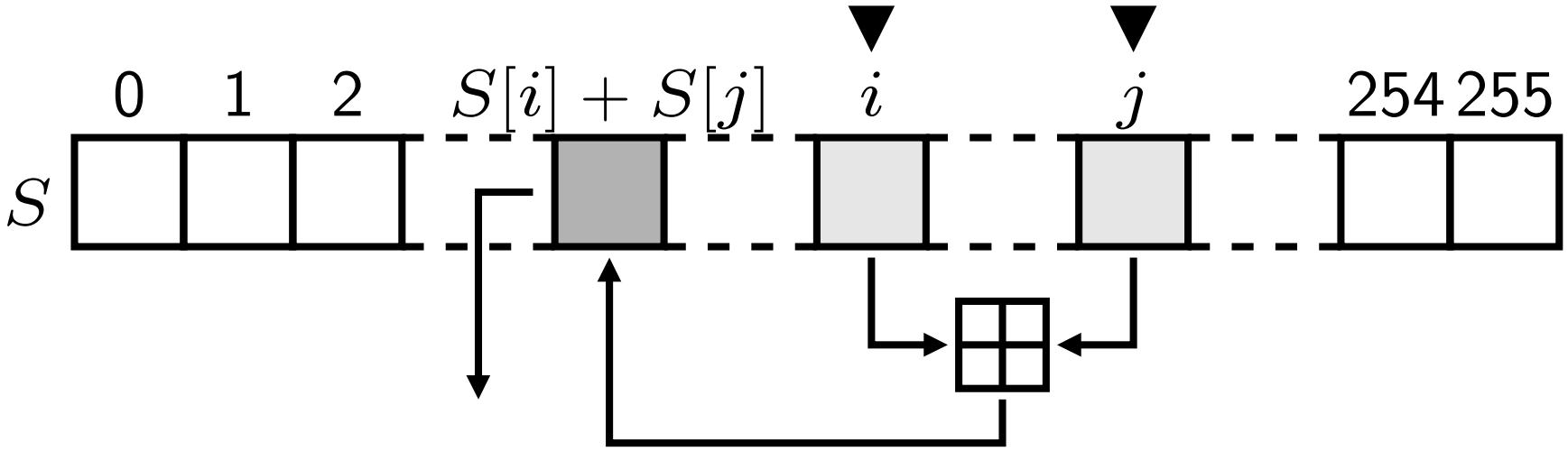
- Transition :
  - $i_{t+1} = i_t + 1$
  - $j_{t+1} = j_t + S[i_t] + K[i_t] \bmod 8$
  - $\texttt{swap}(S[i_t], S[j_{t+1}])$

Keystream generation



- Everything verify by TinyRC4 is also true for RC4.

Keystream generation



C code of RC4

- Very simple code!

```
int i=0,j=0,x,t;
for (x=0; x < len; ++x)
{
  i = (i + 1) % 256;
  j = (j + state[i]) % 256;
  t = state[i];
  state[i] = state[j];
  state[j] = t;
  out[x] = state[(state[i] + state[j]) % 256];
}
```

Code C de RC4

Initialisation

```
int i,j=0,t;
for (i=0; i < 256; ++i)
  state[i] = i;
for (i=0; i < 256; ++i)
{
  j = (j + state[i] + key[i % len]) % 256;
  t = state[i];
  state[i] = state[j];
  state[j] = t;
}
```

Rational design

RC4

- Very simple code!

```
int i=0,j=0,x,t;
for (x=0; x < len; ++x)
{
   i = (i + 1) % 256;
   j = (j + state[i]) % 256;
   out[x] = state[(state[i] + state[j]) % 256];
}
```

- We just remove the `swap` operation.

Rational design

swap

- Let i_t, j_t and z_t the respective values of i, j and of the keystream after t steps.
- Give a formula for i_{t+256}, j_{t+256}, j_{t+512}, z_{t+256} and z_{t+512}. Simplify ?

Period of simplified RC4

- Let i_t and j_t be the state of the counters after t Steps. We have for i_t and z_t :

$$\begin{aligned} i_{t+256} &= i_t \bmod 256 \\ z_{t+256} &= S[S[i_{t+256}] + S[j_{t+256}]] \\ &= S[S[i_t] + S[j_{t+256}]] \end{aligned}$$

- We need to simplify $S[j_{t+256}]$.

Period of simplified RC4

j_{t+256}

$$
\begin{aligned}
j_{t+256} &= j_t + \sum_{k=0}^{255} S[i_{t+k}] \bmod 256 \\
&= j_t + \sum_{k=0}^{255} S[k] \bmod 256
\end{aligned}
$$

S is a permutation, we can simplify the equation :

$$
\begin{aligned}
\sum_{k=0}^{255} S[k] &= \frac{255 \cdot 266}{2} \bmod 256 \\
&= 128 \bmod 256
\end{aligned}
$$

Period of simplified RC4

j_{t+256}

Thus, we obtain :

$$\begin{aligned} j_{t+256} &= j_t + 128 \bmod 256 \\ j_{t+512} &= j_t + 256 \bmod 256 \\ &= j_t \bmod 256 \end{aligned}$$

We have the relation :

$$\begin{aligned} z_{t+512} &= S[S[i_{t+512}] + S[j_{t+512}]] \\ &= S[S[i_t] + S[j_t]] \\ &= z_t \end{aligned}$$

Attacks on RC4

- To attack a stream cipher, we focus on three types of weaknesses :

  - **Keystream property**

  - **Inversion property**

  - **initialization Property**

Inversion of TinyRC4

and then on RC4

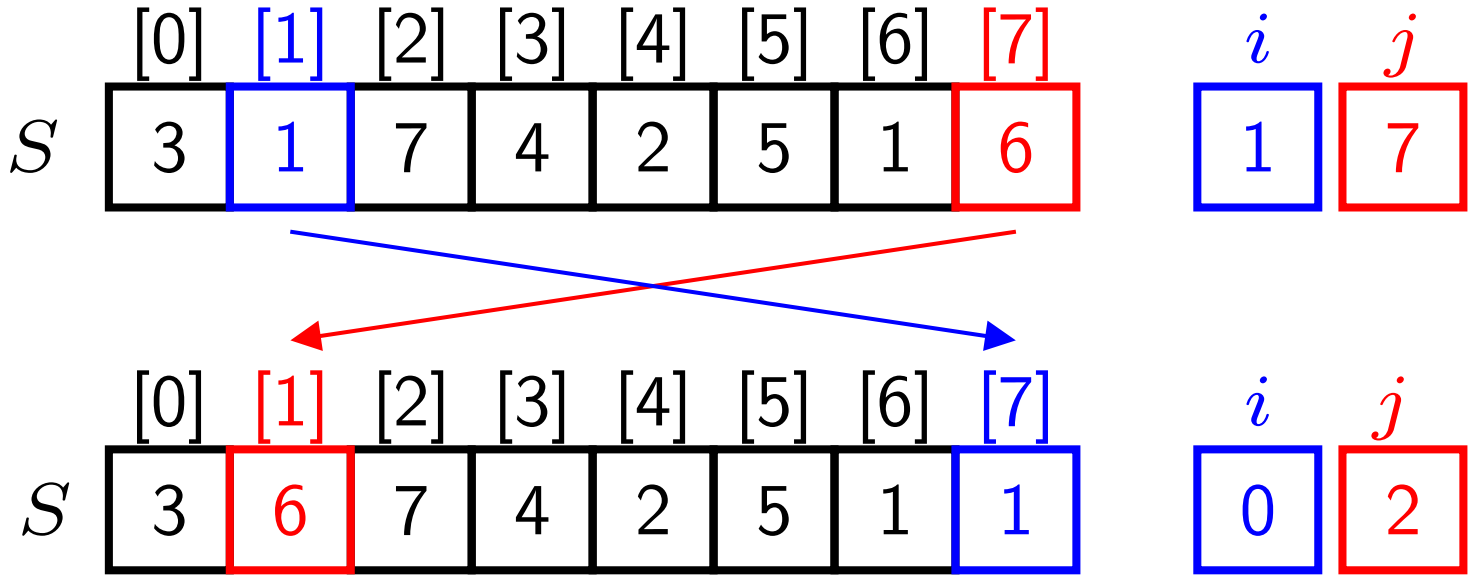
- We assume that we know the internal state after t rounds as well as the last byte of the keystream.
- We know S and $i_t = t$.
- We can easily recover j_t because :

$$k_t = S[S[i_t] + S[j_t]].$$

We know k_t, i_t et S, we can recover j.

Inversion of TinyRC4

Example



- Inversion :
  - — $\texttt{SWAP}(S[i],S[j])$
  - — $i_{t-1} = i_t - 1 \bmod 8$
  - — $j_{t-1} = j_t - S[i_{t-1}] \bmod 8$

Roos's Bias

Property for the initialization

- If the initialization of S was done by a perfect pseudo-random permutation then we would have :

$$\forall y \in \mathbb{Z}/256\mathbb{Z}, \mathbf{Pr}(S[y] = c) = 1/256$$

avec $\forall c \in \mathbb{Z}/256\mathbb{Z}$.

- Is RC4 initialization behaving like a perfect pseudo-random permutation ?

Roos's Bias

Property for the initialization

- We define f_y by

$$\begin{aligned} f_y &= \sum_{x=0}^{y} K[x] + x \\ &= \frac{y(y+1)}{2} + \sum_{x=0}^{y} K[x]. \end{aligned}$$

- The most likely value for $S[y]$ at the end of the initialization is $S[y] = f_y$. **[Roos 1995]**

Roos's Bias

Property for the initialization

y	$\mathbf{Pr}(S[y] = f_y)$							
0-7	0.370	0.368	0.362	0.358	0.349	0.340	0.330	0.322
8-15	0.309	0.298	0.285	0.275	0.260	0.245	0.229	0.216
16-23	0.203	0.189	0.173	0.161	0.147	0.135	0.124	0.112
24-31	0.101	0.090	0.082	0.074	0.064	0.057	0.051	0.044
32-39	0.039	0.035	0.030	0.026	0.023	0.020	0.017	0.014
40-47	0.013	0.012	0.010	0.009	0.008	0.007	0.006	0.006

We observe that $\frac{1}{256} = 0.00390625$

Roos's Bias

- We assume a 40-bit key :

$$K = \{106, 59, 220, 65, 34\}$$

y	0	1	2	3	4	5	6	7
f_y	106	166	132	200	238	93	158	129
$S_{256}[y]$	230	166	87	48	238	93	68	239
y	8	9	10	11	12	13	14	15
f_y	202	245	105	175	151	229	21	142
$S_{256}[y]$	202	83	105	147	151	229	35	142

Roos's Bias

- For $S[1]$ and $S[2]$ we have :

$$\mathbf{Pr}(S[1] = f_1) \approx \left(\frac{256-1}{256}\right)^{256}$$

$$\mathbf{Pr}(S[2] = f_2) \approx \left(\frac{256-1}{256}\right)^{256}$$

$$f_1 = K[0] + K[1] + 1$$

$$f_2 = K[0] + K[1] + K[2] + 3$$

- We obtain equations on the key from the seed.

Generalization

- We have :

$$\mathbf{Pr}(S[y] = f_y) = \left(\frac{256 - y}{256}\right) \cdot \left(\frac{256 - 1}{256}\right)^{256 + \frac{(y+1)y}{2}} + \frac{1}{256}$$

See Roos's paper.

Key recovery

- We have S et $K = \{106, 59, 220, 65, 34\}$

y	0	1	2	3	4	5	6	7
f_y	106	166	132	200	238	93	158	129
$S[y]$	230	166	87	48	238	93	68	239
y	8	9	10	11	12	13	14	15
f_y	202	245	105	175	151	229	21	142
$S[y]$	202	83	105	147	151	229	35	142

- Each time we have $S[y] = f_y$, we get a linear equation on the key bytes. Then, we can build a system of equations.

Key recovery

$$1 + \sum_{x=0}^{1} K[x] = 166 \quad (1)$$

$$10 + \sum_{x=0}^{4} K[x] = 238 \quad (2)$$

$$15 + \sum_{x=0}^{5} K[x] = 93 \quad (3)$$

$$36 + \sum_{x=0}^{8} K[x] = 202 \quad (4)$$

Attack FMS

Fluhrer-Mantin-Shamir

- Let $S_j[i]$ be the i^{th} byte of the permutation S after the j^{th} step.
- We assume that :
  — $S_0[2] = 0$,
  — $S_0[1] \neq 2$.

  Let us write $x = S_0[2]$ and $y = S_0[x]$. With

$$\begin{cases} i_1 = 1 \\ j_1 = 0 + S_0[i_1] = S_0[1] = x \end{cases}$$

We exchange $S_0[1]$ et $S_0[x]$.

Attack FMS

Bias in the keystream

- The keystream is given by :

$$\begin{aligned} z_1 &= S_1[S_1[1] + S_1[x] \bmod 256] \\ &= S_1[x + y] \end{aligned}$$

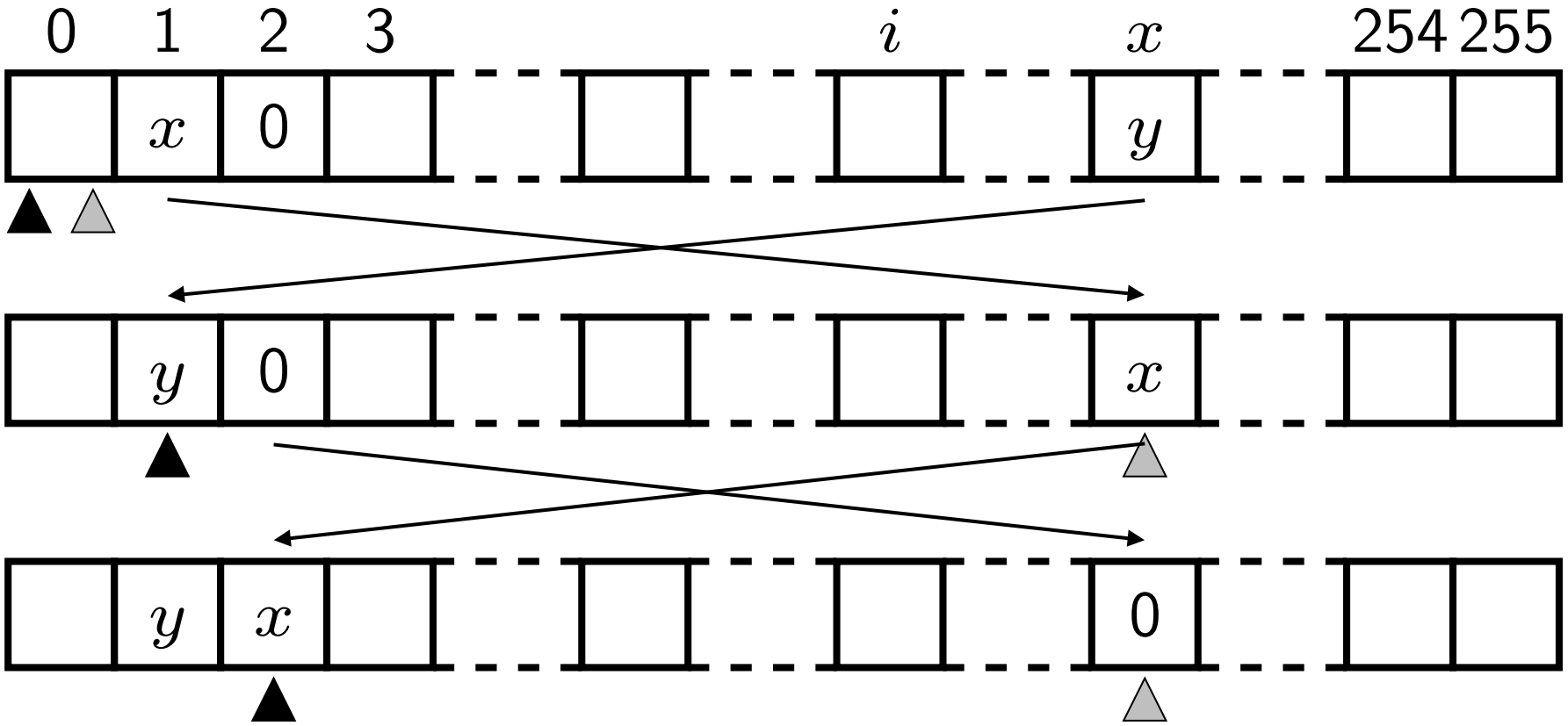
We have : $\begin{cases} i_2 = 2 \\ j_2 = x + S_1[i_2] = x + S_1[2] = x \end{cases}$

- The second byte of the keystream is given by :

$$z_2 = S_2[S_2[2] + S_2[x] \bmod 256] = 0$$

Attack FMS

Basic idea



Attack FMS

Bias

A random permutation has the following properties :

— $S_0[2] = 0$,

— $S_0[1] \neq 2$,

with probability $\frac{1}{256} \cdot \frac{255}{256}$.

For RC4, the probability to have $z_2 = 0$ is given by :

$$\begin{aligned} \mathbf{Pr}(z_2 = 0) &= 1 \cdots \frac{255}{256} + \frac{1}{256} \cdot \frac{256^2 - 255}{256^2} \\ &\approx \frac{1}{128} \end{aligned}$$

Sources

- **RC4 Stream Cipher and Its Variants.** *P. Goutam et S. Maitra, 2011 CRC Press.*
- **Weaknesses in the Key Scheduling Algorithm of RC4.** *S. R. Fluhrer, I. Mantin et A. Shamir, SAC 2001, Springer-Verlag.*
- **A Class of Weak Keys in the RC4 Stream Cipher.** *Andrew Roos, posté sur sci.crypt, 1995.*