Mathématiques-Cryptographie

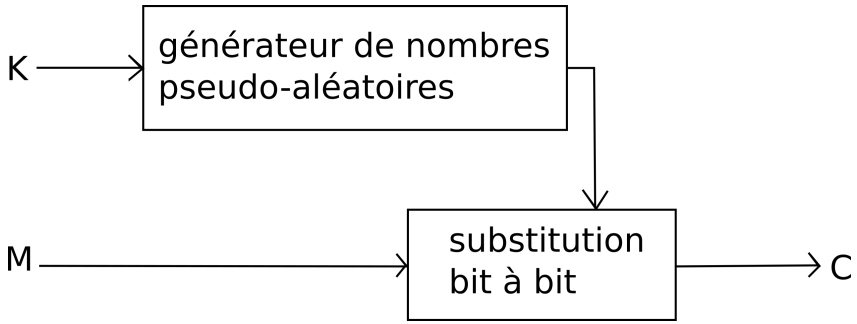
Anne Garcia-Sanchez

M2i cyber2 dev - CFA CCI Avignon

9 avril 2024

Chiffrements par flot

Chiffrement par flot (*stream cipher*), par flux, à la volée



chiffrement par flot synchrone:
texte chiffré = message clair (*xor*) suite secrète.

suite de même longueur que le message appelée **suite chiffrante**.

principe du chiffrement de Vernam

Chiffrements par flot

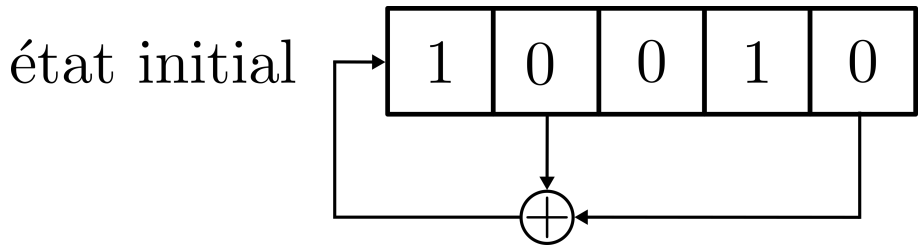
Exemples de chiffrements par flot:

- A5/1: basé sur des LFSR - plusieurs attaques publiées
- RC4 - plusieurs attaques publiées
- Chacha20

Registres à décalage à rétroaction linéaire - LFSR

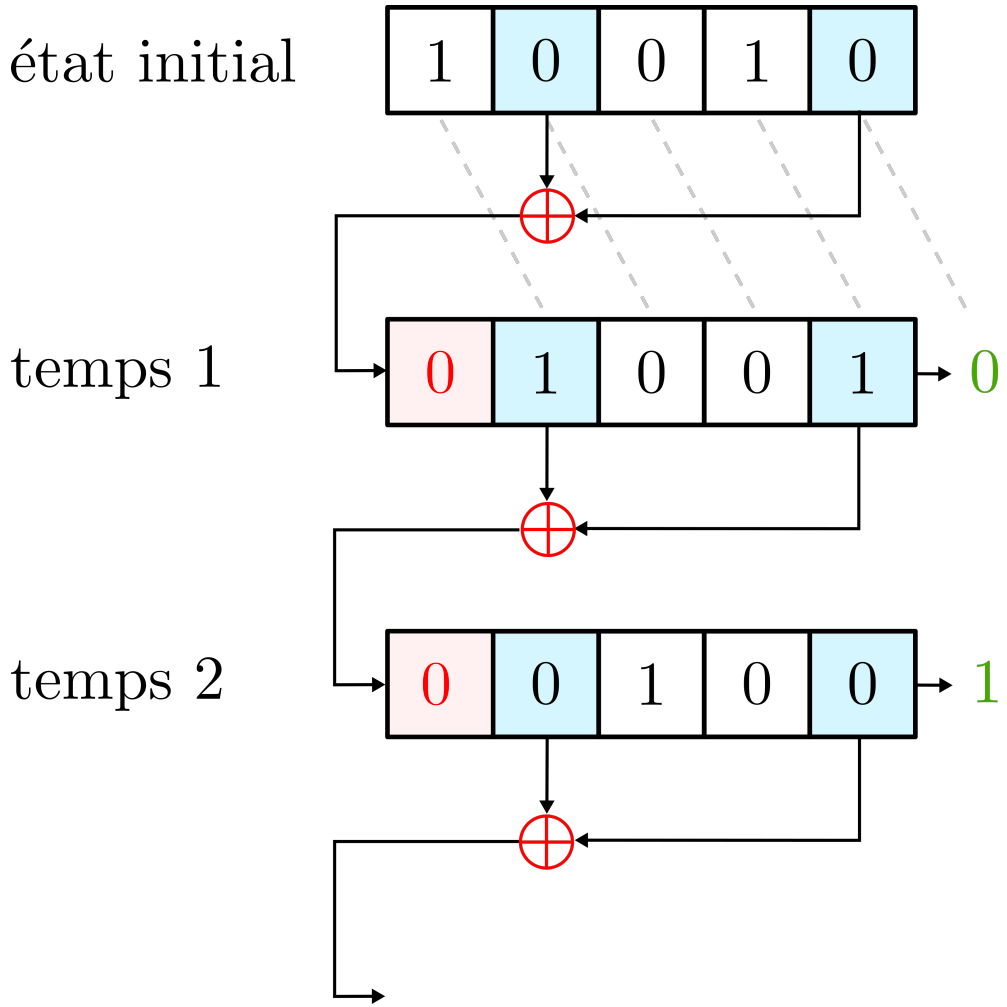
LFSR (*linear feedback shift register)* : engendre une suite vérifiant une relation de récurrence linéaire.

Un registre binaire = mémoire de cellules contenant des bits.

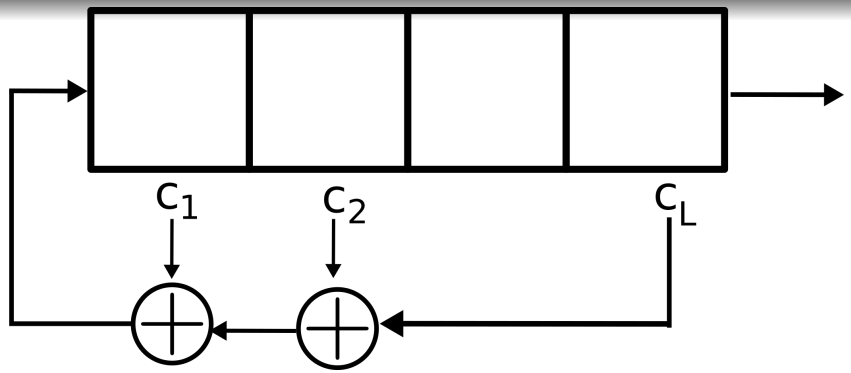


Registres à décalage à rétroaction linéaire - LFSR

à chaque unité de temps chaque bit est décalé d'une cellule vers la droite.



Registres à décalage à rétroaction linéaire - LFSR



La suite produite est périodique

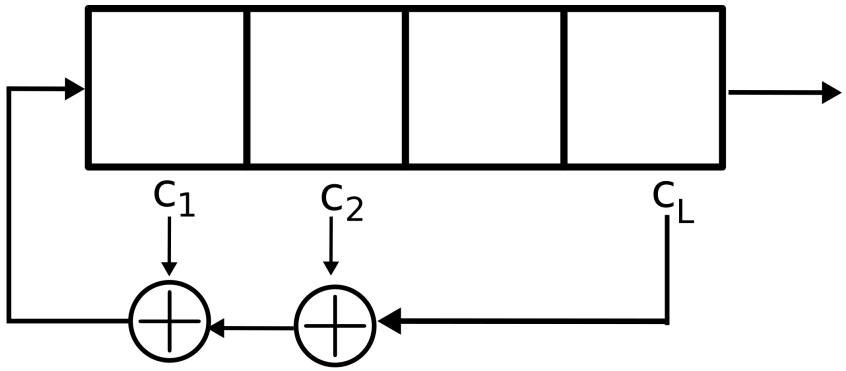
Elle est déterminée par:

état initial du registre + coefficients de rétroaction

coefficients de rétroaction: $c_1, c_2, ..c_L$

Si $c_i = 1$ alors la cellule est reliée
Si $c_i = 0$ alors la cellule n'est pas reliée

Registres à décalage à rétroaction linéaire - LFSR



coefficients de rétroaction: $c_1, ..c_L$ représentés par un polynôme:

$$P(X) = 1 + c_1X + c_2X^2 + ... + c_LX^L$$

polynôme de rétroaction du registre

Rappels d'algèbre: polynômes

Un polynôme à coefficients dans un corps $\mathbb{K}$ est une expression de la forme:

$$P(X) = a_0 + a_1 X + a_2 X^2 + ... + a_n X^n$$

avec n entier et $a_0, a_1, ..., a_n \in \mathbb{K}$

ici les coefficients a_i valent 0 ou 1

degré de P: n

Registres à décalage à rétroaction linéaire - LFSR

Période

registre à n bits: si le polynôme P(X) est bien choisi, le LFSR passe par $2^n - 1$ états avant de recommencer.

Exemple: registre à 128 bits, la période maximale est de $2^{128} - 1$

Très bonnes propriétés statistiques et très rapides

Registres à décalage à rétroaction linéaire - LFSR

 Faiblesse cryptographique

1969 algorithme de Berlekamp-Massey:

LFSR à k bits
$2k$ valeurs successives interceptées
polynôme de rétroaction inconnu

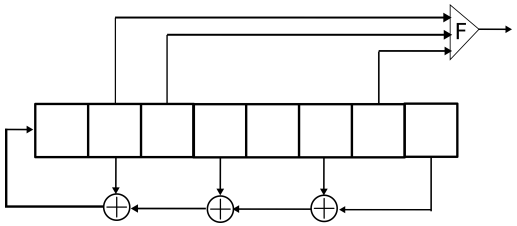
$\rightarrow$ l'algorithme permet de prévoir les bits suivants

Registres à décalage à rétroaction linéaire - LFSR

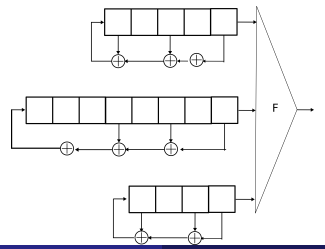
augmenter la complexité de la suite produite: plusieurs méthodes

- **registres à décalage irrégulier**: l'horloge est contrôlée par autre registre

- **registres à rétroaction linéaire filtrés**: certains bits du registre sont utilisés en entrée d'une fonction F qui produit la suite chiffrante



- **registres à rétroaction linéaire combinés**: les sorties de plusieurs registres sont combinés pour produire la suite chiffrante



Random - Python

`random` — Generate pseudo-random numbers¶

Source code: Lib/random.py

This module implements pseudo-random number generators for various distributions.

For integers, there is uniform selection from a range. For sequences, there is uniform selection of a random element, a function to generate a random permutation of a list in-place, and a function for random sampling without replacement.

On the real line, there are functions to compute uniform, normal (Gaussian), lognormal, negative exponential, gamma, and beta distributions. For generating distributions of angles, the von Mises distribution is available.

Almost all module functions depend on the basic function `random()`, which generates a random float uniformly in the semi-open range [0.0, 1.0). Python uses the Mersenne Twister as the core generator. It produces 53-bit precision floats and has a period of 2**19937-1. The underlying implementation in C is both fast and threadsafe. The Mersenne Twister is one of the most extensively tested random number generators in existence. However, being completely deterministic, it is not suitable for all purposes, and is completely unsuitable for cryptographic purposes.

The functions supplied by this module are actually bound methods of a hidden instance of the `random.Random` class. You can instantiate your own instances of `Random` to get generators that don't share state.

Class `Random` can also be subclassed if you want to use a different basic generator of your own devising: in that case, override the `random()`, `seed()`, `getstate()`, and `setstate()` methods. Optionally, a new generator can supply a `getrandbits()` method — this allows `randrange()` to produce selections over an arbitrarily large range.

The `random` module also provides the `SystemRandom` class which uses the system function `os.urandom()` to generate random numbers from sources provided by the operating system.

Warning: The pseudo-random generators of this module should not be used for security purposes. For security or cryptographic uses, see the `secrets` module.

See also: M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator", ACM Transactions on Modeling and Computer Simulation Vol. 8, No. 1, January pp.3–30 1998.

générateur pseudo-aléatoire `random` Python inadapté cryptographie