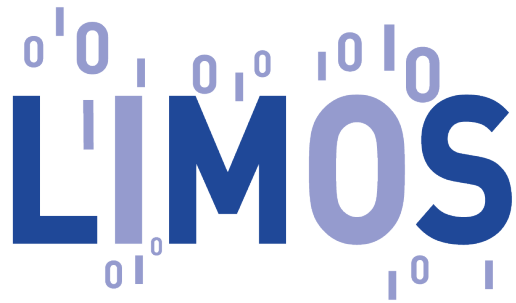
Codes

Pascal Lafourcade



November 21, 2019

LIMOS
LABORATOIRE D'INFORMATIQUE,
DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

Plan

Plan

- Mission Crypto
- Chiffrements historiques
- Code en ligne
- Marmottes
- Codage images
- Codage son Boomwhachers

Mission Crypto 1

Lettre 0 : Mission/Crypto

Lettre explicative : clair chiffré à observer.

Lettre 1 : Agent0111/Agent0111

Code de César avec un décalage de 3.

Lettre 2 : Agent0111/CESAR

H(password) identique $742,829,709$ pour des utilisateurs

Indications : auvergne, departement $=$ 63, flute enchantee, amadeus $=$ mozart

Lettre 3 : Alice/Mozart63

Point pour la lettre 4

Mission Crypto 1

Lettre 4 Blaise/Mozart63

Clef du code de Vigenere MATHS

Lettre 5 : Camille/38Picasso

Avec les 2 points des lettres précédentes: Résoudre le sytème : $63 = 38a + b$ et $-23 = -5a + b$

On trouve l'équation de la droite : $y = 2x - 13$

Lettre 6 : Agent0111/VIGENERE

Décalage de 13.

Lettre 7 : Agent0111/-189

Déduction logique des symboles du texte.

Mission Crypto 1

Lettre 8 : Agent0111/9430715286

Transposition : découpage du texte en bandelettes

Lettre 9 : Agent0111/1615

Partage de secret: $y = ax^2 + bx + s$.

$-190 = a \times 1 + b \times 1 + c$

$1450 = a \times 1652 + b \times 165 + c$

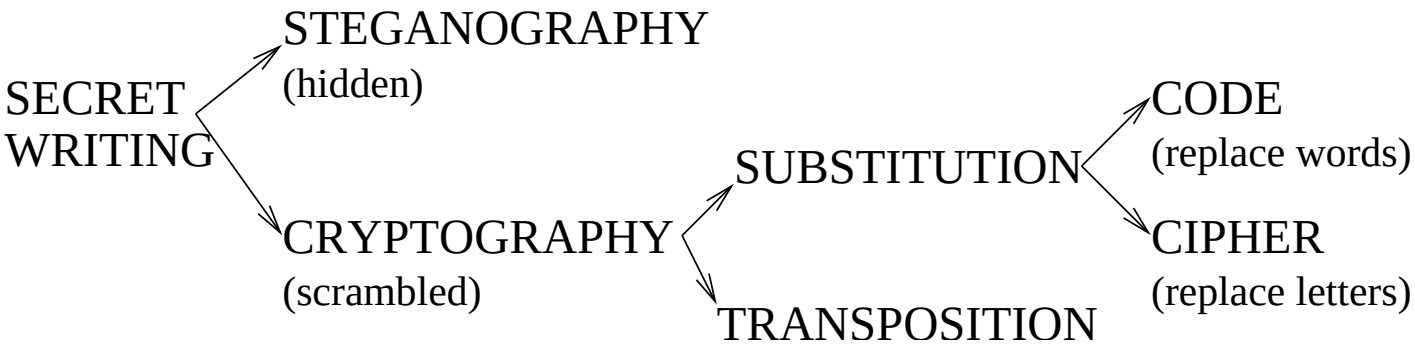
$9430 = a \times 1862 + b \times 186 + c$

Trouver a, b, s avec les 3 points découverts avant.

Lettre 10 : Agent0111/130

Découverte de l'Agent0111.

Information hiding



- Cryptology: the study of secret writing.
- Steganography: the science of hiding messages in other messages.
- Cryptography: the science of secret writing.

  Note: terms like encrypt, encode, and encipher are often (loosely and wrongly) used interchangeably

Slave

Mono-alphabetic substitution ciphers

- Simplest kind of cipher. Idea over 2,000 years old.
- Let $\mathcal{K}$ be the set of all permutations on the alphabet $\mathcal{A}$. Define for each $e \in \mathcal{K}$ an encryption transformation E_e on strings $m = m_1 m_2 \cdots m_n \in \mathcal{M}$ as

$$E_e(m) = e(m_1)e(m_2)\cdots e(m_n) = c_1 c_2 \cdots c_n = c\,.$$

- To decrypt c, compute the inverse permutation $d = e^{-1}$ and

$$D_d(c) = d(c_1)d(c_2)\cdots d(c_n) = m\,.$$

- E_e is a simple substitution cipher or a mono-alphabetic substitution cipher.

Substitution cipher examples

- KHOOR ZRUOG

Substitution cipher examples

- KHOOR ZRUOG = HELLO WORLD

  Caesar cipher: each plaintext character is replaced by the character three to the right modulo 26.

Substitution cipher examples

- KHOOR ZRUOG = HELLO WORLD

  Caesar cipher: each plaintext character is replaced by the character three to the right modulo 26.

- ZI anzr vf Nqnz

Substitution cipher examples

- KHOOR ZRUOG = HELLO WORLD

  Caesar cipher: each plaintext character is replaced by the character three to the right modulo 26.

- ZI anzr vf Nqnz = My name is Adam

  ROT13: shift each letter by 13 places.
  Under Unix: tr a-zA-Z n-za-mN-ZA-M.

- 2-25-5 2-25-5

Substitution cipher examples

- KHOOR ZRUOG = HELLO WORLD

  Caesar cipher: each plaintext character is replaced by the character three to the right modulo 26.

- ZI anzr vf Nqnz = My name is Adam

  ROT13: shift each letter by 13 places.
  Under Unix: tr a-zA-Z n-za-mN-ZA-M.

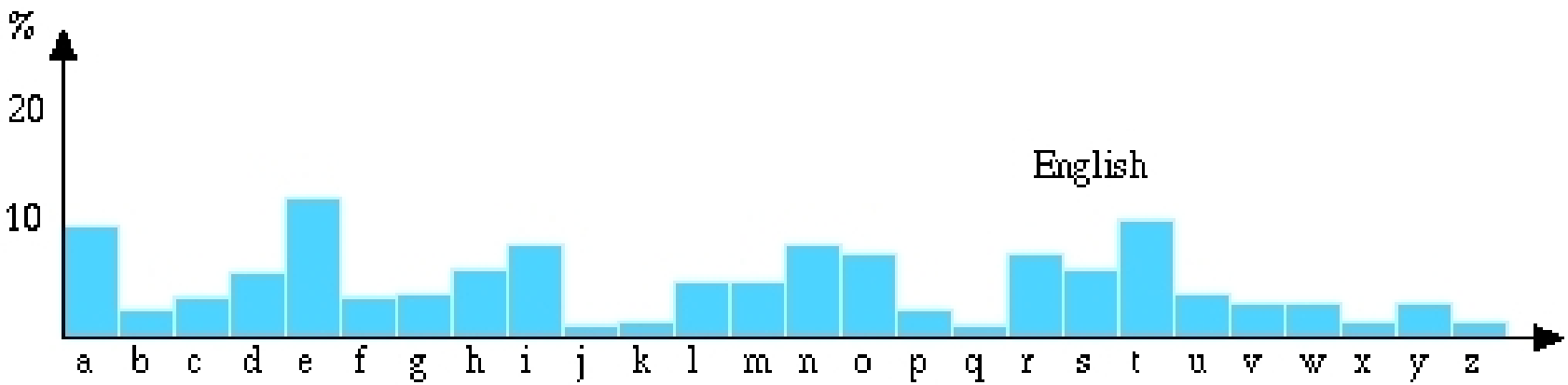
- 2-25-5 2-25-5 = BYE BYE

  Alphanumeric: substitute numbers for letters.

How hard are these to cryptanalyze? Caesar? General?

(In)security of substitution ciphers

- Key spaces are typically huge. 26 letters $\rightsquigarrow$ 26! possible keys.
- Trivial to crack using frequency analysis (letters, digraphs...)
- Frequencies for English based on data-mining books/articles.



How to break a monoalphabetic cipher

- Guess the target language
- Count letter frequencies in the cryptogram C
- Match cryptogram's frequencies with language's frequencies
- Use the partially decrypted message to correct errors.

Homophonic substitution ciphers

- To each $a \in \mathcal{A}$, associate a set $H(a)$ of strings of t symbols, where $H(a), a \in \mathcal{A}$ are pairwise disjoint. A homophonic substitution cipher replaces each a with a randomly chosen string from $H(a)$. To decrypt a string c of t symbols, one must determine an $a \in A$ such that $c \in H(a)$. The key for the cipher is the sets $H(a)$.

Homophonic substitution ciphers

- To each $a \in \mathcal{A}$, associate a set $H(a)$ of strings of t symbols, where $H(a), a \in \mathcal{A}$ are pairwise disjoint. A homophonic substitution cipher replaces each a with a randomly chosen string from $H(a)$. To decrypt a string c of t symbols, one must determine an $a \in A$ such that $c \in H(a)$. The key for the cipher is the sets $H(a)$.

Example:

$\mathcal{A} = \{a, b\}$, $H(a) = \{00, 10\}$, and $H(b) = \{01, 11\}$. The plaintext ab encrypts to one of 0001, 0011, 1001, 1011.

Rational: makes frequency analysis more difficult.

Cost: data expansion and more work for decryption.

Polyalphabetic substitution ciphers

- Idea (Leon Alberti): conceal distribution using family of mappings.

- A polyalphabetic substitution cipher is a block cipher with block length t over alphabet $\mathcal{A}$ where:
  - the key space $\mathcal{K}$ consists of all ordered sets of t permutations over $\mathcal{A}$, $(p_1, p_2, \ldots, p_t)$.
  - Encryption of $m = m_1 \cdots m_t$ under key $e = (p_1, \cdots, p_t)$ is $E_e(m) = p_1(m_1) \cdots p_t(m_t)$.
  - Decryption key for e is $d = (p_1^{-1}, \cdots p_t^{-1})$.

Example: Vigenère ciphers

- Key given by sequence of numbers $e = e_1, \ldots, e_t$, where

$$p_i(a) = (a + e_i) \bmod n$$

  defining a permutation on an alphabet of size n.

- Example: English ($n = 26$), with k = 3,7,10

  m = THI SCI PHE RIS CER TAI NLY NOT SEC URE

then

$E_e(m)$ = WOS VJS SOO UPC FLB WHS QSI QVD VLM XYO

One-time pads (Vernam cipher)

- A one-time pad is a cipher defined over $\{0, 1\}$. Message $m_1 \cdots m_n$ is encrypted by a binary key string $k_1 \cdots k_n$.

$$\begin{aligned} E_{k_1 \cdots k_n}(m_1 \cdots m_n) &= (m_1 \oplus k_1) \cdots (m_n \oplus k_n) \\ D_{k_1 \cdots k_n}(c_1 \cdots c_n) &= (c_1 \oplus k_1) \cdots (c_n \oplus k_n) \end{aligned}$$

- Example:

$$\begin{array}{rcl} m & = & 010111 \\ k & = & 110010 \\ \hline c & = & 100101 \end{array}$$

- Since every key sequence is equally likely, so is every plaintext!

  Unconditional (information theoretic) security, if key isn't reused!

- Moscow–Washington communication previously secured this way.
- Problem?

One-time pads (Vernam cipher)

- A one-time pad is a cipher defined over $\{0, 1\}$. Message $m_1 \cdots m_n$ is encrypted by a binary key string $k_1 \cdots k_n$.

$$\begin{aligned} E_{k_1 \cdots k_n}(m_1 \cdots m_n) &= (m_1 \oplus k_1) \cdots (m_n \oplus k_n) \\ D_{k_1 \cdots k_n}(c_1 \cdots c_n) &= (c_1 \oplus k_1) \cdots (c_n \oplus k_n) \end{aligned}$$

- Example:

$$\begin{array}{rcl} m & = & 010111 \\ k & = & 110010 \\ \hline c & = & 100101 \end{array}$$

- Since every key sequence is equally likely, so is every plaintext!

  Unconditional (information theoretic) security, if key isn't reused!

- Moscow–Washington communication previously secured this way.

- Problem? Securely exchanging and synchronizing long keys.

Transposition ciphers

- For block length t, let $\mathcal{K}$ be the set of permutations on $\{1, \ldots, t\}$. For each $e \in \mathcal{K}$ and $m \in \mathcal{M}$

$$E_e(m) = m_{e(1)} m_{e(2)} \cdots m_{e(t)} .$$

- The set of all such transformations is called a transposition cipher.
- To decrypt $c = c_1 c_2 \cdots c_t$ compute $D_d(c) = c_{d(1)} c_{d(2)} \cdots c_{d(t)}$, where d is inverse permutation.
- Letters unchanged so frequency analysis can be used to reveal if ciphertext is a transposition. Decrypt by exploiting frequency analysis for diphthongs, tripthongs, words, etc.

Example: transposition ciphers

- $C =$ Aduaenttlydhatoiekounletmtoihahvsekeeeleeyqonouv

Example: transposition ciphers

- $C =$ Aduaenttlydhatoiekounletmtoihahvsekeeeleeyqonouv

A	n	d	i	n	t	h	e	e	n
d	t	h	e	l	o	v	e	y	o
u	t	a	k	e	i	s	e	q	u
a	l	t	o	t	h	e	l	o	v
e	y	o	u	m	a	k	e		

Table defines a permutation on 1, ..., 50.

Example: transposition ciphers

- C = Aduaenttlydhatoiekounletmtoihahvsekeeeleeyqonouv

A	n	d	i	n	t	h	e	e	n
d	t	h	e	l	o	v	e	y	o
u	t	a	k	e	i	s	e	q	u
a	l	t	o	t	h	e	l	o	v
e	y	o	u	m	a	k	e		

Table defines a permutation on 1, ..., 50.

- Idea goes back to Greek Scytale: wrap belt spirally around baton and write plaintext lengthwise on it.

Composite ciphers

- Ciphers based on just substitutions or transpositions are not secure
- Ciphers can be combined. However . . .
  - two substitutions are really only one more complex substitution,
  - two transpositions are really only one transposition,
  - but a substitution followed by a transposition makes a new harder cipher.
- Product ciphers chain substitution-transposition combinations.
- Difficult to do by hand
  $\rightsquigarrow$ invention of cipher machines.

ENIGMA

Three-rotor German military Enigma machine

Dayly keys are used and stored in a book.

There are 10^{114} possibilities for one cipher.

Other German Tricks

A space was omitted or replaced by an X. The X was generally used as point or full stop. They replaced the comma by Y and the question sign by UD. The combination CH, as in "Acht" (eight) or "Richtung" (direction) were replaced by Q (AQT, RIQTUNG).

Kerchoff's Principle

In 1883, a Dutch linguist Auguste Kerchoff von Nieuwenhof stated in his book "La Cryptographie Militaire" that:

"the security of a crypto-system must be totally dependent on the secrecy of the key, not the secrecy of the algorithm."

Author's name sometimes spelled Kerckhoff

Shannon's Principle 1949

Confusion

The purpose of confusion is to make the relation between the key and the ciphertext as complex as possible.

Ciphers that do not offer much confusion (such as Vigenere cipher) are susceptible to frequency analysis.

Diffusion

Diffusion spreads the influence of a single plaintext bit over many ciphertext bits.

The best diffusing component is substitution (homophonic)

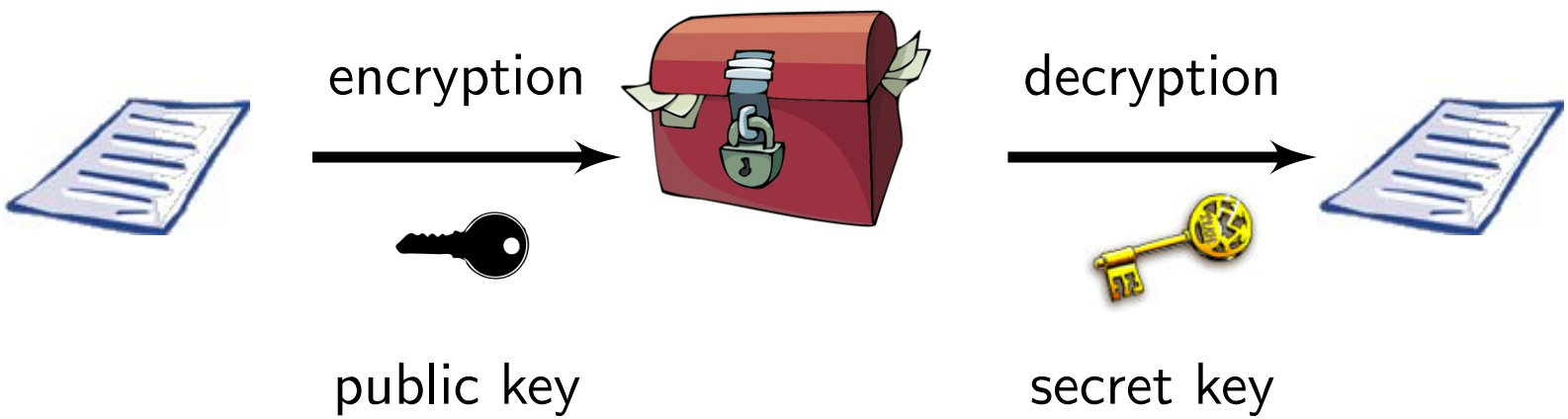
Principle

A good cipher design uses Confusion and Diffusion together

Symmetric vs Asymmetric Encryption

Symmetric Encryption (DES, AES)

Asymmetric Encryption (RSA, Elgamal ...)



Comparison

- Size of the key
- Complexity of computation (time, hardware, cost ...)
- Number of different keys ?
- Key distribution
- Signature only possible with asymmetric scheme

Computational cost of encryption

2 hours of video (assumes 3Ghz CPU)

	DVD 4,7 G.B		Blu-Ray 25 GB	
Schemes	encrypt	decrypt	encrypt	decrypt
RSA 2048(1)	22 min	24 h	115 min	130 h
RSA 1024(1)	21 min	10 h	111 min	53 h
AES CTR(2)	20 sec	20 sec	105 sec	105 sec

Travail à la maison

Tour de magie

Bit de parité.

Activité : Menteur

On a 4 cartes avec deux visages tristes ou contents

Le robot choisit la carte qu'il veut. L'humain doit deviner sa carte. Mais le robot ne sait répondre que 1 et 0...

Menteur

Mais le robot PEUT tricher une fois !

Attention : une SEULE fois

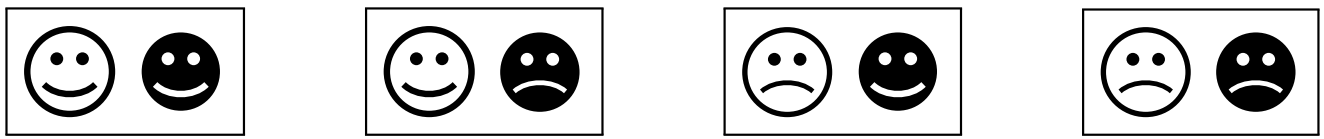
En combien de fois l'humain est-il sûr de deviner la carte du robot ?

Activité : Menteur

Fiche Robot

Entoure le numéro de ta carte !

1 2 3 4



Réponses :

1	2	3	4	5	6	7
0 ou 1	0 ou 1	0 ou 1	0 ou 1	0 ou 1	0 ou 1	0 ou 1

Mensonge :

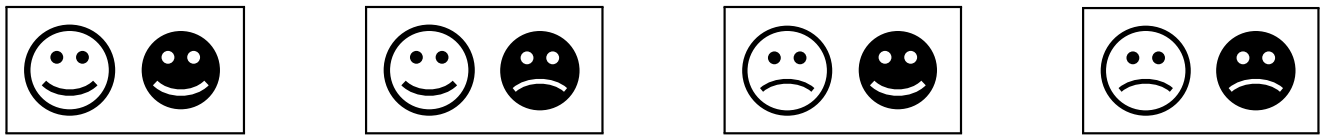
Activité : Menteur

Fiche Humain

Questions	Réponses
1. Dis 1 si :	0 ou 1
2. Dis 1 si :	0 ou 1
3. Dis 1 si :	0 ou 1
4. Dis 1 si :	0 ou 1
5. Dis 1 si :	0 ou 1
6. Dis 1 si :	0 ou 1
7. Dis 1 si :	0 ou 1

Quelle carte le robot a-t-il ?

1 2 3 4



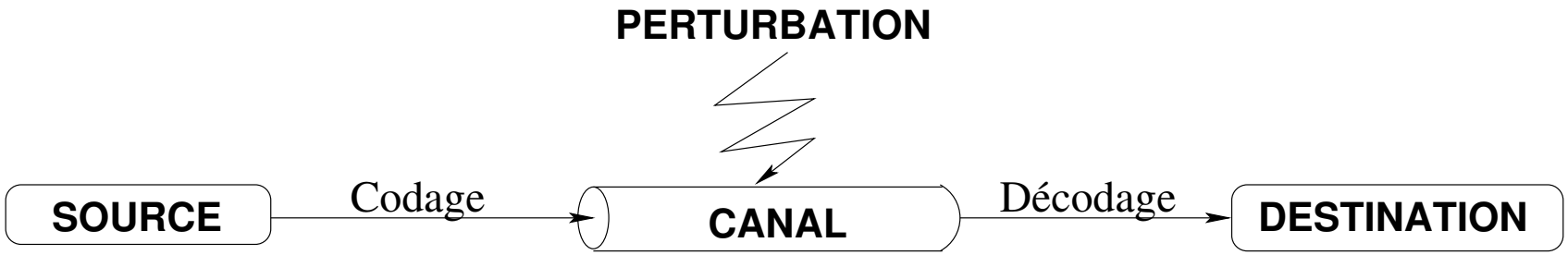
Activité Image

Numérisation

FAX/Scanner

Représentation de l'information

Claude Elwood Shannon



Efficacité de la transmission : compression des données
Sécurité de l'information : cryptage, authentification
Intégrité du message : correction d'erreurs

Critères de qualité d'un code :

1. Intégrité de l'information : tolérance aux fautes (détection/correction des erreurs)
2. Sécurité de l'information : authentification (Chiffrement)
3. Efficacité de la transmission : compression des données.

Code du Fax

Code du FAX

Require: M, une suite de 1728 pixels (de $M[0]$ à $M[1727]$)
Ensure: C, le message M compressé
  $n \leftarrow 1$ // *compteur des pixels consécutifs de même valeur*
  $C \leftarrow$ "" // *chaîne vide*
  for i de 1 à 1727 **do**
    if $M[i-1] = M[i]$ **then**
      $n \leftarrow n+1$ // *incrémente le nombre de pixels consécutifs de valeur* $M[i]$
    else
      Concaténer n et la couleur du dernier pixel $M[i-1]$ à C
      $n \leftarrow 1$ // *ré-initialisation du compteur*
    end if
  end for
  Concaténer n et la couleur du dernier pixel $M[1727]$ à C
  Renvoyer C

Fax

Questions

- Par 2 coder une image de taille 10×10, la transmettre et la décoder.
- Que pensez-vous du code fax présenté plus haut?
- Trouver une image qui demande plus de place avec ce codage.
- Proposer une amélioration

Fax

Solution

Le résultat peut même être plus long que si on avait codé simplement les pixels noirs par des 1 et les pixels blancs par des 0, par exemple lorsque l'image consiste entièrement en une suite de "01". Ce n'est sûrement pas très courant pour les fax, mais on apprend ainsi que ce code n'a aucune garantie théorique!

Pour l'améliorer, on peut par exemple coder un nombre k de "01" consécutifs par "k01", et ainsi étendre les possibilités du code.

Activité Musique

Numérisation

Micro encodage partition

Code Ambigü : Bijectivité n'est pas suffisant

Pour un message source $a_1 ... a_n$, chaîne sur un alphabet source quelconque, et pour un alphabet V, appelons f la fonction de codage. On a alors le message codé $c_1 ... c_n = f(a_1) ... f(a_n)$, avec $c_i \in V^+$ pour tout i.

Codage des lettres de l'alphabet $S = \{A, \ldots, Z\}$ par les entiers $C = \{0, \ldots, 25\}$ en base 10

$$f(A) = 0, f(B) = 1, \ldots, f(J) = 9, \ldots, f(Z) = 25\,.$$

Le mot de code 1209 peut alors correspondre à différents messages : BUJ ou MAJ ou BCAJ.

Un code C[1] sur un alphabet V est dit *non ambigu* si, pour tout $x = x_1 \ldots x_n \in V^+$, il existe au plus une séquence $c = c_1 \ldots c_m \in C^+$ telle que $c_1 \ldots c_m = x_1 \ldots x_n$.

LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

[1] C : ensemble des mots de code.

Code

Propriété

Un code C sur un alphabet V est non ambigu si et seulement si pour toutes séquences $c = c_1 \dots c_n$ et $d = d_1 \dots d_m$ de C^+ :

$$c = d \Rightarrow (n = m \text{ et } c_i = d_i \text{ pour tout } i = 1, \dots, n)\,.$$

Example

Sur l'alphabet $V = \{0, 1\}$,

- le code $C = \{0, 01, 001\}$ n'est pas uniquement déchiffrable.
- le code $C = \{01, 10\}$ est uniquement déchiffrable.
- le code $C = \{0, 10, 110\}$ est uniquement déchiffrable.

Code Préfixe

Propriété

Un code C sur un alphabet V a la *propriété du préfixe* (il est *instantané* ou *irréductible*) si et seulement si pour tout couple de mots de code distincts (c_1, c_2), c_2 n'est pas un préfixe de c_1.

Autrement dit, aucun mot fini d'un code préfixe ne peut se prolonger pour donner un autre mot.

Example

$a = 101000$, $b = 01$, $c = 1010$: b n'est pas un préfixe de a mais c est un préfixe de a.

Préfixe

Propriété

Tout code possédant la propriété du préfixe est uniquement déchiffrable.

Preuve : Soit un code C sur V qui n'est pas uniquement déchiffrable. Alors il existe une chaîne $a \in V^n$ telle que $a = c_1 \dots c_l = d_1 \dots d_k$, les c_i et d_i étant des **mots de** C et $c_i \neq d_i$ pour au moins un i. Choisissons le plus petit i tel que $c_i \neq d_i$ (pour tout $j < i$, $c_j = d_j$). Alors $l(c_i) \neq l(d_i)$, sinon, vu le choix de i, on aurait $c_i = d_i$, ce qui est en contradiction avec la définition de i. Si $l(c_i) < l(d_i)$, c_i est un préfixe de d_i et dans le cas contraire d_i est un préfixe de c_i. C n'a donc pas la propriété du préfixe.

Propriété

Tout code dont tous les mots sont de même longueur possède la propriété du préfixe.

Exemples

- Indicatif international
- Code UTF8
- Code Binaire
- Code de Finbonacci (Théorème de Zeckendorf)
- Code de Shannon-Fano (compression sans perte)
- Code de Huffman

Tour de magie

Choisir un nombre entre 0 et 31

1	3	5	7
9	11	13	15
17	19	21	23
25	27	29	31

2	3	6	7
10	11	14	15
18	19	22	23
26	27	30	31

4	5	6	7
12	13	14	15
20	21	22	23
28	29	30	31

8	9	10	11
12	13	14	15
24	25	26	27
28	29	30	31

16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31

Tour de magie

Choisir un nombre entre 0 et 32

1	4	6	9
12	14	17	19
22	25	27	30

2	7	10	15
20	23	28	31

3	4	11	12
16	17	24	25
32			

5	6	7	18
19	20	26	27
28			

8	9	10	11
12	29	30	31
32			

13	14	15	16
17	18	19	20

21	22	23	24
25	26	27	28
29	30	31	32

Théorème de Zeckendorf

Suite de Fibonacci

$$F_1 = F_2 = 1$$

$$F_{n+2} = F_{n+1} + F_n$$

Théorème

Pour tout entier naturel N, il existe une unique suite d'entiers $c_0, \ldots, c_k$, avec $c_0 \geq 2$ et $c_{i+1} > c_i + 1$, tels que

$$N = \sum_{i=0}^{k} F_{c_i}$$

où F_n est le n-ième nombre de Fibonacci.

Preuve de Zeckendorf : Existence

Cas de base : Si $N \leq 5$, on a $F_1 = 1 = F_2$, $F_3 = 2$ $F_4 = 3$ et $F_5 = 5$. Vérifie que $F_1 + F_3 = F_4$ et $F_1 + F_2 + F_4 = F_5$.

Induction : Soit $F_k \leq N < F_{k+1}$, avec $k \geq 5$, le plus grand nombre de Fibonacci inférieur ou égal à N et soit $M = N - F_k$.

Donc M est plus petit que N donc par hypothèse de récurrence il possède une représentation.

Alors, pour tout nombre F_ℓ de cette représentation,

$$F_k + F_\ell \leq F_k + M = N$$

Or $N < F_{k+1} = F_k + F_{k-1}$ donc $F_\ell < F_{k-1}$.

Ainsi la représentation de M, augmentée de F_k, est une représentation de N.

Preuve de Zeckendorf : Lemme

Lemme

La somme de tout ensemble non vide de nombres de Fibonacci distincts et non consécutifs, dont le plus grand élément est F_j, est strictement inférieure à F_{j+1}.

Preuve par récurrence simple sur le nombre d'éléments d'un tel ensemble S.

Initialisation : triviale avec 1, 1, 2, 3, 5.

Récurrence : Si S a plus d'un élément, enlevons F_j. Par hypothèse de récurrence, la somme des éléments restants est strictement inférieure à F_{j-1} donc la somme totale est strictement inférieure à $F_{j-1} + F_j = F_{j+1}$.

Preuve de Zeckendorf : Unicité

Par contradiction, supposons qu'il y a deux ensembles différents S_n et T_n tels que $\sum_{F_i \in S_n} F_i = n = \sum_{F_i \in T_n} F_i$. On soustrait ces deux ensembles pour obtenir deux ensembles disjoints $S_n^* = S_n \setminus T_n$ et $T_n^* = T_n \setminus S_n$. Par construction nous avons $\sum_{F_i \in S_n^*} F_i = n - \sum_{F_i \in T_n \cap S_n} F_i = \sum_{F_i \in T_n^*} F_i$. Soit F_j le maximum de S_n^* et F_k le maximum de T_n^*, sans perte de généralité supposons $F_j < F_k$, ainsi de par la construction des nombres de Fibonnacci $F_{j+1} \leq F_k$. Par le lemme précédent

$$\sum_{F_i \in S_n^*} F_i < F_{j+1} \leq F_k$$

Comme F_k est le maximum de T_n^* nous avons $\sum_{F_i \in T_n^*} F_i \geq F_k$. Ceci contredit l'égalité de la somme de ces deux ensembles.

Activité : Transmission de pensée

Tour de magie cartes[2].

1. La foule choisit 5 cartes d'un jeu de 52.
2. L'asssitant en cache une et ordonne les 4 restantes.
3. Le magicien retrouve la carte cachée.

[2] https://www.math.univ-paris-diderot.fr/diffusion/_media/fiches/transmission-pensee.pdf

Activité : Transmission de pensée

Ordre des cartes :

Pique > Coeur > Carreau > Tréfle, couleur puis numéro

Numérotation:

- P M G : 1
- P G M : 2
- M P G : 3
- M G P : 4
- G P M : 5
- G M P : 6

Exemple :

- Caché : 3 Coeur
- Visible : Dame Coeur, Roi Carreau, 4 Pique, 3 Trègle

Activité : Marmottes

Concevoir un terrier pour l'hiver

1. À partir de l'entrée, on peut construire deux couloirs. Au bout de chaque couloir, on peut soit creuser une salle soit faire un embranchement vers deux autres couloirs, et ainsi de suite.
2. Pour ne pas se gêner durant leur hibernation, les marmottes vont chacune occuper une salle différente au bout d'un couloir.
3. Chaque marmotte se réveille un nombre précis de fois dans l'hiver.

Marmottes

Décompte des déplacements : Une marmotte dormant à 4 couloirs de l'entrée se réveillant 5 fois dans l'hiver va parcourir 4 x 5 = 20 couloirs (on ne comptera que les allers).

Objectif :

Minimiser les bruits des vibrations des pas susceptibles de réveiller le groupe.

`https://members.loria.fr/MDuflot/files/med/marmottes.html`

Marmottes

Prénoms

- Alice : 6
- Bob : 5
- Eve : 4
- Igor : 2
- Luc : 2
- Regis : 5
- Sophie : 2
- Espérance : 5
- Quentin : 1

LIMOS LABORATOIRE D'INFORMATIQUE DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

“BARBARA RASE BASILE LE BARBIER !” = 97

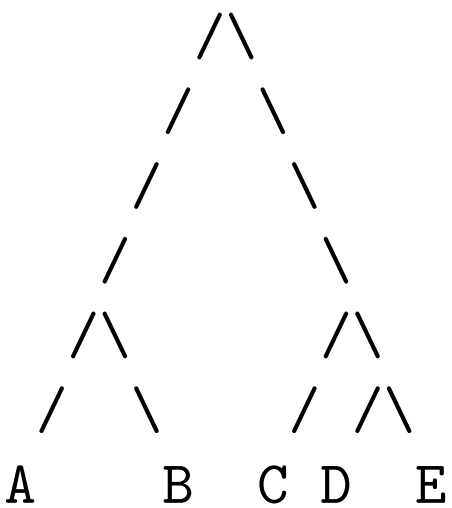
Marmottes : Trouver un bon terrier (Shannon-Fano)

1. Calculer les fréquences d'appartition des lettres.
2. Trier les fréquences .
3. Divisier la liste ordonnée en deux listes de de telle sorte que les sommes des fréquences soient les plus proches possible.
4. Donner zéro à la partie gauche et 1 à la partie droite.
5. Recommencer récursivement les deux derniers points sur la partie droite et la partie gauche.

Utilisé dans la méthode de compression IMPLODE utilisée dans l'algorithme de compression ZIP.

Marmottes : Exemple Shannon-Fano

A=15 B=7 — C=6 — D=6 E=5



Code : A=00 B=01 C=10 D=110 E=111

2.28 bits par symboles

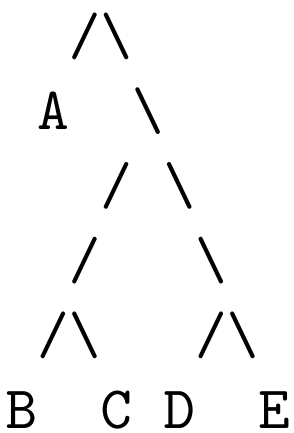
Marmottes : Trouver le meilleur terrier (Huffman)

- on choisit deux parmi les marmottes qui se lèvent le moins souvent, et on les relie par un morceau de terrier et on note la somme des valeurs des deux marmottes.
- on recommence exactement la même chose, mais les deux marmottes reliées à l'étape précédentecomptent maintenant pour une seule marmotte qui se réveillerait 5 fois
- on continue, jusqu'à ce que toutes les marmottes soient reliées en un seul terrier.

Exemple : Cyrano de Bergerac en UTF-8 la version compressée avec Huffman gain de 40 %

Marmottes : Exemple Huffman

A=15 — B=7 C=6 — D=6 E=5

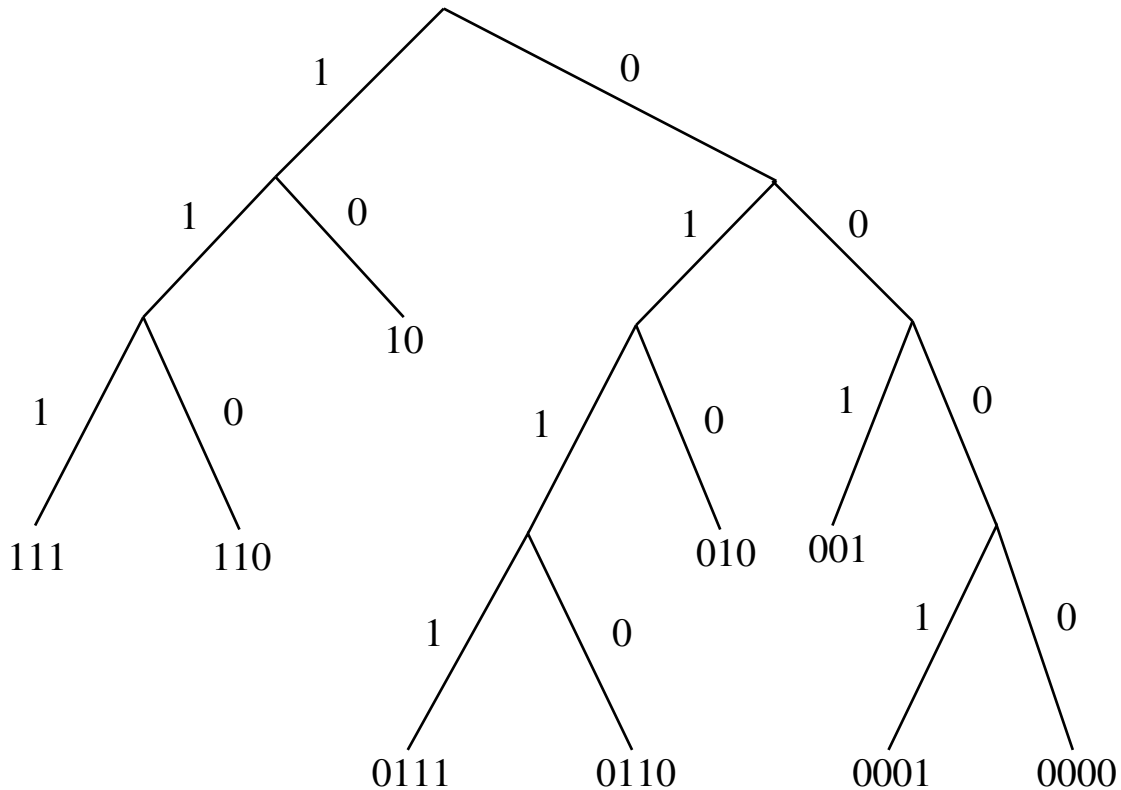


Code : A=0 B=100 C=101 D=110 E=111

2.23 bits par symboles

Arbre de David Albert Huffman 1952

C'est un arbre binaire tel que tout sous-arbre a soit 0 soit 2 fils (il est *localement complet*).



$$\{111, 110, 10, 0111, 0110, 010, 001, 0001, 0000\}.$$

Huffman

Propriété

Un code de Huffman possède la propriété du préfixe.

Preuve : Si un mot de code c_1 est un préfixe de c_2, alors le chemin représentant c_1 dans l'arbre de Huffman est inclus dans le chemin représentant c_2. Comme c_1 et c_2 sont, par définition, associés à des feuilles de l'arbre, $c_1 = c_2$. Il n'existe donc pas deux mots différents dont l'un est le préfixe de l'autre, et le code de Huffman a la propriété du préfixe.

Huffman

Théorème

Tout code qui possède la propriété du préfixe est contenu dans un code de Huffman.

Preuve

Soit un arbre de Huffman complet (toutes les feuilles sont à distance constante de la racine) de hauteur l, la longueur du plus long mot de C. Chaque mot c_i de C est associé à un chemin depuis la racine jusqu'à un nœud.

On peut alors élaguer le sous-arbre dont ce nœud est racine (tous les mots pouvant être représentés dans les nœuds de ce sous-arbre ont c_i pour préfixe).

Tous les mots de C sont toujours dans les nœuds de l'arbre résultant.

On peut effectuer la même opération pour tous les mots.

On a finalement un arbre de Huffman contenant tous les mots de C.

Code

Théorème de Kraft

Il existe un code uniquement déchiffrable sur un alphabet V dont les n mots $\{c_1, \ldots, c_n\}$ sont de longueur $l_1, \ldots, l_n$ si et seulement si

$$\sum_{i=1}^{n} \frac{1}{|V|^{l_i}} \leq 1 \,.$$

Preuve

($\Rightarrow$) Soit C un code uniquement déchiffrable, d'arité q.

Soit m la longueur du plus long mot de C.

Pour $1 \leq k \leq m$, soit r_k le nombre de mots de longueur k.

Nous développons l'expression suivante, pour un entier quelconque u, avec $u \geq 1$:

$$\left(\sum_{i=1}^{n} \frac{1}{q^{l_i}} \right)^u = \left(\sum_{k=1}^{m} \frac{r_k}{q^k} \right)^u .$$

Une fois développée, chaque terme de cette somme est de la forme

$$\frac{r_{i_1} \ldots r_{i_u}}{q^{i_1+\ldots+i_u}}$$

Et en regroupant pour chaque valeur $s = i_1 + \ldots + i_u$, on obtient les termes

$$\sum_{i_1+\ldots+i_u=s} \frac{r_{i_1} \ldots r_{i_u}}{q^s}$$

Preuve

Soit $N(s) = \sum_{i_1+\ldots+i_u=s} r_{i_1} \ldots r_{i_u}$. L'expression initiale s'écrit :

$$(\sum_{i=1}^{n} \frac{1}{q^{l_i}})^u = \sum_{s=u}^{mu} \frac{N(s)}{q^s}$$

$N(s)$ est le nombre de combinaisons de mots de C de longueur totale s.

Preuve

Comme C est uniquement déchiffrable, deux combinaisons de mots de C ne peuvent être égales au même mot sur l'alphabet de C. Comme C est d'arité q, et que $N(s)$ est inférieur au nombre total de messages de longueur s sur cet alphabet, alors $N(s) \leq q^s$. Cela donne

$$\left(\sum_{i=1}^{n} \frac{1}{q^{l_i}} \right)^u \leq mu - u + 1 \leq mu \,.$$

Et donc $\sum_{i=1}^{n} \frac{1}{q^{l_i}} \leq (mu)^{1/u}$, puis $\sum_{i=1}^{n} \frac{1}{q^{l_i}} \leq 1$ quand u tend vers l'infini.

($\Leftarrow$) La réciproque est une conséquence du théorème de McMillan.

Théorème de Mc Millan

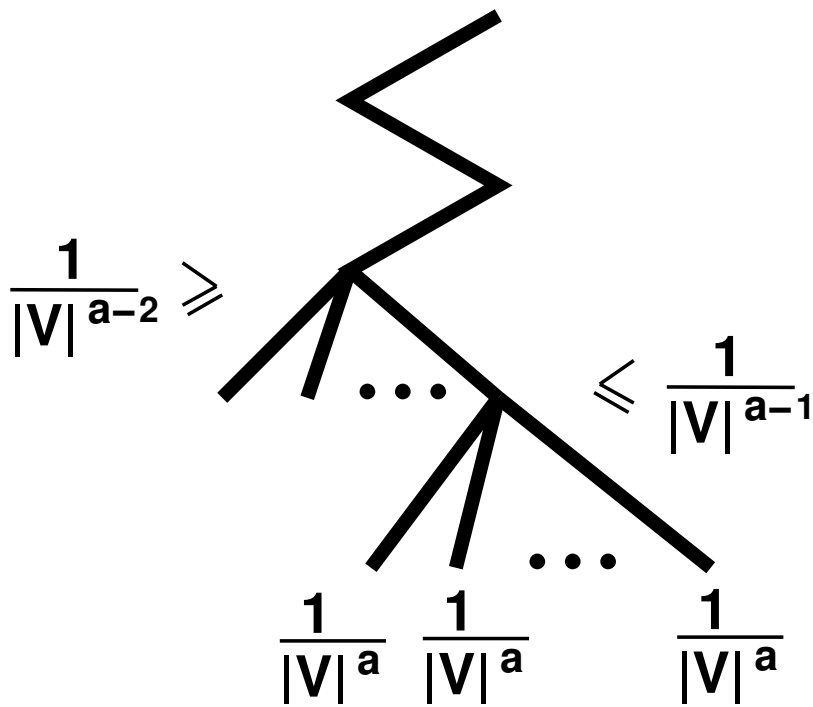
Théorème

Sur un alphabet V, il existe un code qui possède la propriété du préfixe dont les mots $\{c_1, \ldots, c_n\}$ sont de longueur $l_1, \ldots, l_n$ si et seulement si

$$\sum_{i=1}^{n} \frac{1}{|V|^{l_i}} \leq 1 \,.$$

Preuve

On suppose qu'il existe un code instantané (c'est-à-dire qui vérifie la propriété du préfixe) avec les longueurs de mots $l_1, \ldots, l_n$. On a montré qu'on peut construire sa représentation par un arbre de Huffman.



Preuve

On considère un nœud père de feuilles de l'arbre de Huffman. Ce nœud possède donc k feuilles, avec $k \leq |V|$, chacune représentant un mot du code de longueur $l_i = l_{i+1} = \ldots = l_{i+k} = a$ pour une certaine constante a. Ainsi la contribution de ce nœud père particulier à la somme $\sum_{i=1}^{n} \frac{1}{|V|^{l_i}}$ vaut $\frac{k}{|V|^a} \leq \frac{|V|}{|V|^a} = \frac{1}{|V|^{a-1}}$, comme indiqué sur la figure ci-contre. Cette contribution est donc au plus équivalente à la contribution obtenue en remplaçant ce nœud et l'ensemble de ses feuilles par une seule feuille de poids $\frac{1}{|V|^{a-1}}$. En remontant ainsi étage par étage dans l'arbre de Huffman, on voit sur la figure que la contribution d'un sous-arbre de profondeur b est majorée par $\frac{1}{|V|^b}$ et que la somme de l'arbre tout entier est majorée par $\frac{1}{|V|^0} = 1$.

Preuve

($\Leftarrow$) Réciproquement, on suppose que l'inégalité est satisfaite. On cherche à construire un arbre de Huffman dont les mots de code ont pour longueur $l_1, \ldots, l_n = l$, mots que l'on suppose classés par ordre croissant de leurs longueurs.

On part d'un arbre de hauteur l. On procède ensuite par induction.

Pour pouvoir choisir un nœud dans l'arbre, correspondant à un mot de longueur l_k, il faut qu'on puisse trouver dans l'arbre, un sous-arbre complet de hauteur $l - l_k$. Mais, toutes les opérations d'élagage précédentes consistent à enlever des sous-arbres de hauteur plus grande (correspondant à des mots plus courts). Donc, s'il reste au moins $|V|^{l-l_k}$ feuilles dans l'arbre restant, l'arbre restant contient un sous-arbre, de hauteur $l - l_k$. Montrons qu'il reste effectivement $|V|^{l-l_k}$ feuilles dans l'arbre restant.

Preuve

Si on a déjà "placé" les mots de longueur $l_1, \ldots, l_{k-1}$, on a alors enlevé $\sum_{i=1}^{k-1} |V|^{l-l_i}$ feuilles de l'arbre initial par les opérations successives d'élagage. Il reste donc $|V|^l(1 - \sum_{i=1}^{k-1} |V|^{-l_i})$ feuilles. Or, on sait d'après l'inégalité de Kraft que :

$$\sum_{i=k}^{n} \frac{1}{|V|^{l_i}} \leq 1 - \sum_{i=1}^{k-1} \frac{1}{|V|^{l_i}}$$

soit:

$$|V|^l(1 - \sum_{i=1}^{k-1} |V|^{-l_i}) \geq \sum_{i=k}^{n} |V|^{l-l_i} \geq |V|^{l-l_k}$$

On peut donc placer le mot de longueur l_k, et en réitérant l'opération, construire l'arbre de Huffman. Le code de Huffman, de longueurs de mots $l_1, \ldots, l_n$, vérifie la propriété du préfixe.

Algorithme de Huffman: condition

Construction d'un schéma d'encodage optimal d'une source sans mémoire $\mathcal{S} = (S, \mathcal{P})$. L'alphabet de codage est V, de taille q.

Il est nécessaire à l'optimalité du résultat de vérifier que $q-1$ divise $|S|-1$ (afin d'obtenir un arbre localement complet). Dans le cas contraire, il est facile de rajouter des symboles à S, de probabilités d'occurrence nulle, jusqu'à ce que $q-1$ divise $|S|-1$.

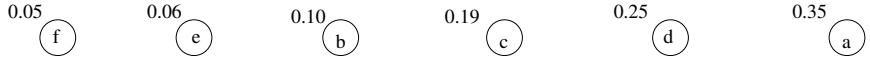
Marmottes : Trouver le meilleur terrier

- on choisit deux parmi les marmottes qui se lèvent le moins souvent, et on les relie par un morceau de terrier et on note la somme des valeurs des deux marmottes.
- on recommence exactement la même chose, mais les deux marmottes reliées à l'étape précédentecomptent maintenant pour une seule marmotte qui se réveillerait 5 fois
- on continue, jusqu'à ce que toutes les marmottes soient reliées en un seul terrier.

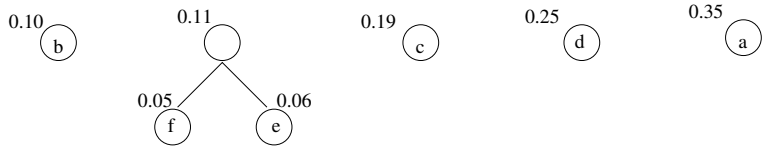
Exemple : Cyrano de Bergerac en UTF-8 la version compressée avec Huffman gain de 40 %

Comment transmettre l'arbre

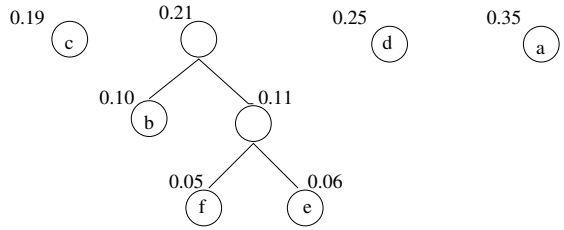
1. Faire un arbre en se basant sur les fréquences d'apparition delettres dans une langue donnée et l'échanger.
2. Construire un arbre pour un texte et l'envoyé avec le texte.



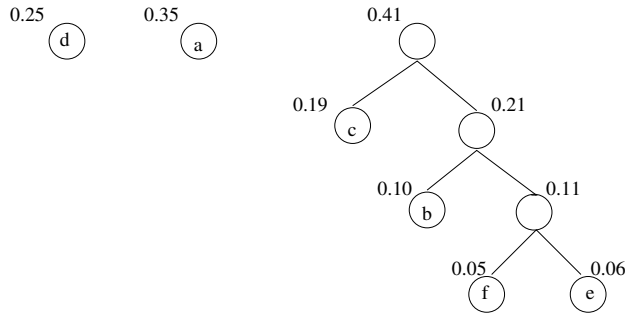
tri dans l'ordre croissant



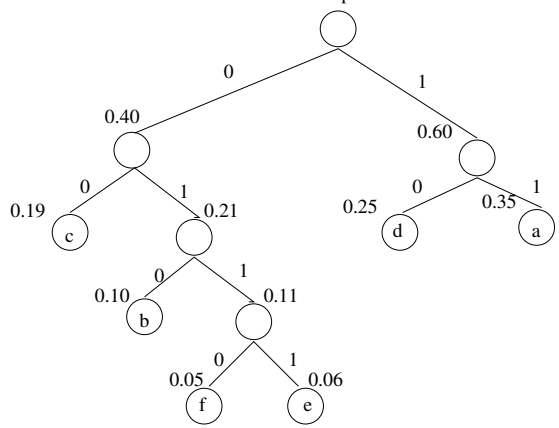
construction de l'arbre et tri du niveau le plus haut



étape suivante



étape suivante



arbre final

Optimalité

Théorème

Un code issu de l'algorithme de Huffman est optimal parmi tous les codes instantanés de $\mathcal{S}$ sur V.

Il ne dit pas que l'algorithme de Huffman est le meilleur pour coder une information dans tous les cas ; mais qu'en fixant pour modèle une source $\mathcal{S}$ sans mémoire sur un alphabet V, il n'y a pas de code plus efficace qui a la propriété du préfixe.

Huffman Dynamique

La commande `pack` de Unix implémente cet algorithme dynamique.

Pour construire l'arbre de Huffman et le mettre à jour, on compte le nombre d'occurrences de chaque caractère et le nombre de caractères déjà lus; on connaît donc, à chaque nouvelle lecture, la fréquence de chaque caractère depuis le début du fichier jusqu'au caractère courant; les fréquences sont donc calculées dynamiquement.

Algorithme Compression Huffman Dynamique

Soit $nb(c)$, le nombre d'occurrences d'un caractère c
Initialiser l'arbre de Huffman (AH) avec le caractère @,
$nb(@) \leftarrow 1$
while on n'est pas à la fin de la source **do**
  Lire un caractère c de la source
  if c'est la première occurrence de c **then**
    $nb(c) \leftarrow 0$
    $nb(@) \leftarrow nb(@) + 1$
    Afficher en sortie le code de @ dans AH suivi de c.
  else
    Afficher le code de c dans AH
  end if
  $nb(c) \leftarrow nb(c) + 1$
  Mettre à jour AH avec les nouvelles fréquences
end while

Algorithme de décompression de Huffman Dynamique

Soit $nb(c)$, le nombre d'occurrences d'un caractère c
Initialiser l'arbre de Huffman (AH) avec le caractère @,
$nb(@) \leftarrow 1$
while on n'est pas à la fin du message codé **do**
  Lire un mot de code c du message (jusqu'à une feuille de AH)
  if $c = @$ **then**
    $nb(@) \leftarrow nb(@) + 1$
    Lire dans c les k bits du message et les afficher en sortie.
    $nb(c) \leftarrow 0$
  else
    Afficher le caractère correspondant à c dans AH
  end if
  $nb(c) \leftarrow nb(c) + 1$
  Mettre à jour AH avec les nouvelles fréquences
end while

Run-Length Encoding (RLE)

Lorsque au moins 3 caractères identiques successifs sont rencontrés, il affiche plutôt un code spécial de répétition suivi du caracteère àa répéter et du nombre de répétitions.

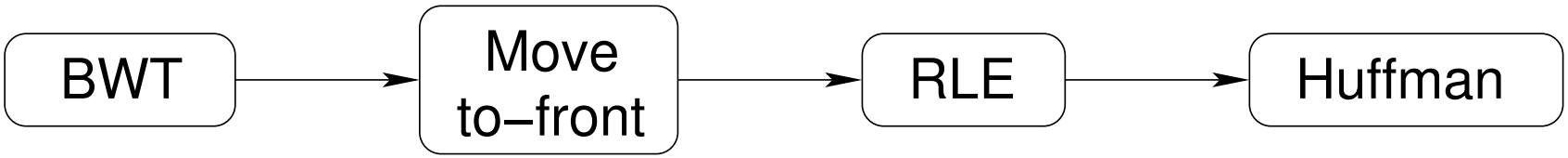
Move-to-Front

Dèes qu'un caractère apparaît dans la source, il est d'abord codeé par sa valeur, puis il passe en début de liste, et sera dorénavant codé par 0, tous les autres caractères étant décalés d'une unité. Move-to-front permet d'avoir plus de codes proches de 0 que de 255.

BWT Michael Burrows et David Wheeler

Idée : trier les caractères d'une chaîne afin que le Move-to-front et le RLE soient les plus efficaces possible.

Bzip2



Algorithme de Lempel-Ziv

Idée: Algorithme de compression par dictionnaire

(utiliser des mots au lieu des lettres)

Exemple : “un gars bon est un bon gars” est codé par “un,gars,bon,est,1,3,2”

Il existe plusieurs variantes de cet algorithme ;

LZ77 et LZ78 sont libres de droits.

Algorithme : LZ77

LZ77 utilise une fenêtre coulissante en 2 parties:

- partie 1: le dictionnaire
- partie 2 : tampon de lecture.

Initialement, la fenêtre est située de façon à ce que le tampon de lecture soit sur le texte, et que le dictionnaire n'y soit pas.

À chaque itération, l'algorithme cherche dans le dictionnaire le plus long facteur qui se répète au début du tampon de lecture ; ce facteur est codé par le triplet (i, j, c) où :

- i est la distance entre le début du tampon et la position de la répétition dans le dictionnaire;
- j est la longueur de la répétition;
- c est le premier caractère du tampon différent du caractère correspondant dans le dictionnaire.

LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

Après avoir codé cette répétition, la fenêtre coulisse de $j + 1$

Exercice : LZ77

1. Coder la séquence “abcdefabcdefabcdefabcdef” avec LZ77.
2. Coder cette séuence avec LZ77 et une taille de fenêtre de recherche tenant sur 3 bits.

Solution : LZ77

"a b c d e f abcdef a bcdefabcde f"

1. Il faut faire attention à bien mettre la dernière lettre (ici le f) dans le dernier triplet et donc se limiter à en reprendre seulement 10, et pour le dernier triplet, il faut bien sûr que la distance soit toujours plus longue que la longueur : $(0, 0, a)$ $(0, 0, b)$ $(0, 0, c)$ $(0, 0, d)$ $(0, 0, e)$ $(0, 0, f)$ $(6, 6, a)$ $(12, 10, f)$.

"a b c d e f abcdef a bcdefa b cde f"

2. Là, la distance et la longueur doivent tenir sur 3 bits, et sont donc comprises entre 0 et 7. Il faut donc couper le dernier bloc en deux morceaux : $(0, 0, a)$ $(0, 0, b)$ $(0, 0, c)$ $(0, 0, d)$ $(0, 0, e)$ $(0, 0, f)$ $(6, 6, a)$ $(6, 6, b)$ $(6, 3, f)$.

Comparaison

Algorithme	Fichier compressé	Taux de compression	Vitesse	
			Codage	Décodage
7-Zip-4.42 (?)	6.20 Mo	61.07%	14.51s	0.46s
rzip-2.1 -9 (LZ77+Go)	6.20 Mo	61.09%	9.26s	2.94s
ppmd-9.1 (Prédictif)	6.26 Mo	60.71%	11.26s	12.57s
bzip2-1.0.3 (BWT)	6.69 Mo	57.96%	7.41s	3.16s
gzip-1.3.5 -9 (LZ77)	7.68 Mo	51.77%	2.00s	0.34s
gzip-1.3.5 -2 (LZ77)	8.28 Mo	47.99%	1.14s	0.34s
WinZip-9.0 (LZW+?)	7.72 Mo	51.55%	5s	5s
compress (LZW)	9.34 Mo	41.31%	1.11s	0.32s
lzop-1.01 -9 (LZW+?)	9.35 Mo	41.32%	6.73s	0.09s
lzop-1.01 -2 (LZW+?)	10.74 Mo	32.54%	0.45s	0.09s
pack (Huffman)	11.11 Mo	30.21%	0.33s	0.27s

Comparaison de la compression d'un fichier de courriels (15.92 Mo de texte, images, exécutables, etc.) par différents algorithmes, sur un PIV 1.5GHz.

Vocabulaire

Taux d'erreurs

Probabilité qu'un bit transmis par le canal soit différent du bit émis, le nombre de bits erronés reçus par rapport au nombre de bits émis.

Ordre de grandeur du taux d'erreurs

Ligne	**Taux d'erreurs**
Disquette	10^{-9} : à 5 Mo/s, 3 bits erronés par minute
CD-ROM optique	10^{-5} : 7ko erronés sur un CD de 700 Mo
DAT audio	10^{-5} : à 48 kHz, deux erreurs par seconde
Disques Blu-ray	$< 2 \cdot 10^{-4}$: environ 1.6Mo erronés pour 32Go
HDD	$> 10^{-4}$: environ 10^6 erreurs pour 10Go
Technologie Flash	10^{-5}
SSD	$< 10^{-5}$
Mémoires à semi-conducteurs	$< 10^{-9}$
Liaison téléphonique	entre 10^{-4} et 10^{-7}
Télécommande infrarouge	10^{-12}
Fibre optique	10^{-9}
Satellite	10^{-6} (Voyager), 10^{-11} (TDMA)
ADSL	10^{-3} à 10^{-9}
Réseau informatique	10^{-12}

Définitions

Le codage transforme le bloc à transmettre $s = [s_1, \ldots, s_k]$ en un bloc de $n = k + r$ symboles, $\phi(s) = [c_1, \ldots, c_k, \ldots, c_n]$ qui comporte r symboles de redondance.

L'ensemble $C_\phi = \left\{\phi(s) \ : s \in V^k\right\}$, image par ϕ de V^k, est appelé un *code*(n, k) ; ses éléments sont appelés *mots de code*.

La fonction de codage ϕ est bien sûr injective.

Rendement R d'un code(n, k)

C'est le taux de chiffres de source contenus dans un mot de code :

$$R = \frac{k}{n}.$$

Exemple

Code de parité

$V = \{0, 1\}$, on code le mot $m = (s_1, \ldots, s_k)$ par $\phi(m) = (s_1, \ldots, s_k, c_{k+1})$ où $c_{k+1} = (\sum_{i=1}^{k} s_i) \mod 2 = \oplus_{i=1}^{k} s_i$. On a donc

$$c_{k+1} \oplus \left(\bigoplus_{i=1}^{k} s_i \right) = 0,$$

$c_{k+1} = 1$ ssi le nombre de bits non nuls dans m est impair.

Exercice : Numéro de Sécurité sociale

Construction

Nombre de $n = 15$ chiffres : un numéro d'identification K sur $k = 13$ chiffres suivi de la clef C de $r = 2$ chiffres calculée pour que $K + C$ soit un multiple de 97.

1. Quel est la clef du numéro de sécurité sociale 2.63.05.38.516.305 ?
2. Quel est le rendement de ce code ?
3. Combien d'erreurs de chiffres, la clef du numéro de sécurité sociale permet-elle de détecter ?

Solution : Numéro de Sécurité sociale

1. La clef C est telle que $2630538516305 + C$ est multiple de 97 ; donc $C = 97 - 2630538516305 \mod 97 = 96 - 33 = 64$.
2. $R = \frac{13}{15} \simeq 86,67\%$.
3. Le code permet de détecter une erreur sur un seul chiffre. Soit $n = [K, C]$ l'entier de 15 chiffres associé au numéro de sécurité sociale suivi de la clef. Une erreur sur un seul chiffre c_i d'indice i donne $n' = n + e \cdot 10^i$ avec $e \neq 0$ et $-c_i \leq e \leq 9 - c_i$. On a alors $n' \mod 97 = e \cdot 10^i \neq 0$ car e et 10 sont premiers avec 97 ; l'erreur est détectée.
   Toutefois contre, une erreur sur deux chiffres n'est pas toujours détectée. Par exemple, deux erreurs de chiffres transforment le numéro de sécurité sociale valide 2.63.05.38.516.301.68 en le numéro

   2.63.05.38.516.398.68 qui est aussi valide.

Code-barres EAN (European Article Numbering)

Dérivé du code américain *UPC-A* sur 12 chiffres : un zéro est ajouté en tête du numéro UPC-A, $c_{13} = 0$.

Code barre

Composé d'un numéro sur 13 chiffres $c_{12} - -c_{11}c_{10}c_9c_8c_7c_6 - -c_5c_4c_3c_2c_1c_0$. Les 12 chiffres de gauche $c_{12}, \ldots, c_1$ identifient le produit ; c_0 est le chiffre de parité: Soit

- $a = \sum_{i=1}^{6} c_{2i}$ la somme des chiffres de rang pair
- $b = \sum_{i=1}^{6} c_{2i-1}$ celle des chiffres de rang impair.

$c_0 = 10 - (a + 3 \times b) \mod 10$.

EAN-128 permet de coder des lettres.

Le numéro ISBN (*International Standard Book Number*)

ISBN

Formé de 10 chiffres $c_{10}c_9c_8c_7c_6c_5c_4c_3c_2c_1$, structurés en quatre segments $A - B - C - D$ séparés par un tiret. Les neuf premiers chiffres $A - B - C$ identifient le livre : A identifie la communauté linguistique, B le numéro de l'éditeur et C le numéro d'ouvrage chez l'éditeur.

La clef de contrôle $D = c_1$ est un symbole de parité qui est soit un chiffre entre 0 et 9, soit la lettre X qui représente 10.

Cette clef c_1 est telle que $\sum_{i=1}^{10} i \times c_i$ est un multiple de 11 ; autrement dit, $c_1 = 11 - \left(\sum_{i=2}^{10} i \times c_i\right) \mod 11$.

Norme ISO 2108 à partir du 1 janvier 2007

Identifie les livres par EAN-13 : les trois premiers chiffres valent 978, les 9 suivants sont les 9 premiers chiffres du code ISBN (ABC) et le dernier est le chiffre de contrôle EAN-13.

Exercice : ISBN

- Compléter la séquence 210-059-911 en un numéro ISBN correct.
- Quel est son code-barres EAN-13 associé ?

Solution : ISBN

On obtient $c_1 = 11 - \sum_{i=2}^{10} c_i =$
$11 - (1*2 + 1*3 + 9*4 + 9*5 + 5*6 + 0*7 + 0*8 + 1*9 + 2*10 \mod 11) = 9$, soit le code ISBN 210-059-911-9.
Le code EAN-13 est 9-782100-59911-c_0 avec, en utilisant un chiffre sur deux,
$c_0 = (10 - (9 + 8 + 1 + 0 + 9 + 1 + 3*(7 + 2 + 0 + 5 + 9 + 1))) \mod 10 = 0$. On obtient donc le code EAN-13 978-2-10-059911-0, comme vous pouvez le vérifier sur la couverture.

RIB d'un compte bancaire $B - G - N - R$

Il comporte 23 caractères. B et G sont des nombres de 5 chiffres qui identifient respectivement la banque et le guichet.

N est composé de 11 alphanumériques, que l'on convertit en une séquence S_N de 11 chiffres en remplaçant les lettres éventuelles cycliquement comme suit : A et J en 1 ; B, K, S en 2 ; C, L, T en 3 ; D, M, U en 4 ; E, N, V en 5 ; F, O, W en 6 ; G, P, X en 7 ; H, Q, Y en 8 ; I, R, Z en 9.

Enfin la clef RIB R est un nombre de deux chiffres qui est tel que, mis bout à bout, B, G, S_N et R forment un nombre n de 23 chiffres multiple de 97. Autrement dit, $R = 97 - ([BGS_N] \times 100) \mod 97$.

LUHN-10 : carte bancaire

Les 16 chiffres $c_{15}c_{14}c_{13}c_{12} - c_{11}c_{10}c_9c_8 - c_7c_6c_5c_4 - c_3c_2c_1c_0$ d'une carte bancaire

Vérification de la validité

Multiplier les chiffres d'indice impair par 2 et on soustrait 9 si le nombre obtenu est supérieur à 9. Tous ces chiffres sont additionnés et sommés aux autres chiffres d'indice pair. La somme finale doit être un multiple de 10.

7 chiffres au dos de la carte

- Les 4 premiers sont les 4 derniers chiffres du numéro de carte.
- Les 3 derniers sont le résultat d'un calcul qui fait intervenir, outre N, les 4 chiffres mm/aa de sa date d'expiration (N et $mmaa$ sont combinés avant d'être chiffrés par un 3-DES-EDE à l'aide d'une clef universelle appelée clef de vérification de carte; trois des chiffres du résultats sont alors sélectionnés pour donner le CVV.

Codes de Redondance Cyclique : CRC

CRC

Un mot binaire $u = [u_{m-1} \dots u_0] \in \{0,1\}^m$ est représenté par un polynôme $P_u(X) = \sum_{i=0}^{m-1} u_i X^i$ de $F_2[X]$.

Par exemple, $u = [10110]$ est représenté par le polynôme $P_u = X^4 + X^2 + X$.

CRC

Polynôme générateur

Un code CRC est caractérisé par un P_g de degré r : $P_g(X) = X^r + \sum_{i=0}^{r-1} g_i X^i$ dans $F_2[X]$.

Le mot source binaire $s = [s_{k-1} \dots s_0]$ associé au polynôme $P_s(X) = \sum_{i=0}^{k-1} s_i X^i$, est codé par le mot de code binaire $c = [c_{n-1} \dots c_0] = [s_{k-1} \dots s_0 c_{r-1} \dots c_0]$ où $[c_{r-1} \dots c_0]$ est la représentation du reste de la division euclidienne de $X^r.P_s$ par P_g. La multiplication de P_s par X^r se traduisant par un décalage de bits, le polynôme $P_c(X) = \sum_{i=0}^{n-1} c_i X^i$ vérifie donc

$$P_c = P_s.X^r + (P_s.X^r \mod P_g).$$

CRC

Exemple

Avec le polynôme générateur $P_g = X^2 + 1$, le mot $u = [10110]$ est codé en ajoutant les bits de redondance du polynôme $(X^4 + X^2 + X)X^2 \mod X^2 + 1 = X^6 + X^4 + X^3 \mod X^2 + 1 = X$ qui correspond aux bits de redondance [10]. Le mot codé est donc $\phi(u) = [1011010]$.

Le calcul du reste d'une division euclidienne est en $O(n \log n)$ par un algorithme de produit de polynômes par DFT.

Décodage CRC

Remarque

Dans $F_2[X]$ on a aussi $P_c = P_s.X^r - (P_s.X^r \mod P_g)$; le polynôme P_c est donc un multiple de P_g.

À la réception de $c' = [c'_{n-1} \dots c'_0]$, il suffit de calculer le reste $R(X)$ du polynôme $P_{c'}$ par le polynôme P_g. Si ce reste est nul, le mot reçu est un mot de code et l'on ne détecte pas d'erreur. Sinon, si $R(X) \neq 0$, il y a eu erreur lors de la transmission.

Détection d'erreur

1 erreur de transmission

Les coefficients du polynôme $P_e = P_{c'} - P_c$ valent 0 sur les bits non erronés et 1 aux bits de c modifiés. L'erreur est détectée si et seulement si P_e n'est pas un multiple de P_g.

Si un des multiples de degré au plus $n-1$ de P_g possède j monômes, alors le code CRC généré par g ne peut détecter qu'au plus $j-1$ erreurs.

Réciproquement, pour qu'un code CRC détecte toute erreur portant sur au plus j bits, il faut et il suffit que tout polynôme de degré au plus $n-1$ et ayant au plus j monômes ne soit pas multiple de P_g.

L'erreur P_e égale à P_g, ou multiple de P_g, n'est pas détectée.

Le nombre d'erreurs de bit détectées est au plus $\omega(P_g)$ où $\omega(P_g)$ est le nombre de coefficients non nuls de P_g.

Propriété

Un code CRC de générateur $P_g(X) = X^r + \sum_{i=1}^{r-1} g_i X^i + 1$ détecte toute erreur portant sur $r-1$ bits consécutifs ou moins.

Preuve : Une séquence d'erreurs portant sur au plus $r-1$ bits consécutifs commençant au $i$ème bit du mot est associée à un polynôme $P_e = X^i Q$ avec Q de degré inférieur strictement à r ; Q n'est donc pas multiple de P_g. Et comme $g_0 = 1$, P_g est premier avec X ; on en déduit que P_e n'est pas multiple de P_g donc l'erreur est bien détectée.

Distance d'un code

Un code (n, k) est dit *t-détecteur* (resp. *t-correcteur*) s'il permet de détecter (resp. corriger) toute erreur portant sur t chiffres ou moins lors de la transmission d'un mot de code de n chiffres.

Exemples

- L'ajout d'un bit pour contrôler la parité des 7 bits qui le précèdent est un code systématique par bloc. C'est un code (8,7). Il est 1-détecteur et 0-correcteur avec un rendement de 87,5%.
- Le contrôle de parité longitudinale et transversale sur 21 bits (avec ajout de 11 bits de contrôle) est un code (32,21). Il est 1-correcteur avec un rendement de 65,625%.

Poids de Hamming

Définition

le nombre de composantes non nulles de x, noté $w(x)$, où $x = (x_1, \ldots, x_n) \in V^n$

$$w(x) = |\{i \in \{1, \ldots, n\} / x_i \neq 0\}|.$$

w satisfait l'inégalité triangulaire : $w(x \oplus y) \leq w(x) + w(y)$.

Distance de Hamming

$$d_H(x, y) = |\{i \in \{1, \ldots, n\} / x_i \neq y_i\}|.$$

Définition

Soit C un code (n, k) sur F. On dit que C est un *code linéaire* de longueur n et de dimension k lorsque C est un sous-espace vectoriel de F^n de dimension k.

Exemple

Considérons le code $(4, 3)$ de parité sur F_2. Le mot de code associé au mot d'information $x^t = [x_0, x_1, x_2]$ est alors $b^t = [b_0, b_1, b_2, b_3]$ défini par : $b_0 = x_0$, $b_1 = x_1$, $b_2 = x_2$, $b_3 = x_0 + x_1 + x_2 \mod 2$.
Ce code est un code linéaire sur F_2 de matrice génératrice.

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \text{ et on a alors : } b^t = x^t G$$

Codage

La matrice génératrice G, s'écrit sous la forme dite *systématique*:

$$G = \left[\begin{array}{c|c} & \\ L & R \\ & \end{array}\right]$$

où L est une matrice carrée $k \times k$ inversible, et R une matrice $k \times r$.

On construit alors $G' = L^{-1}G$, soit :

$$G' = \left[\begin{array}{c|c} & \\ I_k & T = L^{-1}R \\ & \end{array}\right]$$

La matrice G' est appelée matrice génératrice *normalisée (ou canonique)* du code C.

Tout mot source $u \in F^k$ (vecteur ligne) est codé par $\phi(u) = u^t G' = [u^t, u^t.T]$.

Rappel : $(AB)^{\mathsf{T}} = B^{\mathsf{T}} A^{\mathsf{T}}$.

Proprieté

Soit H la matrice $r \times n$: $H = \left[\begin{array}{c|c} T^t & -I_r \end{array}\right]$. Alors $x^t =$

$[x_1, \ldots x_n] \in C$ si et seulement si $Hx = 0$.
H est appelée *matrice de contrôle* de C.

Preuve : Soit $x^t \in C$. Il existe $u \in F^k$ tel que $x^t = u^t[I_k|T]$. D'où $x^t H^t = u^t[I_k|T] \left[\begin{smallmatrix} T \\ -I_r \end{smallmatrix}\right]$. Or, $[I_k|T] \left[\begin{smallmatrix} T \\ -I_r \end{smallmatrix}\right] = [T - T] = [0]$; par conséquent pour tout x, $Hx = 0$.

Réciproquement, si $Hx = 0$ alors
$x^t H^t = [x_1, \ldots, x_n] \left[\begin{smallmatrix} T \\ -I_r \end{smallmatrix}\right] = [0, \ldots, 0]_r$. La composante j de ce système (pour $1 \leq j \leq r$) donne alors
$[x_1, \ldots, x_k] \left[\begin{smallmatrix} T_{1,j} & \cdots & T_{k,j} \end{smallmatrix}\right]^t - x_{k+j} = 0$. Soit
$x_{k+j} = [x_1, \ldots, x_k] \left[\begin{smallmatrix} T_{1,j} & \cdots & T_{k,j} \end{smallmatrix}\right]^t$. D'où également
$[x_{k+1}, \ldots, x_{k+r}] = [x_1, \ldots, x_k]T$. Finalement on a
$[x_1, \ldots, x_n] = [x_1, \ldots, x_k][I_k|T] = [x_1, \ldots, x_k]G'$, et $x \in C$.

Pour détecter une erreur si on reçoit un mot y, il suffit donc de calculer Hy, qu'on appelle le *syndrome d'erreurs*. On détecte une erreur si et seulement si $Hy \neq 0$.

On cherche alors à calculer le mot émis x à partir de y. Pour cela, on calcule le vecteur d'erreurs $e = y - x$. Ce vecteur e est l'unique élément de F^n de poids de Hamming $w_H(e)$ minimal tel que $He = Hy$. En effet, si C est t-correcteur, il existe un unique $x \in C$, tel que $d_H(x, y) \leq t$. Comme $d_H(x, y) = w_H(x - y)$, on en déduit qu'il existe un unique $e \in F^n$, tel que $w_H(e) \leq t$. De plus $He = Hy - Hx = Hy$ car $Hx = 0$.

On construit alors une table de $|F|^r$ entrées, dans laquelle chaque entrée i correspond à un élément unique z_i de $\mathrm{Im}(H)$; dans l'entrée i, on stocke le vecteur $e_i \in F^n$ de poids minimal et tel que $He_i = z_i$.

La correction est alors triviale. Quand on reçoit y, on calcule Hy et l'entrée i de la table tel que $z_i = Hy$. On trouve alors e_i dans la table et on renvoie $x = y - e_i$.

Exemple

Soit le code de Hamming binaire $(7, 4)$ et G sa matrice génératrice

$$G = \left[\begin{array}{ccccccc} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right].$$

Dans ce cas, on a :

$$L = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix} \; R = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{ et } T = L^{-1}R = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

Exemple

La matrice génératrice canonique G' et la matrice de contrôle H sont données par :

$$G' = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \quad \text{et} \quad H = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

Si l'on recoit le mot $y = 1111101$, alors le calcul $Hy = [0\,1\,0]^t$ montre qu'il y a erreur. Pour corriger l'erreur, il suffit (la table à $2^3 = 8$ entrées n'est pas nécessaire dans cet exemple simple) de constater que le vecteur $e = 0000010$ est de poids (de Hamming) minimal et tel que $He = Hy$. Donc la correction de y est $x = y + e = 1111111$.

Chiffrement de McEliece

- **Kgen :** Considérer un code (n, k, d), de matrice génératrice G, corrigeant jusqu'à t erreurs ; choisir aléatoirement une matrice $k \times k$, S, inversible sur le corps du code ; choisir aléatoirement une matrice de permutation $n \times n$, P (une seule valeur non nulle, 1, dans chaque ligne et colonne).
- **Clef publique :** $(\mathcal{G} = S \cdot G \cdot P, t)$.
- **Clef privée :** (S, G, P).
- **Chiffrement de m :** Choisir aléatoirement un vecteur e de poids de Hamming inférieur à t ; Calculer $c = m \cdot \mathcal{G} + e$.
- **Déchiffrement :** Calculer $a = c \cdot P^{-1}$; décoder b à partir de a avec le code correcteur C ; résoudre le système linéaire $b = m \cdot S$, pour m.

Correction du Chiffrement de McEliece

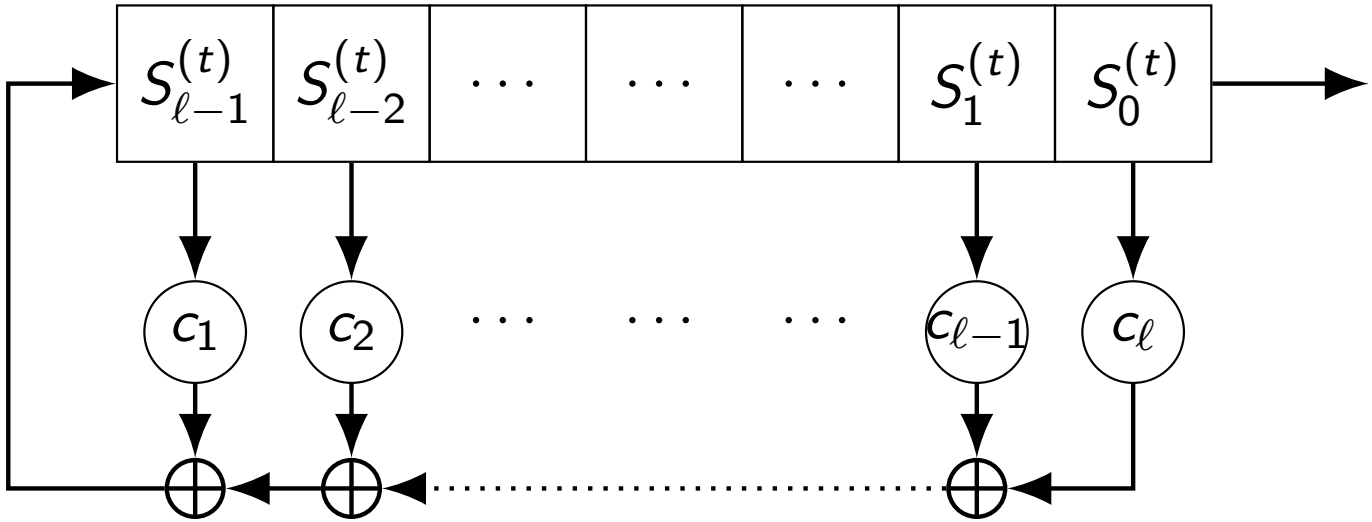
$a = c \cdot P^{-1} = m \cdot S \cdot G + eP^{-1} = (mS) \cdot G + e'$ où e' a le même poids de Hamming que e, toujours inférieur à t. Ainsi, la correction d'erreur sur a donne $b = m \cdot S$. Ensuite, $m = b \cdot S^{-1}$ est la seule solution du système puisque S est inversible.

Taille des clefs

La taille de code typique nécessaire pour obtenir une équivalence de sécurité à 128 bits de clef symétrique est de l'ordre de $(2960, 2288)$.

La matrice génératrice doit être 2288×2960, soit $6\,772\,480$ bits (environ 6.5 Mo) alors qu'une clef RSA serait sur 3072 bits et une clef ECC sur 256 bits.

Linear Feedback Shift Register



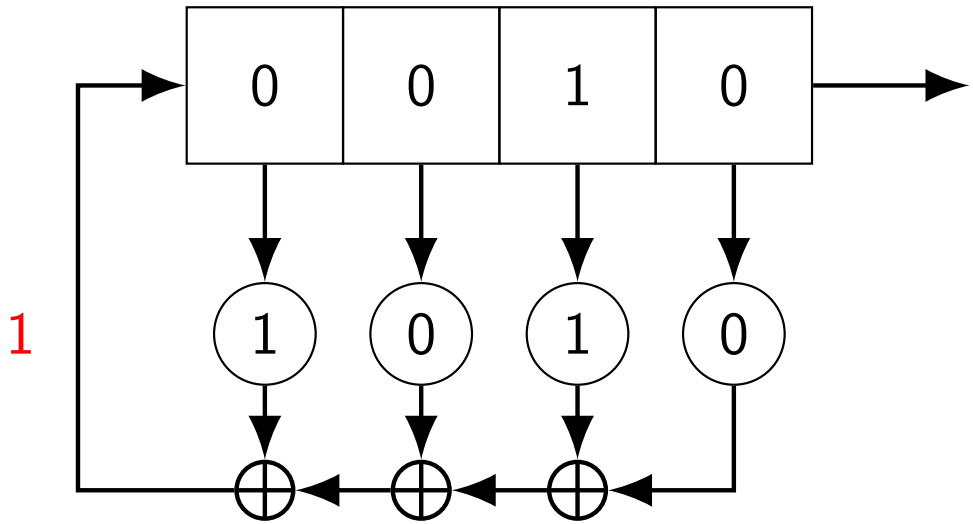
- Length of the register is ℓ, $s^{(0)}$ is the seed
- $\forall c_i \in \{0, 1\}$

$$\forall t \geq 0, s_{\ell-1}^{(t+1)} = \sum_{i=1}^{\ell} c_i s_{\ell-i}^{(t)}$$

$$Shift : s_i^{(t+1)} = s_{i+1}^{(t)}, \forall i, 0 \leq i \leq \ell - 2$$

Example

Seed $s^{(0)} = 0010$ and $c_1 = 1$ $c_2 = 0$ $c_3 = 1$ and $c_4 = 0$

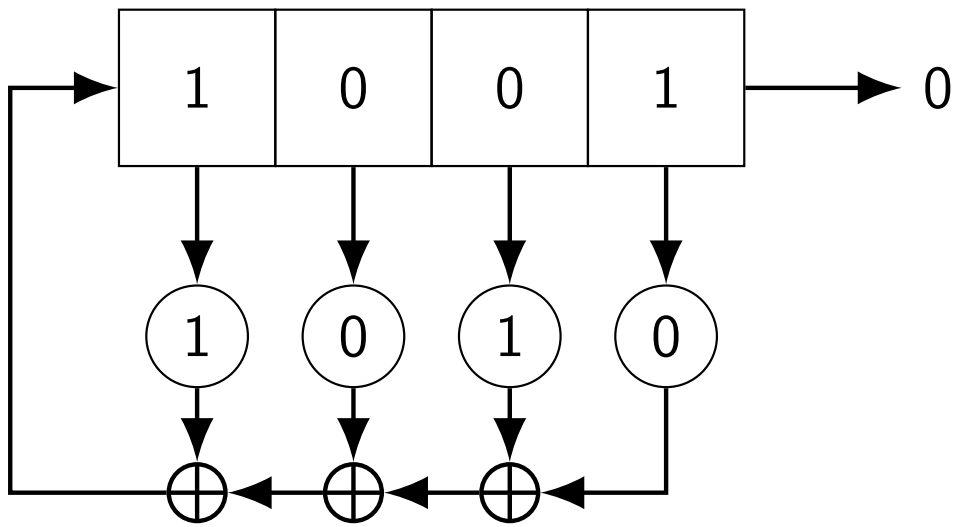


$$
\begin{aligned}
s_3^{(1)} &= (s_3^{(0)} \cdot c_1) \oplus (s_2^{(0)} \cdot c_2) \oplus (s_1^{(0)} \cdot c_3) \oplus (s_0^{(0)} \cdot c_4) \\
&= (0 \cdot 1) \oplus (0 \cdot 0) \oplus (1 \cdot 1) \oplus (0 \cdot 0) \\
&= 1
\end{aligned}
$$

Example first output bit

$c_1 = 1$ $c_2 = 0$ $c_3 = 1$ and $c_4 = 0$

$s_2^{(1)} = s_3^{(0)}, s_1^{(1)} = s_2^{(0)},$ and $s_0^{(1)} = s_1^{(0)}$



Definitions

Period

A serie $(s_n)_{n\in\mathbb{N}}$ is periodic of perido p if $s_{n+p} = s + n, \forall n$.

Retroaction polynomial

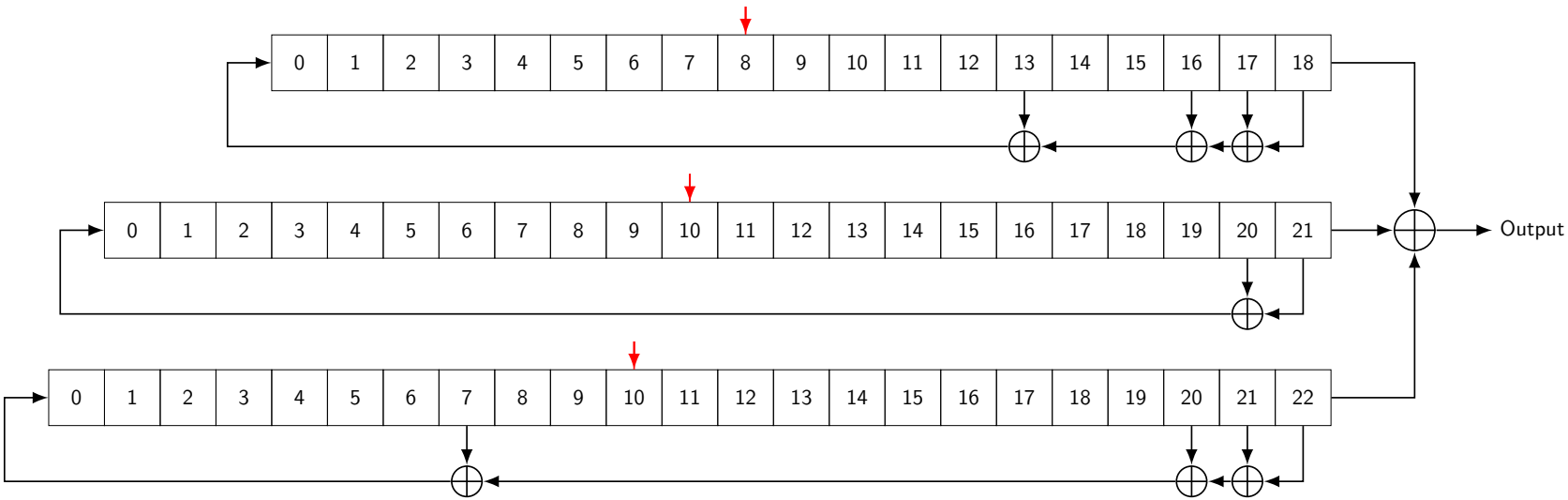
$p(X) \in \mathbb{F}_2[X]$:

$$p(X) = 1 + \sum_{i=1}^{\ell} c_i X^i$$

A5/1 used for GSM in Europe 1994

Red bits are used to determine the majority amont 3 values.

Winner registers are shift.



$$x^{19} + x^{18} + x^{17} + x^{14} + 1$$

$$x^{22} + x^{21} + 1$$

$$x^{23} + x^{22} + x^{21} + x^{8} + 1$$

Attack on A5/1

- 1997, Golic attack in $2^{40.16}$
- 2000, Alex Biryukov, Adi Shamir and David Wagner : few minutes with 2 minutes of plain communication (using in total 300 Go data, in 2^{48} steps).
- 2000 Eli Biham et Orr Dunkelman attack in $2^{39.91}$ with $2^{20.8}$ bits fo data.
- Improvement by Maximov et al for one minute of computation and few clear secands of plain communication.

  Maximov, Alexander; Thomas Johansson; Steve Babbage (2004). "An Improved Correlation Attack on A5/1". Selected Areas in Cryptography 2004: 1–18.

  Barkan, Elad; Eli Biham (2005). "Conditional Estimators: An Effective Attack on A5/1". Selected Areas in Cryptography 2005: 1–19.

LIMOS
LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

- 13 December 2013, with Snowden affirmations, NSA can

RC4 by Ron Rivest in 1987

"Rivest Cipher 4" or "Ron's Code" is a stream cipher used in TLS (Transport Layer Security) and WEP (Wired Equivalent Privacy).

- The key-scheduling algorithm (KSA)
- The pseudo-random generation algorithm (PRGA)

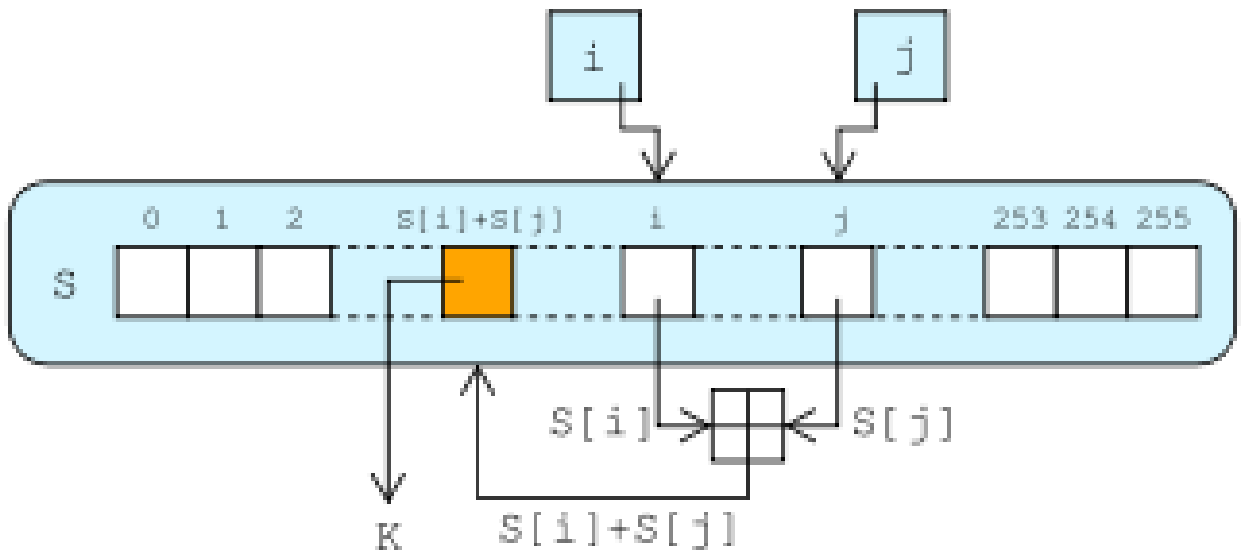
KSA use a key of length between 40 – 128 bits

- Array "S" is initialized to the identity permutation.
- 256 iterations with mixes of bytes of the key at the same time.

```
j := 0
for i from 0 to 255
    j := (j + S[i] + key[i mod keylength]) mod 256
    swap values of S[i] and S[j]
endfor
```

Pseudo-Random Generation Algorithm (PRGA)

```
i := 0; j := 0;
while GeneratingOutput:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap values of S[i] and S[j]
    K := S[(S[i] + S[j]) mod 256]
    output K
```



Recent attacks on RC4

- Fluhrer, Mantin and Shamir attack 2001
- Klein's attack 2005
- John Leyden (2013-09-06). "That earth-shattering NSA crypto-cracking: Have spooks smashed RC4?"
- "Fresh revelations from whistleblower Edward Snowden suggest that the NSA can crack TLS/SSL connections, the widespread technology securing HTTPS websites and virtual private networks (VPNs)."
- " Attack relies on statistical flaws in the keystream generated by the RC4 algorithm. It relies on getting a victim to open a web page containing malicious JavaScript code that repeatedly tries to log into Google's Gmail, for example. This allows an attacker to get hold of a bulk of traffic needed to perform cryptanalysis."

  Nadhem AlFardan, Dan Bernstein, Kenny Paterson, Bertram Poettering and Jacob Schuldt. "On the Security of RC4 in TLS". Royal Holloway University of London. Retrieved March 13, 2013.

RC4 bad

```
int main (int argc , char * argv []) {
    unsigned char S [256] , c;
    unsigned char key [] = KEY;
    int klen = strlen ( key );
    int i,j,k;

    /* Init S[] */
    for (i =0; i <256; i++)
        S[i] = i;

    /* Scramble S[] with the key */
    j = 0;
    for (i =0; i <256; i++) {
         j = (j+S[i]+ key [i% klen ]) % 256;
         S[i] ^= S[j];
         S[j] ^= S[i];
         S[i] ^= S[j];
     }
     /* Generate the keystream and cipher the input stream */
     i = j = 0;
     while ( read (0, &c, 1) > 0) {
          i = (i +1) % 256;
          j = (j+S[i]) % 256;
          S[i] ^= S[j];
          S[j] ^= S[i];
          S[i] ^= S[j];
          c ^= S[(S[i]+S[j]) % 256];
          write (1, &c, 1);
     }}
```

RC4 Good

```
int main (int argc , char * argv []) {
    unsigned char S [256] , c;
    unsigned char key [] = KEY;
    int klen = strlen ( key );
    int i,j,k;

    /* Init S[] */
    for (i =0; i <256; i++)
        S[i] = i;

    /* Scramble S[] with the key */
    j = 0;
    for (i =0; i <256; i++) {
         j = (j+S[i]+ key [i% klen ]) % 256;
         k = S[i];
         S[i] = S[j];
         S[j] = k;
     }
     /* Generate the keystream and cipher the input stream */
     i = j = 0;
     while ( read (0, &c, 1) > 0) {
          i = (i +1) % 256;
          j = (j+S[i]) % 256;
          k = S[i];
          S[i] = S[j];
          S[j] = k;
          c ^= S[(S[i]+S[j]) % 256];
          write (1, &c, 1);
     }}
```

Swap

Classical way (using temporary variable)

```
tmp = a
a = b
b = tmp
```

Without but with + or XOR

```
a = a+b
b = a-b
a = a-b

a = a^b
b = a^b
a = a^b
```

Swap

The buggy adaptation

```
S[i] = S[i]^S[j]
S[j] = S[i]^S[j]
S[i] = S[i]^S[j]
```

because when $i = j$, we have

```
S[i] = S[i]^S[i]
S[i] = S[i]^S[i]
S[i] = S[i]^S[i]
```

- instead of exchanging a value with itself, we set it to 0
- the RC4 state fills up with 0
- the bitstream quickly degrades to a sequence of 0
- encryption does not happen anymore

4 LFSR

De longueurs 25, 31, 33 et 39, pour un total 128 bits.

- $X^{39} + X^{36} + X^{28} + X^4 + 1 \rightarrow a$
- $X^{33} + X^{28} + X^{24} + X^4 + 1 \rightarrow b$
- $X^{31} + X^{24} + X^{16} + X^{12} + 1 \rightarrow c$
- $X^{25} + X^{20} + X^{12} + X^8 + 1 \rightarrow d$

Ces polynômes sont primitifs modulo 2 pour une période totale de $\text{ppcm}(2^{39}-1, 2^{33}-1, 2^{31}-1, 2^{25}-1) = 7 \cdot 23 \cdot 31 \cdot 79 \cdot 89 \cdot 601 \cdot 1801 \cdot 8191 \cdot 121369 \cdot 599479 \cdot 2147483647 \approx 2^{125}$.

Bluetooth

État initial $(\{cl_{-1}\ ; ch_{-1}\ ; cl_0\ ; ch_0\} = IV \in \{0,1\}^4)$

fonction discrète non-linéaire f produit z_t

1. $z_t = a_t \oplus b_t \oplus c_t \oplus d_t \oplus cl_t$;
2. $s_{t+1} = \lfloor \frac{a_t+b_t+c_t+d_t+2ch_t+cl_t}{2} \rfloor \in [0,3]$, codé sur deux bits (sl_{t+1}, sh_{t+1}) ;
3. $cl_{t+1} = sl_{t+1} \oplus cl_t \oplus cl_{t-1} \oplus ch_{t-1}$;
4. $ch_{t+1} = sh_{t+1} \oplus ch_t \oplus cl_{t-1}$.

Conclusion

A retenir

- Code
- Chiffrement