MAT239 – ALGEBRE APPLIQUEE

Vanessa Vitse

Table des matières

Première partie

Cryptographie à clef publique

1 Principe

Deux personnes, usuellement dénommées Alice et Bob, souhaitent s'échanger une donnée sensible mais ne disposent que d'un mode de transmission non sécurisé (internet typiquement) où les messages qui transitent peuvent être lus. L'objectif premier pour Alice et Bob est de s'assurer qu'aucune autre personne ne sera en mesure de pouvoir comprendre cette information : c'est le problème de la *confidentialité* des données. La cryptographie permet de résoudre ce point en utilisant des protocoles de *chiffrement* et *déchiffrement*. Selon *principe de Kerckoffs* (1883), ces protocoles doivent être connus de tous de façon à ce que n'importe qui soit en mesure de les utiliser, par contre ils nécessitent l'utilisation de *clefs secrètes* pour qu'Alice et Bob soient les seuls à pouvoir déchiffrer. La sécurité du chiffrement repose donc essentiellement sur l'utilisation de ces clefs.

1.1 Cryptographie symétrique et asymétrique

En cryptographie, il existe deux types de protocoles suivant que les clefs de chiffrement et déchiffrement sont les mêmes ou non. Lorsqu'Alice et Bob partagent une même clef secrète (donc c'est la même clef qui sert à chiffrer et déchiffrer), on parle de cryptographie symétrique – cette branche de la cryptographie ne sera abordée qu'allusivement dans ce cours. Par opposition, en cryptographie asymétrique ou à clef publique, la clef qui sert à envoyer des messages chiffrés à Alice par exemple est publique (elle mis à disposition de quiconque dans un annuaire, ou sur la page web d'Alice...) et distincte de la clef privée d'Alice qui permet à elle seule de déchiffrer les cryptogrammes qu'on lui aurait envoyés. Ce principe est assez facilement illustré par le fonctionnement d'une boîte aux lettres classique : n'importe qui peut mettre une enveloppe dans la boîte aux lettres d'Alice du moment qu'il connaît son adresse postale, par contre seule Alice qui a la clef de la boîte peut récupérer son courrier.

Plus précisément, supposons que Bob souhaite envoyer un message confidentiel à Alice ; on note
- E resp. D les fonctions de chiffrement et déchiffrement connues de tous
- m le message en clair et c le message chiffré (aussi appelé *cryptogramme*)
- $K_{A,p}$ la clef de chiffrement publique d'Alice connue de tous, $K_{A,s}$ la clef de déchiffrement qui est la clef privée d'Alice.

Le protocole de chiffrement consiste simplement à calculer la valeur de la fonction $c = E_{K_{A,p}}(m)$ et n'importe qui peut faire ce calcul et envoyer un message chiffré à Alice. Pour le déchiffrement, Alice va calculer $D_{K_{A,s}}(c)$ et elle est la seule à pouvoir le faire puisqu'elle seule connaît $K_{A,s}$. Mathématiquement, ceci ne peut fonctionner que si la composition $D_{K_{A,s}} \circ E_{K_{A,s}}$ est égale à la fonction identité.

Le problème de trouver effectivement de telles fonctions et jeux de clefs est loin d'être évident, et il a fallu attendre les années 1970 pour que trois chercheurs dénommés Rivest, Shamir et Adleman résolvent ce challenge mathématique et donnent naissance au célèbre cryptosystème RSA. L'idée est d'utiliser une fonction *à sens unique à trappe*, c'est-à-dire une fonction algorithmiquement facile à calculer mais donc la fonction réciproque est très difficile à calculer sauf lorsque l'on connaît le secret. Pour RSA par exemple, la fonction utilisée est l'exponentiation modulaire qui sera largement étudiée dans ce cours, et la sécurité repose sur le calcul des racines n-èmes modulaires ; la trappe consiste

à connaître la factorisation d'un très grand nombre (au moins 1024 bits) qui est produit de deux nombres premiers.

1.2 Cryptanalyse

Lorsque l'on utilise un cryptosystème, il faut s'assurer qu'il est fiable. Dans le cas de la cryptographie asymétrique par exemple, la connaissance de la clef publique ne doit pas permettre de déduire la clef privée. En ce qui concerne la confidentialité par exemple, on s'attend à ce que le déchiffrement d'un cryptogramme lorsque l'on ne connaît pas la clef secrète soit aussi difficile que de deviner cette clef. La *cryptanalyse* est justement la partie de la cryptologie qui s'occupe d'accéder à la compréhension de messages chiffrés sans avoir la connaissance de l'information secrète, à savoir la clef. On distingue différents buts possibles pour une attaque cryptanalytique suivant l'ambition de l'attaquant :

— trouver la clef de déchiffrement,
— décrypter un cryptogramme,
— extraire des informations partielles d'un cryptogramme.

La difficulté de ces objectifs dépend bien sûr des moyens de l'attaquant. Généralement, quatre niveaux d'attaques peuvent être visés :

1. *attaque à texte chiffré uniquement :* l'attaquant ne dispose que de textes chiffrés ;
2. *attaque à texte clair connu :* l'attaquant connaît des couples clairs/chiffrés ;
3. *attaque à clair choisi :* l'attaquant peut faire chiffrer des messages de son choix ;
4. *attaque à chiffré choisi :* l'attaquant peut faire déchiffrer des messages de son choix.

Afin d'assurer un maximum de sécurité, on demande que l'attaquant, même s'il a eu durant une période d'apprentissage la possibilité de déchiffrer autant de cryptogrammes que souhaité, ne soit pas capable d'extraire une quelconque information d'un cryptogramme donné après sa période d'apprentissage.

La suite du cours consiste à introduire les outils mathématiques nécessaires à la compréhension du cryptosystème RSA. Dès que le bagage mathématique sera suffisant, on traitera le procédé de chiffrement RSA ainsi que d'autres problèmes cryptographiques célèbres tels que le problème d'échange de clef secrète dans le cas de la cryptographie symétrique. On s'intéressera à chaque fois à la sécurité des protocoles présentés et à leur résistance aux attaques précédentes.

2 Arithmétique modulaire

2.1 Division euclidienne et relation de congruence

On rappelle la définition de la division euclidienne dans les entiers (qui est valable sur tout anneau euclidien ; dans le cas des entiers, c'est la valeur absolue qui joue le rôle de stathme, pour les polynômes, c'est le degré).

Définition. *L'ensemble des entiers relatifs $\mathbb{Z}$ vérifie la propriété suivante :*

pour $a, b \in \mathbb{Z}$ deux entiers donnés ($b \neq 0$), il existe un unique couple d'entiers $q \in \mathbb{Z}$ et $r \in \mathbb{N}$ tels que

$$a = bq + r, \quad \text{avec } 0 \leq r < |b|$$

Exemples :

— $a = 7$, $b = 3$: $q = 2$ et $r = 1$
— $a = 7$, $b = -3$: $q = -2$ et $r = 1$
— $a = -7$, $b = 3$: $q = -3$ et $r = 2$
— $a = -7$, $b = -3$: $q = 3$ et $r = 2$

Un peu de vocabulaire :
— r est le *reste* dans la division euclidienne de a par b, et q est le *quotient*.
— lorsque $r = 0$, autrement dit lorsqu'il existe q tel que $a = bq$, on dit que b *divise* a. L'entier b est alors appelé *diviseur* de a et a est un *multiple* de b. On note $b|a$.

Exemples : $2|16$ mais $3 \nmid 16$.

En cryptographie et plus généralement en informatique, tous les calculs se font sur machine où la taille mémoire des entiers est limitée. Par exemple si l'on travaille avec des entiers de 32 bits, il faut que les opérations élémentaires sur les entiers telles que l'addition ou la multiplication donnent des résultats cohérents qui ne dépassent pas 32 bits. L'arithmétique modulaire permet de répondre à ses besoins. On commence par introduire la relation de congruence qui dans notre exemple sur 32 bits permet d'identifier des entiers qui diffèrent d'un multiple de 2^{32} tout en préservant les propriétés usuelles des opérations d'addition, multiplication et inversion.

Définition. *Soient a, b et n trois entiers. On dit que a est* congru *à b* modulo *n et on note $a = b\ [n]$, si a et b ont le même reste dans la division par n.*

Autrement dit

$$\begin{aligned} a = b\ [n] &\Leftrightarrow a - b \text{ est divisible par } n \\ &\Leftrightarrow \exists k \in \mathbb{Z},\ a = b + k.n \end{aligned}$$

Exemples :
— $21 = 1\ [10]$, $-17 = -2\ [3]$, $-17 = 1\ [3]$;
— $10 = 1\ [9]$, $-8 = 1\ [9]$ et plus généralement tous les nombres congrus à 1 modulo 9 sont de la forme $1 + 9n$ $(n \in \mathbb{Z})$;
— un entier a est toujours congru modulo n à son reste r obtenu par division euclidienne de a par n.

La relation de congruence, comme l'égalité classique entre deux entiers, est une *relation d'équivalence*, c'est-à-dire qu'elle est
— *réflexive :* $a = a\ [n]$,
— *symétrique :* $a = b\ [n] \Rightarrow b = a\ [n]$,
— *transitive :* $a = b\ [n]$ et $b = c\ [n] \Rightarrow a = c\ [n]$.

On rappelle la définition de classe d'équivalence dans le cas de la relation d'équivalence donnée par la congruence :

Définition. *Soient a et n deux entiers. La classe d'équivalence de a pour la relation de congruence modulo n est l'ensemble des entiers*

$$\bar{a} := \{b \in \mathbb{Z} : a = b\ [n]\}.$$

Comme dans le cas de l'égalité, l'addition et la multiplication restent compatibles à la relation de congruence :

$$\begin{cases} a = a_1 \ [n] \\ b = b_1 \ [n] \end{cases} \Rightarrow \begin{cases} a + b = a_1 + b_1 \ [n] \\ a \times b = a_1 \times b_1 \ [n] \end{cases}$$

et on hérite également d'autres propriétés intéressantes :

— si $c \in \mathbb{Z}^*$, $a = b \ [n] \Leftrightarrow ac = bc \ [nc]$

— $a = b \ [mn] \Rightarrow (a = b \ [m]$ et $a = b \ [n])$

Application : critères de divisibilité

On rappelle que lorsque l'on écrit un entier $x = (a_n a_{n-1} ... a_1 a_0)_b$ en base b où $0 \geq a_i < b$, cela signifie que $x = a_n.b^n + a_{n-1}.b^{n-1} + ... + a_1.b + a_0 = \sum_{k=0}^{n} a_k.b^k$. Par exemple, si x s'écrit 100011 en base 2, cela signifie que x vaut $2^5 + 2^1 + 2^0 = 35$.

Avec quelques opérations sur les congruences, on déduit facilement des critères de divisibilités classiques :

— *divisibilité par 2 :* $\sum_{k=0}^{n} a_k.10^k = a_0 \ [2]$: un nombre est pair si et seulement si son chiffre des unités est pair.

— *divisibilité par 3 :* $10 = 1 \ [3] \Rightarrow \sum_{k=0}^{n} a_k.10^k = \sum_{k=0}^{n} a_k \ [3]$: un nombre est divisible par 3 si et seulement si la somme de ses chiffres est divisible par 3.

— *divisibilité par 9 :* $10 = 1 \ [9] \Rightarrow$ un nombre est divisible par 9 si et seulement si la somme de ses chiffres est divisible par 9.
Le critère de divisibilité par 9 est à la base de la célèbre preuve par 9 souvent utilisée en école primaire par les élèves pour vérifier leurs calculs. Elle consiste à vérifier un calcul en le refaisant modulo 9. Si les résultats ne sont pas égaux modulo 9 alors le calcul n'est pas correct ; par contre le calcul peut ne pas être correct sans que la preuve par 9 ne le détecte. Par exemple, en trouvant $231 \times 53 = 12243$, l'élève peut facilement vérifier d'une part que $231 \times 53 = 6 \times 8 = 48 = 3 \ [9]$ (en utilisant bien à chaque fois que la congruence s'obtient en faisant la somme des chiffres du nombre modulo 9) et que $12243 = 3 \ [9]$.

Exercice : faire une méthode analogue pour la preuve par 11 en remarquant que $10^k = (-1)^k [11]$. Tester sur l'exemple précédent.

Si l'on revient au problème qui nous préoccupe, à savoir comment faire une arithmétique cohérente sur des entiers qui tiennent en mémoire sur seulement 32 bits, on voit qu'il suffit de travailler avec des relations de congruence modulo 2^{32} et d'*identifier* tout entier à son reste modulo 2^{32} auquel il est congru. On obtient ainsi l'ensemble quotient $\mathbb{Z}/n\mathbb{Z}$:

Définition. *L'ensemble des* entiers modulo *n, noté $\mathbb{Z}/n\mathbb{Z}$, est l'ensemble des classes d'équivalence pour la relation de congruence modulo n.*

En particulier,

— l'addition et la multiplication sont bien définies dans $\mathbb{Z}/n\mathbb{Z}$;

— chaque élément de $\mathbb{Z}/n\mathbb{Z}$ (qui est une classe d'équivalence) a un unique représentant compris entre 0 et $n-1$; pour simplifier les notations, plutôt que de noter $\mathbb{Z}/n\mathbb{Z} = \{\bar{0}; \ldots; \overline{n-1}\}$, on omettra le symbole $\bar{}$ en identifiant une classe à son unique représentant compris entre 0 et $n-1$, et on considérera simplement que

$$\mathbb{Z}/n\mathbb{Z} = \{0; \ldots; n-1\}.$$

Il faut faire attention à cet abus de notation, et ne pas oublier que lorsque l'on somme ou multiplie deux éléments de $\mathbb{Z}/n\mathbb{Z}$, on doit toujours faire la réduction modulo n pour trouver le bon représentant compris entre 0 et $n-1$.

Exemple : l'ensemble $\mathbb{Z}/9\mathbb{Z}$ est muni des lois $+$ et $\times$ dont les tables sont données ci-dessous :

$+$	0	1	2	3	4	5	6	7	8
0	0	1	2	3	4	5	6	7	8
1	1	2	3	4	5	6	7	8	0
2	2	3	4	5	6	7	8	0	1
3	3	4	5	6	7	8	0	1	2
4	4	5	6	7	8	0	1	2	3
5	5	6	7	8	0	1	2	3	4
6	6	7	8	0	1	2	3	4	5
7	7	8	0	1	2	3	4	5	6
8	8	0	1	2	3	4	5	6	7

$\times$	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8
2	0	2	4	6	8	1	3	5	7
3	0	3	6	0	3	6	0	3	6
4	0	4	8	3	7	2	6	1	5
5	0	5	1	6	2	7	3	8	4
6	0	6	3	0	6	3	0	6	3
7	0	7	5	3	1	8	6	4	2
8	0	8	7	6	5	4	3	2	1

2.2 Exponentielle modulaire

La multiplication étant compatible à la relation de congruence, on peut définir facilement l'exponentiation modulaire.

Définition. *Soient $g, n \in \mathbb{N}$. La fonction définie par*

$$f_{g,n} : a \in \mathbb{N} \mapsto f_{g,n}(a) = g^a \ [n]$$

est appelée exponentielle modulaire.

Comme l'exponentielle classique sur les entiers, l'exponentielle modulaire satisfait les propriétés arithmétiques suivantes :

$$(g^x)^y = g^{xy} = (g^y)^x, \qquad g^{x+y} = g^x g^y$$

qui sont utilisées notamment dans le cryptosystème RSA. Comme cette fonction est utilisée de façon intensive dans RSA, on s'intéresse dans la suite de cette section aux calculs rapides de ses valeurs.

L'idée commune à tous les algorithmes d'*exponentiation rapide* se résume de la façon suivante :

- pour faciliter les calculs, toujours considérer les valeurs intermédiaires modulo n ;
- écrire l'exposant x en base 2 : $x = (x_{\ell-1} \dots x_1 x_0)_2 \Leftrightarrow x = \sum_{i=0}^{\ell-1} x_i 2^i$ où $\ell =$ est le nombre de bits de x ;
- calculer de proche en proche les puissances de g en utilisant soit l'élévation au carré soit une multiplication par g.

On considère un exemple d'un tel algorithme appelé *algorithme de droite à gauche* (*right-to-left*), il existe beaucoup d'autres bien plus performants. Pour clarifier les idées, on explique d'abord le principe sur un exemple numérique. Soit à calculer 5^{13} [9] ; on décompose $13 = 2^3 + 2^2 + 2^0$ puis on calcule les carrés successifs $5^{2^0}, 5^{2^1}, \dots, 5^{2^3}$, en réduisant au maximum modulo 9 à chaque étape :

$$5 \ [9]; \quad 5^{2^1} = 25 = 7 \ [9]; \quad 5^{2^2} = 7^2 = 4 \ [9]; \quad 5^{2^3} = (5^{2^2})^2 = 4^2 = 7 \ [9].$$

On multiplie ensuite parmi les nombres obtenus ceux dont l'exposant intervient dans la décomposition de x : $5^{13} = 5^8 \times 5^4 \times 5 = 7 \times 4 \times 5 = 5$ [9].

Algorithme 1 : Algorithme d'exponentiation de droite à gauche

Entrées : g, n et x
Sortie : $g^x\ [n]$
$y \leftarrow 1$, $s \leftarrow g$
tant que $x \neq 0$ **faire**
  si $x = 1\ [2]$ **alors**
    $y \leftarrow y \times s\ [n]$
  $x \leftarrow \lfloor x/2 \rfloor$
  $s \leftarrow s^2\ [n]$
retourner y

Le pseudo-code donné ci-dessous permet de généraliser ce type de calcul pour tous entiers g, n et x.

Exemple : calcul de $g^{283}\ [n]$

y	x	s
1	283	g
g	141	g^2
g^3	70	g^4
g^3	35	g^8
g^{11}	17	g^{16}
g^{27}	8	g^{32}
g^{27}	4	g^{64}
g^{27}	2	g^{128}
g^{27}	1	g^{256}
g^{283}	0	–

Il est pertinent de bien comprendre pourquoi cet algorithme est beaucoup plus rapide que l'algorithme naïf qui consisterait à multiplier g par lui-même x fois modulo n. En effet, par comparaison l'algorithme d'exponentiation rapide n'utilise que $\log_2(x)$ (c'est-à-dire le nombre de bits de x) multiplications et élévations au carré : on dit que cet algorithme est polynomial en la taille des données, contrairement à l'algorithme naïf qui lui est exponentiel en la taille de x.

Application : construction d'une première fonction à sens unique.
On vient de voir qu'il existe des algorithmes très efficaces pour résoudre le problème suivant

$$\text{Étant donnés } n \in \mathbb{Z},\ g \in \mathbb{Z}/n\mathbb{Z} \text{ et } x \in \mathbb{Z}, \text{ trouver } y = g^x\ [n].$$

Par contre (moyennant quelques hypothèses sur g et n), on ne connaît pas à ce jour d'algorithme efficace (c'est-à-dire polynomial) permettant de résoudre le problème inverse appelé *problème du logarithme discret*

$$\text{Étant donnés } n \in \mathbb{Z},\ g \in \mathbb{Z}/n\mathbb{Z} \text{ et } y = g^x\ [n], \text{ trouver } x.$$

La fonction exponentielle est donc typiquement le genre de fonction qui va nous intéresser en cryptographie à clef publique. Dans la suite du cours, on présentera notamment le protocole d'échange de clefs de Diffie et Hellman (1976) et l'algorithme de chiffrement Elgamal qui se fondent sur cette fonction à sens unique.

2.3 Algorithme d'Euclide

Une autre opération sera largement utilisée dans nos protocoles cryptographiques : l'inversion modulaire. Celle-ci est moins évidente à définir que la multiplication ou l'exponentielle modulaire. C'est l'algorithme d'Euclide qui sera l'outil essentiel pour le calcul effectif d'inverses modulaires.

L'algorithme d'Euclide permet en fait de calculer le plus grand diviseur commun de deux nombres (mais aussi de deux polynômes ou plus généralement d'une paire dans un anneau euclidien).

Définition. *Soient a et b deux entiers.*
- *Le* pgcd *de a et b est le plus grand diviseur commun de a et de b ; on le note $pgcd(a,b) = a \wedge b$. On dit que deux nombres sont* premiers entre eux *lorsque leur pgcd est égal à* 1.
- *Le* ppcm *de a et b est le plus petit multiple commun de a et de b ; on le note $ppcm(a,b) = a \vee b$.*

On généralise sans difficulté ces définitions au cas d'ensemble d'entiers : par exemple si d est un diviseur commun tous les éléments de l'ensemble $\{a_1, \ldots, a_n\}$ et que tout autre diviseur commun de cet ensemble est plus petit que d, alors d est le *pgcd* de $\{a_1, \ldots, a_n\}$; on note le pgcd de cet ensemble $\text{pgcd}(a_1, \ldots, a_n)$.

Exemples : $6 \wedge 9 = 3$; $6 \vee 9 = 18$

Le pgcd de nombres peut en fait s'exprimer comme une combinaison linéaire de ces nombres.

Théorème (Théorème de Bézout)**.** *Soient $a, b \in \mathbb{Z}$ deux entiers. Alors*

$$\exists u, v \in \mathbb{Z} \text{ tels que } au + bv = a \wedge b.$$

On dit alors que u et v sont des coefficients de Bézout *de a et b (il n'y a pas unicité de ces coefficients). Plus généralement, si $B = \{b_1, \ldots, b_n\}$ est un ensemble fini d'entiers, alors $pgcd(b_1, \ldots, b_n)$ s'exprime comme une combinaison linéaire $\sum \lambda_k b_k$ des b_k.*

Démonstration. Soient $S = \{\sum_{k=1}^{n} \mu_k b_k\}$ et d un élément non nul de S de valeur absolue minimale. En tant qu'élément de S, d s'exprime comme combinaison linéaire des b_i.
On montre d'abord que $d | b_i$ pour tout i. Soit $b_i = q_i d + r_i$ la division euclidienne d'un des b_i par d. Alors $|r_i| < d$; mais comme $r_i = b_i - q_i d$ est un élément de S dont d est le plus petit élément non nul, la seule possibilité est que $r_i = 0$, i.e. d est un diviseur commun à tous les b_i.
Si maintenant e est un diviseur de tous les b_i, alors il existe q_i' tel que $b_i = q_i' e$ et en particulier $d = \sum \lambda_i b_i = e \sum \lambda_i q_i'$ est un multiple de e. Ainsi d est bien le pgcd de $\{b_1, \ldots, b_n\}$. □

Grâce au théorème de Bézout, on est assuré de l'existence du pgcd. Cependant ce théorème ne permet pas de le calculer effectivement. Le résultat suivant va donner l'idée essentielle utilisée dans l'algorithme d'Euclide pour calculer le pgcd de deux nombres efficacement.

Proposition.

$$pgcd(s,t) = pgcd(s, t - rs), \text{ pour tous éléments } s, t, r.$$

Démonstration. Il est clair que tout diviseur commun à s et t est également diviseur de $t - rs$ et que tout diviseur commun à s et $t - rs$ est un diviseur de $t = (t - rs) + rs$. L'ensemble des diviseurs des deux paires d'entiers étant les mêmes, on déduit facilement le résultat. □

Le principe de l'algorithme est alors facile à comprendre : pour calculer le pgcd de deux entiers, il suffit de remplacer à chaque étape le plus gros des deux entiers par leur différence, jusqu'à ce que les deux nombres soient égaux ; ce nombre est alors forcément égal au pgcd.

Plus précisément, on suppose donnés a et b deux entiers non nuls dont on souhaite calculer le pgcd. On supposera sans perte de généralité que $a \geq b \geq 0$. On commence par définir $r_{-1} = a$ et $r_0 = b$; si r_{i-1} est non nul, on définit récursivement r_i en utilisant la division euclidienne suivante

$$r_{i-2} = q_i r_{i-1} + r_i, \quad 0 \leq r_i < r_{i-1}.$$

Comme la suite (r_i) est une suite positive strictement décroissante, au bout d'un moment on a forcément un reste nul, que l'on note r_{n+1}. D'après la proposition précédente, $\text{pgcd}(a, b) = \text{pgcd}(r_{-1}, r_0) = \cdots = \text{pgcd}(r_n, r_{n+1}) = r_n$.

Algorithme 2 : Algorithme d'Euclide

Entrées : a, b deux entiers positifs
Sortie : $\text{pgcd}(a, b)$
tant que $b > 0$ **faire**
  $tmp \leftarrow b$
  $b \leftarrow a$ [b] (b devient le reste de la division euclidienne de a par b
  $a \leftarrow tmp$ (a devient l'ancien diviseur b)
retourner a

Exemple : calcul de $4864 \wedge 3458$

a	b	q	r
4864	3458	1	1406
3458	1406	2	646
1406	646	2	114
646	114	5	76
114	76	1	38
76	38	2	0

On peut vouloir également retrouver la combinaison linéaire de a et b qui redonne le pgcd en déroulant les équations obtenues dans les différentes divisions euclidiennes. Sur l'exemple, on aurait

$$\begin{aligned}
38 &= 114 - 1 \cdot 76 \\
&= 114 - (646 - 5 \cdot 114) \\
&= 6 \cdot 114 - 1 \cdot 646 \\
&= 6 \cdot (1406 - 2 \cdot 646) - 1 \cdot 646 \\
&= 6 \cdot 1406 - 13 \cdot 646 \\
&= 6 \cdot 1406 - 13 \cdot (3458 - 2 \cdot 1406) \\
&= 32 \cdot 1406 - 13 \cdot 3458 \\
&= 32 \cdot (4864 - 1 \cdot 3458) - 13 \cdot 3458 \\
&= 32 \cdot 4864 - 45 \cdot 3458.
\end{aligned}$$

Évidemment, cette méthode est un peu fastidieuse, d'autant qu'elle nécessite en plus de se souvenir de toutes les divisions euclidiennes effectuées. Dans la version étendue de l'algorithme d'Euclide (essentielle pour le calcul d'inverse modulaire), on va utiliser une relation de récurrence sur les coefficients

de Bézout obtenus entre deux divisions euclidiennes successives afin de trouver directement la combinaison linéaire qui lie a et b. Plus formellement, on introduit les suites (s_i) et (t_i) définies de la façon suivante

— Initialisation : $\begin{cases} s_{-1} = 1 & t_{-1} = 0 \\ s_0 = 0 & t_0 = 1 \end{cases}$

— Relation de récurrence : $\begin{cases} s_{i+1} = s_{i-1} - q_i s_i \\ t_{i+1} = t_{i-1} - q_i t_i. \end{cases}$

On peut montrer par récurrence que pour tout i, on a $s_i a + t_i b = r_i$ et que donc en particulier $s_n a + t_n b = r_n = \text{pgcd}(a, b)$. Le résultat est vrai par définition pour $i = -1, 0$, supposons-le vrai pour $i - 1$ et i (hypothèse de récurrence) et montrons qu'alors il est encore vrai pour $i + 1$: on écrit $r_{i+1} = r_{i-1} - r_i q_i = (s_{i-1}a + t_{i-1}b) - (s_i a + t_i b)q_i = (s_{i-1} - q_i s_i)a + (t_{i-1} - q_i t_i)b = s_{i+1}a + t_{i+1}b$.

2.4 Éléments inversibles de $\mathbb{Z}/n\mathbb{Z}$

Définition. *Soit $a \in \mathbb{Z}$. On dit que a est inversible dans $\mathbb{Z}/n\mathbb{Z}$ s'il existe $b \in \mathbb{Z}/n\mathbb{Z}$ tel que $ab = 1$ $[n]$. b est appelé* **inverse** *de a modulo n, et noté a^{-1}.*
On note $(\mathbb{Z}/n\mathbb{Z})^\times$ l'ensemble des éléments inversibles modulo n. [1]

Le calcul de l'inverse (multiplicatif) modulaire est moins évident que l'addition ou la multiplication. Il n'est d'abord pas clair que tout élément de $\mathbb{Z}/n\mathbb{Z}$ admette un inverse ; par exemple, si l'on écrit la table de multiplication de $\mathbb{Z}/4\mathbb{Z}$, on constate que l'inverse de 2 n'existe pas. Le théorème suivant donne non seulement une caractérisation des éléments inversibles de $\mathbb{Z}/n\mathbb{Z}$, mais sa preuve permet également d'avoir une méthode pratique pour calculer un inverse.

Théorème.

$$x \text{ est inversible dans } \mathbb{Z}/n\mathbb{Z} \text{ si et seulement si } x \wedge n = 1$$

Démonstration.
— Si $x \wedge n = 1$, avec Bézout on a l'existence de u et v tels que $xu + nv = 1$. En prenant l'équation modulo n, on a que u est un inverse de x dans $\mathbb{Z}/n\mathbb{Z}$.
— Réciproquement, si x et n ne sont pas premier entre eux, alors

$$\exists p \geq 2,\ p|x \text{ et } p|n$$

Si par l'absurde, il existait un inverse modulaire y de x, on aurait

$$xy = 1 + kn \Rightarrow p|(xy - kn) = 1$$

contradiction.

En particulier pour trouver l'inverse, on utilise les coefficients de Bézout. □

Exemples :

1. Les inversibles modulo 9 sont les éléments qui sont premiers à 9 : $1, 2, 4, 5, 7$. On retrouve dans la table de multiplication de $\mathbb{Z}/9\mathbb{Z}$ les inverses de ces éléments :

1. On prendra garde à ne pas confondre les notations $(\mathbb{Z}/n\mathbb{Z})^\times$ et $(\mathbb{Z}/n\mathbb{Z})^*$, qui dans le second cas signifie $\mathbb{Z}/n\mathbb{Z} \setminus \{0\}$.

$1^{-1} = 1$ [9] $4^{-1} = 7$ [9] $7^{-1} = 4$ [9]
$2^{-1} = 5$ [9] $5^{-1} = 2$ [9]

Un bon exercice est de les recalculer en utilisant l'algorithme d'Euclide étendu pour obtenir les coefficients de Bézout.

2. Calcul de l'inverse de 80^{-1} [21] : on utilise Euclide étendu de bas en haut

80	21	17	4	1	0
	3	1	4	4	
-19	$+5$	-4	$+1$	0	

De la relation de Bézout, on déduit l'inverse de 80 modulo 21 :

$$80 \times 5 + 21 \times (-19) = 1 \Rightarrow 80^{-1} = 5 \ [21].$$

3. Si p est premier, tous les éléments non nuls de $\mathbb{Z}/p\mathbb{Z}$ sont inversibles. On dit que $\mathbb{Z}/p\mathbb{Z}$ est un *corps*.

Application : le chiffre affine ou substitution mono-alphabétique.
C'est l'exemple le plus simple de chiffrement avec une clef secrète (donc dans le contexte de la cryptographie symétrique). L'idée consiste à d'abord encoder les lettres de l'alphabet par les éléments de $\mathbb{Z}/26\mathbb{Z}$ (le A est encodé 0, le B 1,...) et d'utiliser une fonction de chiffrement affine du type

$$y = (ax + b) \ [26],$$

où x est la lettre encodée à chiffrer et y le chiffré de x.
Pour que la fonction de chiffrement soit bijective (c'est-à-dire que chaque cryptogramme corresponde bien à un unique message), il faut que a soit inversible modulo 26. En résumé, les trois ingrédients de ce chiffrement lettre par lettre sont donc

— *une clef secrète :* $k = (k_1, k_2)$ où $k_1 \in (\mathbb{Z}/26\mathbb{Z})^\times$, $k_2 \in \mathbb{Z}/26\mathbb{Z}$;
— *la fonction de chiffrement :* $c_i = k_1 m_i + k_2$ [26] ;
— *la fonction de déchiffrement :* $m_i = k_1^{-1}(c_i - k_2)$ [26].

Par exemple, si Bob veut envoyer une lettre à Alice qui commence précisément par le prénom de celle-ci, il faut qu'ils conviennent d'abord secrètement d'une clef commune (k_1, k_2). Supposons qu'ils aient choisi $(k_1, k_2) = (3, 11)$. Alors Bob peut commencer à chiffrer sa lettre :

A	L	I	C	E
0	11	8	2	4
11	18	9	17	23
L	*S*	*J*	*R*	*X*

et le message qu'il enverra à Alice commencera donc par *LSJRX*. Si l'on considère le cas particulier où $(k_1, k_2) = (1, 1)$, on retrouve le chiffrement qui était utilisé par César en 50 avant JC. Évidemment, si l'on utilise toujours la même clef, ce chiffrement n'est pas sûr du tout... mais on peut comprendre qu'à l'époque peu de personnes étaient capables de lire, donc ce procédé devait quand même apporter un plus dans la sécurité des communications (et avait le mérite de pouvoir se faire de tête).

Exercice : calculer le nombre de clefs possibles dans le chiffrement affine.

Vu le faible nombre de clefs du chiffrement affine, on pourrait très bien imaginer une attaque par force brute testant toutes les clefs possibles. Il est cependant très facile de faire une attaque plus performante : cette méthode s'appelle la *cryptanalyse fréquentielle*. En effet, comme le chiffre affine préserve

la fréquence d'apparition des lettres (dans l'exemple précédent, la lettre A sera toujours chiffrée par la lettre L), si l'on dispose d'un message chiffré, il devient possible de compter les occurrences de chaque lettre et d'identifier les lettres les plus courantes avec celles de l'alphabet qui reviennent le plus souvent dans un texte français (le E et le A typiquement). Ainsi pour retrouver les deux nombres secrets qui composent la clef, il suffit de deviner comment sont chiffrées au moins 2 lettres.

2.5 Nombres premiers

On verra que dans les protocoles cryptographiques, on doit souvent commencer par choisir des grands nombres premiers et que la sécurité repose souvent sur le fait que ces nombres ne peuvent pas être devinés. Dans cette section, on fait les rappels élémentaires sur les propriétés des nombres premiers.

Définition. *Un nombre $p \in \mathbb{Z}$ est premier s'il est différent de ± 1 et n'est divisible que 1 et par lui-même.*

Exemples :
- 2, 3, 5, 7, 11, 13 sont les plus petits nombres premiers positifs
- autres nombres premiers : 9576890767, 5991810554633396517767024967580894321153, 5885903965180586669073549360644800583458138238012033647539649735017287.
- le plus grand actuellement connu (début 2013) : $2^{57885161}-1$ (s'écrivant avec presque 17 millions de chiffres ! !)

Le théorème fondamental de l'arithmétique est la décomposition en facteurs premiers :

Théorème. *Tout entier $n \in \mathbb{Z}$ ($n \neq 0, 1$) peut se décomposer de façon unique en produit de facteurs premiers :*

$$n = \pm p_1^{\alpha_1} p_2^{\alpha_2} \dots p_n^{\alpha_n}, \quad p_i \text{ premiers et distincts}, \ \alpha_i \in \mathbb{N}^*.$$

Démonstration. Pour simplifier, on suppose $n > 0$; lorsque $n < 0$, il suffit de prendre la décomposition de $-n$.

Montrons par récurrence que tout entier positif n admet une décomposition en facteurs premiers. Si $n = 2$, il n'y a rien à montrer. Supposons le résultat vrai pour tout entier $2 \leq k \leq n-1$, et montrons qu'alors le résultat est encore vrai pour $k = n$. Si n est premier, alors la décomposition est toute trouvée. Sinon, $n = pq$ avec $p, q > 1$, alors on applique l'hypothèse de récurrence à $p < n$ et $q < n$ et on obtient la décomposition en facteurs premiers de n comme produit des décompositions de p et q.

L'unicité se démontre avec le lemme d'Euclide (démontré ci-dessous). □

Lemme (Lemme d'Euclide)**.** *Soient $a, b \in \mathbb{Z}$ et p un nombre premier tel que $p|ab$, alors $p|a$ ou $p|b$.*

Démonstration. Si $p|a$, il n'y a rien à prouver ; autrement, $p \nmid a$ et p et a sont premiers entre eux et donc il existe u, v entiers tels que $ua + vp = 1$. En multipliant cette égalité par b, on voit que nécessairement p qui divise ab doit diviser b. □

La preuve du théorème fondamental permet de trouver la décomposition en facteurs premiers d'un nombre :
- soit il est premier, soit il admet un diviseur premier,
- s'il admet un diviseur premier, on recommence avec le quotient obtenu.

Pour les diviseurs premiers potentiels d'un nombre n, il suffit de ne considérer que les diviseurs inférieurs à $\sqrt{n}$ (car un nombre ne peut pas être le produit de deux nombres supérieurs à $\sqrt{n}$). Ensuite, pour obtenir la liste des nombres premiers inférieurs à $\sqrt{n}$, on utilise le crible d'Erathostène : on liste les nombres inférieurs à $\sqrt{n}$, on retire de la liste le plus petit nombre premier (à savoir 2), et on barre tous ses multiples dans la liste ; on retire alors le plus petit nombre restant qui est forcément premier (c'est 3) et comme précédemment on barre de la liste tous ses multiples ; on recommence ainsi de suite jusqu'à ce qu'il n'y ait plus de nombre dans la liste.

Avec le théorème fondamental de décomposition en facteurs premiers, il est facile de déduire un algorithme naïf de calcul de pgcd et de ppcm de deux nombres :

Propriété. *Si $a = p_1^{\alpha_1} \dots p_n^{\alpha_n}$ et $b = p_1^{\beta_1} \dots p_n^{\beta_n}$ sont les décompositions en facteurs premiers de a et b, alors*

$$\begin{cases} a \wedge b = p_1^{\min(\alpha_1,\beta_1)} \dots p_n^{\min(\alpha_n,\beta_n)}, \\ a \vee b = p_1^{\max(\alpha_1,\beta_1)} \dots p_n^{\max(\alpha_n,\beta_n)}. \end{cases}$$

En particulier,

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

De plus, on a la caractérisation suivante du pgcd de deux entiers a et b :

$$\begin{cases} x|a \\ x|b \end{cases} \Leftrightarrow x|(a \wedge b).$$

Démonstration. On déduit facilement tous ces résultats de la décomposition en facteurs premiers. □

Exemple :

$90 = 2 \times 3^2 \times 5 \times 7^0$ et $105 = 2^0 \times 3 \times 5 \times 7$, donc $\begin{cases} 90 \wedge 105 = 2^0 \times 3 \times 5 \times 7^0 = 15 \\ 90 \vee 105 = 2 \times 3^2 \times 5 \times 7 = 630. \end{cases}$

Malheureusement, comparé à l'algorithme d'Euclide, cet algorithme naïf est beaucoup trop coûteux puisqu'il nécessite de factoriser des nombres. En effet, si les nombres à factoriser sont assez grands, ce calcul peut rapidement devenir infaisable. Par exemple si a et b sont des entiers de 100 chiffres, aucun des algorithmes de factorisation connus à l'heure actuelle ne permettrait de donner les facteurs premiers même en utilisant une machine très puissante pendant plusieurs heures. Alors que l'algorithme d'Euclide permet de faire le calcul du pgcd de a et b en quelques secondes...

On donne une dernière application de l'algorithme d'Euclide étendu : la résolution de systèmes congruentiels à l'aide du fameux théorème des restes chinois.

Théorème (Th. des restes chinois ou CRT)**.** *Soient n, m deux nombres premiers entre eux et a, b deux entiers quelconques. Alors le système suivant*

$$\begin{cases} x = a \ [n] \\ x = b \ [m] \end{cases} \tag{1}$$

admet une unique solution x modulo nm.

Lemme (Lemme de Gauss)**.** *Soient a, b, c des entiers tels que $a|bc$ et $a \wedge b = 1$. Alors $a|c$.*

Démonstration. Ce résultat s'obtient en considérant la décomposition en facteurs premiers de bc. □

Démonstration du théorème.
— $n \wedge m = 1 \Rightarrow \exists u, v \in \mathbb{Z}$ tels que $un + vm = 1$
$\Rightarrow x_0 = bun + avm$ est une solution particulière
— soit x une solution de (1), alors $\begin{cases} x - x_0 = 0\ [n] \\ x - x_0 = 0\ [m] \end{cases} \Rightarrow n|(x - x_0)$ et $m|(x - x_0) \Rightarrow nm|(x - x_0)$
(application directe du Lemme de Gauss).
Réciproquement si $nm|(x - x_0)$, alors x est solution de (1)
— bilan : (1) admet bien x_0 comme unique solution modulo nm

□

La preuve de ce théorème est à connaître, puisqu'elle est effective (c'est-à-dire qu'elle donne une méthode pour trouver les solutions du problème).

Exemple :

Soit à résoudre le système congruentiel suivant : $\begin{cases} x = 2\ [3] \\ x = 3\ [4] \\ x = 1\ [5]. \end{cases}$ On utilise deux fois le théorème des restes chinois :

— $\begin{cases} x = 2\ [3] \\ x = 3\ [4] \end{cases}$: Euclide étendu + CRT $\rightarrow 4 - 3 = 1 \Rightarrow x = 2 \times 4 - 3 \times 3 = -1 = 11\ [12]$

— $\begin{cases} x = 11\ [12] \\ x = 1\ [5] \end{cases}$: Euclide étendu + CRT $\rightarrow 5 \times 5 - 12 \times 2 = 1 \Rightarrow x = 11 \times 5^2 - 2 \times 12 = 251 = 11\ [60]$

On peut réinterpréter le théorème des restes chinois d'un point de vue des ensembles : si m et n sont deux entiers premiers entre eux, alors il y a une bijection entre l'ensemble $\mathbb{Z}/nm\mathbb{Z}$ et l'ensemble $\mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/m\mathbb{Z}$ (c'est-à-dire qu'à chaque élément du premier ensemble correspond un et seul couple dans le deuxième ensemble).

On termine cette section avec quelques résultats sur la répartition des nombres premiers.

Théorème. *Il existe une infinité de nombres premiers.*

Démonstration. Par l'absurde, si $\mathcal{P} = \{p_1, \ldots, p_n\}$ est un ensemble fini de nombres premiers, alors $p = \prod_{k=1}^{n} p_k + 1$ ne peut pas être premier puisqu'il est plus grand que tous les p_k de $\mathcal{P}$, et pourtant il n'est divisible par aucun des nombres premiers p_k de $\mathcal{P}$, ce qui contredit le théorème fondamental de l'arithmétique. □

Ce théorème est déjà rassurant, dans la mesure où l'on peut penser qu'il y a suffisamment de choix possibles. Mais il est également essentiel de s'assurer de la bonne répartition de ces nombres.

Définition. *Soit la fonction de compte des nombres premiers :* $\pi(n) = \#\{p \text{ premier } : p \leq n\}$

Exemples : $\pi(10) = 4$; $\pi(100) = 25$

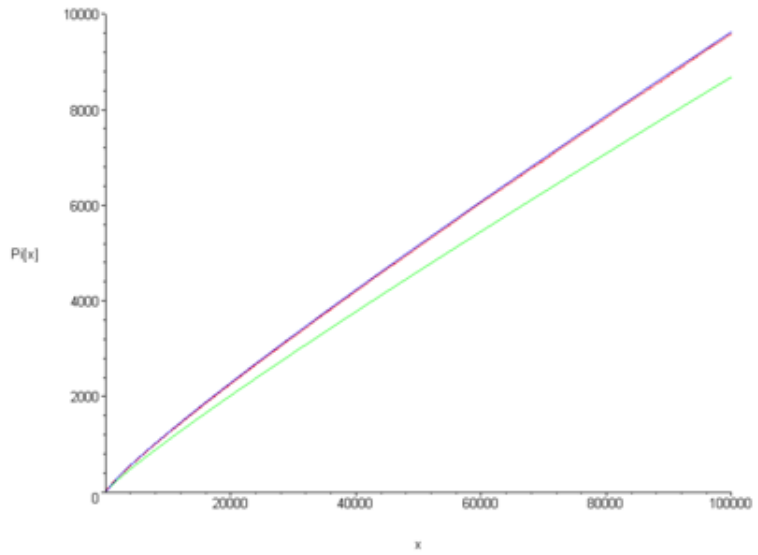


FIGURE 1 – Graphe comparant $\pi(x)$ (en rouge), $\frac{x}{ln(x)}$ (en vert) et $\int \frac{dt}{ln(t)}$ (en bleu)

Théorème. *Pour tout entier x suffisamment grand, on a*

$$\pi(x) \underset{x\to\infty}{\sim} \frac{x}{\ln(x)}.$$

De façon plus précise,

$$\pi(x) \underset{x\to\infty}{\sim} \int_2^x \frac{dt}{\ln(t)}.$$

On peut interpréter ce théorème de la façon suivante : si l'on choisit aléatoirement un nombre près de x, la probabilité que ce nombre soit premier est d'environ $\frac{1}{\ln(x)}$. Par exemple, si $x = 1\,000\,000\,000$, alors en moyenne un nombre sur 21 est premier près de x.

2.6 Théorème d'Euler-Fermat et test de non-primalité

Le théorème d'Euler-Fermat (également appelé "petit théorème de Fermat") est le théorème sur lequel se fonde le cryptosystème RSA. On introduit quelques notions avant de l'énoncer.

Définition (Indicatrice d'Euler)**.** *On appelle* indicatrice d'Euler *de n le nombre d'éléments inversibles de $\mathbb{Z}/n\mathbb{Z}$:*

$$\varphi(n) = \left|(\mathbb{Z}/n\mathbb{Z})^\times\right| = \#\{x : 0 \leq x \leq n \text{ et } x \wedge n = 1\}$$

Exemples :
- $\varphi(9) = 6$; $\varphi(11) = 10$
- plus généralement : p premier $\Rightarrow \varphi(p) = p - 1$

Plus généralement, si l'on connaît la factorisation de n, il est facile de calculer l'indicatrice de n :

Propriété. — *Si $m \wedge n = 1$, alors $\varphi(nm) = \varphi(n)\varphi(m)$ (fonction multiplicative).*

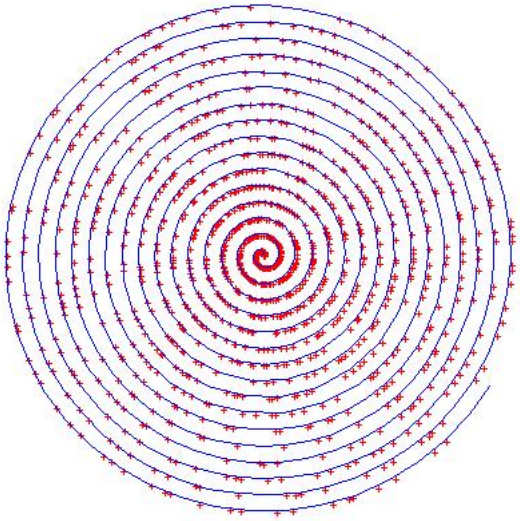


FIGURE 2 – Répartition des nombres premiers inférieurs à 10000 sur la droite réelle

— *Si p est premier,*

$$\varphi(p^n) = p^n - p^{n-1} = p^n\left(1 - \frac{1}{p}\right);$$

en particulier $\varphi(p) = p - 1$.

— *Si $n = p_1^{\alpha_1} \dots p_k^{\alpha_k}$ (DFP), alors*

$$\varphi(n) = n\left(1 - \frac{1}{p_1}\right) \dots \left(1 - \frac{1}{p_k}\right).$$

Démonstration.
— Si $m \wedge n = 1$, alors le théorème des restes chinois assure que $\mathbb{Z}/nm\mathbb{Z} \simeq \mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/m\mathbb{Z}$. On peut même être plus précis et montrer qu'il existe une bijection entre $(\mathbb{Z}/nm\mathbb{Z})^\times$ et $(\mathbb{Z}/n\mathbb{Z})^\times \times (\mathbb{Z}/m\mathbb{Z})^\times$: en effet si $m \wedge n = 1$ alors un entier x est premier à mn si et seulement si il est premier à n et à m. En considérant les cardinalités des deux ensembles, on obtient alors le résultat.
— Soit $1 \leq x \leq p^n$. Alors x est premier à p^n si et seulement si il est premier à p. En particulier, x n'est pas premier à p^n si et seulement si x est un multiple de p ; il y a exactement p^{n-1} multiples de p dans cet intervalle, d'où le résultat.
— Le dernier point se déduit aisément des deux précédents en utilisant une récurrence sur k.

□

Exemple : $\varphi(36) = \varphi(2^2 3^2) = 36\left(1 - \frac{1}{3}\right)\left(1 - \frac{1}{2}\right) = 36 \times \frac{2}{3} \times \frac{1}{2} = 12.$

Théorème (Théorème d'Euler-Fermat)**.** *Soient a, n ($n \geq 0$) deux entiers* **premiers entre eux**, *alors*

$$a^{\varphi(n)} = 1 \ [n]$$

Démonstration. On donne ici une démonstration ensembliste du théorème assez astucieuse.
On rappelle que la condition $a \wedge n = 1$ est équivalente à $a \in (\mathbb{Z}/n\mathbb{Z})^\times$. L'application $f : (\mathbb{Z}/n\mathbb{Z})^\times \to (\mathbb{Z}/n\mathbb{Z})^\times, x \mapsto ax$ est donc

— bien définie : si x inversible, ax l'est aussi : $(ax)^{-1} = x^{-1}a^{-1}$
— bijective : $ax = ay \Rightarrow x = y$ donc injective (a étant inversible) et comme $(\mathbb{Z}/n\mathbb{Z})^\times$ est fini, aussi surjective.

Si on note $a_1, \ldots, a_{\varphi(n)}$ les éléments inversibles de $\mathbb{Z}/n\mathbb{Z}$, on a alors $\{a_1, \ldots, a_{\varphi(n)}\} = \{f(a_1), \ldots, f(a_{\varphi(n)})\}$ (égalité ensembliste), et donc le produit des éléments du premier ensemble est égal au produit des éléments du deuxième : $a_1 \times \ldots \times a_{\varphi(n)} = f(a_1) \times \ldots \times f(a_{\varphi(n)}) = a^{\varphi(n)} \times a_1 \times \ldots \times a_{\varphi(n)}$ $[n]$. En simplifiant de part et d'autre de l'égalité par l'élément inversible $a_1 \times \ldots \times a_{\varphi(n)}$ modulo n, on obtient $a^{\varphi(n)} = 1$ $[n]$. □

Ce théorème signifie entre autres que si $a \wedge n = 1$ et si l'on connaît $\varphi(n)$, alors on peut simplifier les exposants modulaires $a^x = a^{x_0}$ $[n]$, en prenant pour x_0 le reste dans la division euclidienne de x par $\varphi(n)$.

Il est à noter cependant qu'en pratique, le calcul de $\varphi(n)$ est difficile. Par exemple, si n est le produit de deux nombres premiers distincts p et q, le calcul de $\varphi(n)$ est aussi difficile que la factorisation de n, qui comme on l'a déjà évoqué, est un problème difficile. On renvoie aux exercices de TD pour la preuve de ce résultat.

Lorsque p est premier, le théorème d'Euler-Fermat permet de déduire un test de *pseudo-primalité*. Son énoncé devient :

Théorème. — *Si p premier et $a \in \mathbb{Z}/p\mathbb{Z}$ tel que $p \nmid a$ (autrement dit $a \wedge p = 1$), alors*

$$a^{p-1} = 1 \ [p]$$

— *Si p premier et $a \in \mathbb{Z}$, alors $a^p = a$ $[p]$*

Démonstration. Le premier point est l'application directe d'Euler-Fermat pour $n = p$. Seul le deuxième point est à vérifier. Considérons le cas où a n'est pas premier à p, i.e. est un multiple de p : l'égalité à démontrer se réécrit alors $0 = 0$ $[p]$... □

En considérant la réciproque du théorème, on obtient le test de primalité probabiliste suivant :

1. choisir a aléatoire tel que $1 \leq a \leq n$
2. n passe le test si $a^{n-1} = 1$ $[n]$

Si n passe le test, n a une bonne probabilité d'être premier, par contre si n ne passe pas le test, on est certain que n n'est pas premier :

$$2^{25009996} = 13697276 \ [25009997] \Rightarrow 25009997 \text{ n'est pas premier.}$$

3 Factorisation et RSA

3.1 RSA en confidentialité

Grâce à ces pré-requis mathématiques, on peut maintenant donner tous les ingrédients de RSA en chiffrement :

— on se place dans l'ensemble $(\mathbb{Z}/n\mathbb{Z}, \times)$ avec $n = pq$ produit de deux grands nombres premiers ;

— la fonction de chiffrement est une simple exponentielle modulo n dont l'exposant e est appelé *exposant de chiffrement* **public** :

$$x \mapsto x^e \ [n].$$

Cet exposant e doit satisfaire $e \wedge \varphi(n) = 1$;

— la fonction de déchiffrement est également une exponentielle modulo n, c'est en fait l'application réciproque de la fonction de chiffrement :

$$x \mapsto x^d \ [n].$$

L'*exposant de déchiffrement* d qui est **privé**, satisfait donc $ed = 1 \ [\varphi(n)]$, i.e. c'est l'inverse de e modulo $\varphi(n)$;

— on peut vérifier sans difficulté grâce à Euler-Fermat que ce déchiffrement est bien l'application réciproque de la fonction de chiffrement :

1. on se place d'abord sur $(\mathbb{Z}/n\mathbb{Z})^\times$. Soit $x \wedge n = 1$, alors $x^{\varphi(n)} = 1 \ [n]$. Alors

$$x^{ed} = x^{1+k\varphi(n)} = x.(x^{\varphi(n)})^k = x \ [n].$$

2. si $x \notin (\mathbb{Z}/n\mathbb{Z})^\times$, alors on peut supposer par exemple que $p|x$; dans ce cas, $x^{ed} = x = 0 \ [p]$. Par ailleurs, soit $x \wedge q = 1$ et $x^{ed} = x \cdot x^{k\varphi(n)} = x \cdot x^{k\varphi(p)\varphi(q)} = x \cdot (x^{q-1})^{k\varphi(p)} = x \cdot 1 = x \ [q]$; soit $x = 0 \ [q]$ et $x^{ed} = x = 0 \ [q]$. On conclut alors directement avec les restes chinois.

On dit que la fonction de chiffrement $x \mapsto x^e \ [n]$ est une *fonction à sens unique à trappe*, puisqu'il suffit de connaître d pour être capable de calculer facilement la fonction réciproque.

Le déroulement du protocole RSA en chiffrement est alors le suivant. Supposons qu'Alice souhaite envoyer un message chiffré à Bob.

1. Etape 1 : Bob fixe les paramètres du cryptosystème

   (a) Bob génère p,q premiers et distincts de 512 bits qu'il garde secrets.
   Il calcule $n = pq$ et l'envoie à Alice via un canal non sécurisé, et calcule $\varphi(n) = (p-1)(q-1)$ qu'il garde secret.

   (b) Bob choisit un exposant e premier avec $\varphi(n)$ (souvent $e = 3$ ou $e = 2^{16} + 1$ dans les implémentations industrielles) qu'il envoie à Alice via un canal non sécurisé.
   Il calcule $d = e^{-1} \ [\varphi(n)]$ qu'il garde secret.

2. Etape 2 : Alice envoie un message chiffré à Bob
   Pour envoyer son message $m \in \mathbb{Z}/n\mathbb{Z}$ chiffré à Bob, Alice calcule $c = m^e \ [n]$ avec la clef publique de Bob, et l'envoie à Bob.

3. Etape 3 : Bob décrypte
   Bob décrypte c en calculant $c^d = (m^e)^d = m \ [n]$ avec sa clef secrète d.

Exemple :

1. Bob génère $p = 47$ et $q = 59$, puis calcule $n = pq = 2773$ et $\varphi(n) = (p-1)(q-1) = 2668$. Il choisit $e = 17$ et en déduit $d = e^{-1} = 157 \ [2668]$ avec l'algorithme d'Euclide étendu.

2. Alice veut coder `ITS ALL GREEK TO ME`, soit
   $m = 0920\,1900\,0112\,1200\,0718\,0505\,1100\,2015\,0013\,0500$ (où l'espace est codé 00, `A`= 01, ...)
   Le 1er bloc $m_1 = 0920$ donne $c_1 = 920^{17} = 948 \ [2773]$, etc ...

3. Bob reçoit $c = 0948\,2342\,1084\,1444\,2663\,2390\,0774\,0919\,1655$.
   Pour déchiffrer c_1, il calcule $948^{157} = 920$ [2773] qui correspond bien à `IT`, etc...

Remarques :

1. Le calcul de l'exposant de déchiffrement est facile puisqu'il s'obtient simplement en utilisant l'algorithme d'Euclide étendu.
2. Le chiffrement et déchiffrement peuvent être rapidement calculés en utilisant un algorithme d'exponentiation rapide.
3. Le modulo RSA s'obtient rapidement. En effet, on rappelle que l'obtention de ce modulo nécessite de choisir de grands nombres premiers aléatoires. Pour choisir un grand nombre premier, on choisit un nombre aléatoire dans un intervalle fixe et on applique des tests de primalité pour savoir si ce nombre est premier. On recommence tant que le nombre n'est pas premier (et on est assuré avec le théorème de densité des nombres premiers que cette méthode finira par aboutir).

3.2 Analyse de la sécurité

Il est clair que la sécurité de RSA repose complètement sur la capacité qu'a l'attaquant de calculer des racines e-èmes de l'unité. Pour l'instant, la seule approche que l'on connaît pour faire ce calcul (en ne faisant aucune hypothèse supplémentaire) est de calculer la factorisation de n pour retrouver $\varphi(n)$ et en déduire d comme le ferait le détenteur de la clef secrète lui-même. C'est pour cette raison que l'on dit souvent que la sécurité de RSA repose sur la difficulté de la factorisation. Pour illustrer la difficulté de factoriser, on peut citer Martin Gardner lors de la parution de l'article RSA (*"A new kind of cipher that would take millions of years to break"*, Sci. Am., 1977) expliquant que factoriser un entier de 420 bits prendrait environ 40×10^{15} années. Bien sûr, les algorithmes de factorisations se sont perfectionnés (mais la complexité de tels algorithmes reste sous-exponentielle....) et le record actuel (2009) est la factorisation d'un nombre semi-premier (i.e. produit de deux nombres premiers) de plus de 770 bits moyennant environ 2000 ans de calcul sur une seule machine (ces calculs sont bien entendus partiellement parallélisables).

Néanmoins lorsque l'attaquant dispose d'une aide extérieure (comme c'est le cas par exemple dans une attaque à texte chiffré choisi), il est possible de faire de nombreuses attaques sur RSA. On renvoie aux exercices de TD pour plus de détails.

4 Résiduosité quadratique

Soit $n \in \mathbb{N}, n > 2$. On s'intéresse à la fonction élévation au carré modulaire $x \mapsto x^2 \bmod n$; contrairement à la fonction de chiffrement dans le cryptosystème RSA, cette fonction est clairement non injective ($(-1)^2 = 1 \bmod n$) mais son utilisation reste pertinente en cryptographie. On peut en effet établir des protocoles basés sur les deux problèmes suivants :

Problème de résiduosité quadratique (version décisionnelle)
Étant donnés c, n, est-ce que l'entier c est un carré modulo n ?

Problème de calcul des racines carrés (version calculatoire)
Étant donnés c, n, trouver x s'il existe tel que $x^2 = c \bmod n$.

La difficulté de ces problèmes est variable suivant que n est premier ou composé. Plus précisément, lorsque n est premier il est facile de résoudre ces deux problèmes, par contre lorsque n est composé c'est beaucoup plus difficile.

4.1 Carrés modulo p

On considère dans la suite p premier avec $p > 2$.

Propriété. *Le nombre de carrés (ou résidus quadratiques) modulo p est égal à* $(p+1)/2$ *; en particulier, le nombre de résidus quadratiques non nul est égal à* $(p-1)/2$.

Démonstration. Soit $\psi : \mathbb{Z}/p\mathbb{Z}^* \to \mathbb{Z}/p\mathbb{Z}^*, x \mapsto x^2 \bmod p$ l'application d'élévation au carré. Tout élément de l'image a exactement deux antécédents, en effet :

$$x^2 = a^2 \bmod p \Leftrightarrow (x-a)(x+a) = 0 \bmod p,$$

en particulier si $x \neq a$ alors $(x-a)$ est inversible modulo p et $x + a = 0 \bmod p$, i.e. $x = -a \bmod p$. Par conséquent $\#\mathrm{Im}\psi = (p-1)/2$. □

Ce résultat peut se généraliser de la façon suivante :

Lemme. *Soit* $P \in \mathbb{Z}/p\mathbb{Z}[X]$ *un polynôme de degré d. Alors P a au plus d racines dans* $\mathbb{Z}/p\mathbb{Z}$.

Démonstration. On fait une récurrence sur le degré d de P :

- Pour $d = 1$: il est facile de voir P admet exactement une solution.
- supposons la propriété vraie pour tous les polynômes de degré d donné (hypothèse de récurrence), et montrons qu'alors elle est encore vraie pour tous les polynômes de degré $d+1$. Supposons que P de degré $d+1$ admette une racine a modulo p (si une telle racine n'existe pas alors la démonstration est terminée). On effectue la division euclidienne de P par $X - a$: $P(X) = (X-a)Q(X)$, puis on considère une autre racine $b \neq a$ de P. Alors $P(b) = (b-a)Q(b) = 0 \bmod p$ donc $Q(b) = 0$ puisque $b - a \in \mathbb{Z}/p\mathbb{Z}^\times$. Comme Q a au plus d racines qui sont toutes des racines de P, le polynôme P a lui-même au plus $d+1$ racines.

□

Remarque : ceci est faux si p n'est pas premier, par exemple le polynôme $X^2 - 1$ admet quatre racines dans $\mathbb{Z}/15\mathbb{Z}$, à savoir ± 1 et ± 4.

Définition. *Soit* $p > 2$ *un nombre premier et a un entier, alors le symbole de Legendre* $\left(\frac{a}{p}\right)$ *vaut*

$$\begin{cases} 0 \text{ si } p|a \\ 1 \text{ si } a \text{ résidu quadratique inversible mod } p \\ -1 \text{ sinon} \end{cases}$$

On peut facilement résoudre le problème de résiduosité quadratique décisionnel dans le cas premier avec le résultat suivant :

Théorème (Théorème d'Euler)**.**

$$\left(\frac{a}{p}\right) = a^{\frac{p-1}{2}} \bmod p$$

Démonstration. — Si $a = 0 \bmod p$, alors $a^{\frac{p-1}{2}} = 0 \bmod p$.

— On suppose $a \neq 0$. D'après Euler-Fermat, on a pour tout $a \neq 0$, $(a^{\frac{p-1}{2}})^2 = a^{p-1} = 1 \bmod p$. En particulier, $a^{\frac{p-1}{2}} = \pm 1 \bmod p$ d'après le lemme précédent. Maintenant si $a = b^2 \bmod p$ (i.e. a est un carré modulo p), alors $a^{\frac{p-1}{2}} = b^{p-1} = 1 \bmod p$. Or il y a $(p-1)/2$ carrés non nuls modulo p qui sont donc exactement toutes les racines de $X^{(p-1)/2} - 1$ (cf lemme précédent). On a bien que $a^{\frac{p-1}{2}} = 1 \bmod p$ si et seulement si a est un carré non nul modulo p.

□

Ce théorème permet donc de répondre affirmativement au problème de résiduosité quadratique décisionnel modulo un nombre premier p. Il permet également de déduire de nombreuses propriétés sur le symbole de Legendre.

Corollaire.

$$\left(\frac{a}{p}\right)\left(\frac{b}{p}\right) = \left(\frac{ab}{p}\right).$$

Corollaire. (-1) *est un carré modulo* p *si et seulement si* $p = 1$ *mod* 4.

Démonstration. Si $p = 1 \bmod 4$ alors $p = 1 + 4k$, donc $(-1)^{(p-1)/2} = (-1)^{2k} = 1 \bmod p$. Pour la réciproque, on raisonne par contraposée : si $p = 3 \bmod 4$, alors $p = 3 + 4k$, donc $(-1)^{(p-1)/2} = (-1)^{2k+1} = -1 \bmod p$. □

Pour la version calculatoire du problème modulo un nombre premier, il est également facile de faire le calcul de racines carrées lorsque $p = 3 \bmod 4$. En effet soit $a \in \mathbb{Z}$ tel que $\left(\frac{a}{p}\right) = 1$, alors $\left(a^{\frac{p+1}{4}}\right)^2 = a^{\frac{p+1}{2}} = a^{\frac{p-1}{2}} a = 1.a \bmod p$ donc $a^{\frac{p+1}{4}}$ est une racine carré de $a \bmod p$.

Lorsque $p = 1 \bmod 4$, le calcul est possible mais plus compliqué car il nécessite l'utilisation d'un algorithme probabiliste qui construit d'abord un élément non résidu quadratique modulo p (ceci ne pose pas de problème en soi puisqu'un élément sur deux vérifie cette propriété dans $\mathbb{Z}/p\mathbb{Z}^*$ et qu'il est facile de détecter s'il est non quadratique). Pour plus de détails, se référer à l'algorithme de Tonelli-Shanks.

4.2 Racines carrées modulaires et factorisation

On a vu dans la section précédente que la sécurité de RSA repose sur la difficulté de savoir calculer des racines e-èmes modulaires où e est la valeur de l'exposant public de chiffrement. On sait également que si l'on connaît un algorithme qui permet de factoriser n, alors ce calcul devient facile (il suffit de calculer la clef de déchiffrement d et d'élever à la puissance d modulo n le nombre dont on souhaite calculer la racine e-ème modulaire).

Si l'on s'autorise [2] à considérer la valeur $e = 2$, on peut montrer qu'en fait la difficulté du calcul des racines carrées est équivalente à celle du calcul de la factorisation de $n = pq$.

2. ce qui n'est pas possible dans RSA à cause des problèmes d'injectivité que cela génère dans la fonction de chiffrement... néanmoins, on verra que l'on peut corriger ce défaut lorsque l'on présentera le système de chiffrement de Rabin.

— Comme le contexte est différent de celui de RSA, il faut déjà se convaincre que si la décomposition de n est connue, i.e. p, q connus, alors on sait calculer les racines carrées modulo n : d'après le théorème des restes chinois,

$$x^2 = a \bmod n \Leftrightarrow \begin{cases} x^2 = a \bmod p \\ x^2 = a \bmod q \end{cases}$$

donc comme l'on sait calculer les racines carrés modulo p et q, on sait les calculer modulo n. Plus précisément, comme il y a au plus deux racines carrées modulo p et au plus deux racines carrées modulo q, on attend au plus quatre racines carrées modulo n.
Exemple : $p = 3$, $q = 11$, $n = 33$, $a = 31$.
$\left(\frac{31}{3}\right) = \left(\frac{1}{3}\right) = 1 \bmod 3$ donc a est un carré mod p
$\left(\frac{31}{11}\right) = \left(\frac{9}{11}\right) = \left(\frac{3^2}{11}\right) = 1 \bmod 11$ donc a est un carré mod 11
On voit que $\begin{cases} x = 1 \bmod 3 \\ x = 3 \bmod 11 \end{cases} \Leftrightarrow x = 25 \bmod 33$ est une [3] racine carrée de a modulo 33.

— Réciproquement, étant donné un algorithme $\mathcal{A}$ qui renvoie une racine d'un résidu quadratique a modulo n donné en entrée, on peut déduire facilement un algorithme probabiliste qui permet de factoriser n (en temps polynomial) : on choisit a aléatoirement [4] dans $(\mathbb{Z}/n\mathbb{Z})^\times$ puis on considère $b = \mathcal{A}(a^2)$ la sortie de l'algorithme ; on a alors $a^2 = b^2 \bmod n$ et $(a-b)(a+b) = 0 \bmod n$. Deux cas sont possibles
  — soit $a \neq \pm b$ et dans ce cas $(a - b) \wedge n$ est un facteur non trivial de n
  — soit $a = \pm b$ (ce qui arrive avec une probabilité 1/2), alors on recommence avec un autre choix de a.

On déduit de l'analyse faite précédemment la propriété suivante qui résulte du théorème des restes chinois :

Propriété. *Soit $n = pq$ un produit de deux nombres premiers p et q, alors $\#\{c^2 : c \in (\mathbb{Z}/n\mathbb{Z})^\times\} = \varphi(n)/4 = \frac{p-1}{2}\frac{q-1}{2}$.*

4.3 Applications à la cryptographie

4.3.1 Cryptosystème de Rabin

C'est un schéma de chiffrement à clef publique, très similaire à RSA sauf que celui-ci a une sécurité équivalente à la difficulté de factoriser.

— clef publique : $n = pq$
— clef privée : $p = 3 \bmod 4, q = 3 \bmod 4$ grands nombres premiers
— chiffrement : un message clair $m \in \mathbb{Z}/n\mathbb{Z}$ est chiffré en $m^2 \bmod n$
— déchiffrement : on calcule les 4 racines carrées modulo n d'un message chiffré $c \in \mathbb{Z}/nZ$ en utilisant la connaissance de p et q et en appliquant les restes chinois.

Le seul désavantage de ce schéma est qu'il faut distinguer parmi les 4 antécédents de c celui qui correspond au message clair. S'il s'agit d'un texte intelligible, ceci ne pose pas de difficulté, sinon il faut introduire de la redondance ou un padding pour pouvoir distinguer la racine qui convient.

3. Il y en a en fait 3 autres, à savoir 14, 17 et 8.
4. Bien entendu, si l'on tire un entier a qui n'est pas premier avec n - ce que l'on détecte facilement avec le calcul du pgcd de a et n - alors on a trouvé la factorisation de n.

4.3.2 Authentification Zero-Knowledge de Fiat-Shamir

Comment Alice peut-elle prouver à Bob qu'elle connaît la factorisation de $n = pq$ sans révéler p et q ?

Si l'on sait faire une telle preuve de connaissance sans divulgation ("Zero-knowledge"), on imagine très bien comment en déduire un protocole d'autentification : Bob est face à un interlocuteur qui prétend être Alice, il connaît la clef publique n d'Alice, il lui suffit donc de s'assurer que l'interlocuteur en connaît la factorisation.

Voici comment Alice et Bob peuvent procéder. Bob va proposer une suite de challenges à Alice :

1. Bob choisit $y \in \mathbb{Z}/n\mathbb{Z}^\times$ et envoie le challenge $c = y^2 \bmod n$ à Alice
2. Alice renvoie deux carrés (c_1, c_2) tels que $c_2 = cc_1$, générés aléatoirement [5], à Bob.
3. Bob choisit alors c_i un des deux carrés envoyés par Alice et lui demande d'en calculer une racine carrée modulo n.
4. Alice renvoie la racine r demandée (ce qu'elle peut faire puisqu'elle connaît la factorisation de n).
5. Bob teste si r^2 est bien égal au carré demandé modulo n.

Bob n'apprend rien si ce n'est la valeur d'une racine carrée d'un nombre aléatoire modulo n, ce qu'il aurait pu aussi apprendre tout seul en élevant au carré des nombres aléatoires.

L'espion Charlie ne peut pas se faire passer pour Alice, car il ne sait jamais répondre *simultanément* aux deux questions de Bob (sinon il saurait calculer la racine de $c = \frac{c_2}{c_1}$). Par contre il peut fournir (c_1, c_2) permettant de répondre *alternativement* à l'une des questions possées par Bob ; par exemple, en choisissant un couple de la forme (a^2, ca^2) où a est aléatoire, il sait répondre à la première question, et en choisissant un couple de la forme $(a^2/c, a^2)$ avec a aléatoire, il sait répondre à la deuxième question.

4.4 Symbole de Jacobi

Contrairement à la version calculatoire du problème de résiduosité quadratique, le problème décisionnel admet une réponse partielle lorsque n n'est pas premier. Plus précisément, si n est un entier impair sans facteurs carrés, pour une moitié des éléments de $\mathbb{Z}/n\mathbb{Z}$, il est possible de dire que ce ne sont pas des carrés.

Définition. *Soit $n \in \mathbb{N}^*$ un entier impair et soit $n = p_1^{\alpha_1} \dots p_k^{\alpha_k}$ sa décomposition en facteurs premiers. Pour tout entier $a \in \mathbb{Z}$, on définit le symbole de Jacobi de a modulo n par*

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \dots \left(\frac{a}{p_k}\right)^{\alpha_k}.$$

Exemples

1. $\left(\frac{x^2}{n}\right) = 1$
2. $n = pq$ produit de deux premiers, $a \wedge n = 1$, on distingue quatre cas
   (a) $\left(\frac{a}{p}\right) = 1$ et $\left(\frac{a}{q}\right) = 1$, alors $\left(\frac{a}{n}\right) = 1$ et a est un carré modulo n
   (b) $\left(\frac{a}{p}\right) = -1$ et $\left(\frac{a}{q}\right) = -1$, alors $\left(\frac{a}{n}\right) = 1$ mais a n'est pas un carré modulo n
   (c) $\left(\frac{a}{p}\right) = 1$ et $\left(\frac{a}{q}\right) = -1$, alors $\left(\frac{a}{n}\right) = -1$ et a n'est pas un carré modulo n

5. Par exemple, elle tire un nombre aléatoire $a \in \mathbb{Z}/n\mathbb{Z}^\times$ et prend $c_1 = a^2$.

(d) $\left(\frac{a}{p}\right) = -1$ et $\left(\frac{a}{q}\right) = 1$, alors $\left(\frac{a}{n}\right) = -1$ et a n'est pas un carré modulo n

On détecte donc facilement les non-carrés pour une moitié des éléments de $\mathbb{Z}/n\mathbb{Z}$.

On peut calculer de façon efficace le symbole de Jacobi sans connaître la factorisation de n :

Propriétés. *Soient $n \in \mathbb{N}^*$ et $a; v$ deux entiers.*

1. $\left(\frac{a}{n}\right) \in \{0; \pm 1\}$
2. $\left(\frac{a}{n}\right) = 0$ *ssi* $a \wedge n \neq 1$
3. $\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right)$
4. *Si* $a = b \ mod\ n$, *alors* $\left(\frac{a}{n}\right) = \left(\frac{b}{n}\right)$
5. *Lois complémentaires à la loi de réciprocité quadratique*
   (a) $\left(\frac{-1}{n}\right) = (-1)^{\frac{n-1}{2}}$
   (b) $\left(\frac{2}{n}\right) = (-1)^{\frac{n^2-1}{8}} = \begin{cases} 1 \text{ si } n = 1, 7 \ mod\ 8 \\ -1 \text{ si } n = 3, 5 \ mod\ 8 \end{cases}$
6. *Réciprocité quadratique : Soient $mn \in \mathbb{N}^*$ des entiers impairs tels que $m \wedge n = 1$, alors*

$$\left(\frac{m}{n}\right)\left(\frac{n}{m}\right) = (-1)^{\frac{(n-1)(m-1)}{4}}.$$

Exemples

$$\begin{aligned} \left(\frac{31}{91}\right) &= \left(\frac{91}{31}\right)(-1)^{45\times 15} \\ &= -\left(\frac{29}{31}\right) \\ &= \left(\frac{31}{29}\right)(-1)^{15\times 14} \\ &= -\left(\frac{2}{29}\right) \\ &= 1 \end{aligned}$$

Application 1 : test de primalité de Solovay-Strassen
Pour tester si un entier n est premier, on tire un nombre a aléatoire dans $\mathbb{Z}/n\mathbb{Z}$ et on teste l'égalité

$$\left(\frac{a}{n}\right) \stackrel{?}{=} a^{\frac{n-1}{2}} \mod n.$$

Comme pour Fermat, ce test permet de dire en cas de non égalité que n n'est pas premier, mais ne permet pas de conclure en cas d'égalité. Cependant comme il peut exister des entiers a pour lesquels $a^{n-1} = 1$ et $\left(\frac{a}{n}\right) \neq \pm 1$, le test de Solovay-Strassen est bien un raffinement du test de Fermat.

Le calcul d'un tel test est rapide puisque le membre de gauche s'obtient à partir d'un algorithme de type Euclide qui calcule les divisions successives avec tests de parité, et le terme de droite avec une exponentiation rapide.

Exemple : Soit $n = 561$. Avec un algorithme d'exponentiation rapide, on calcule $13^{280} = 13^{2^3} \times 13^{2^4} \times 13^{2^8} = 256 \times 460 \times 460 = 1 \bmod 561$, qui est bien différent de la valeur du symbole de Jacobi correspondant :

$$\begin{aligned}\left(\frac{13}{561}\right) &= \left(\frac{561}{13}\right)(-1)^{6\times 280} \\ &= \left(\frac{2}{13}\right) \\ &= -1\end{aligned}$$

Le test de Solovay-Strassen permet donc de détecter que 561 n'est pas premier. Par contre, on peut voir que le test de Fermat échoue sur cet entier pour tout choix de a avec $2 \leq a \leq 560$ et $a \wedge 561 = 1$ (on dit que ce nombre est *un nombre de Carmichael*).

En pratique, si n n'est pas premier, le test de Solovay-Strassen ne passe pas pour au moins la moitié des éléments a compris entre 2 et $n-1$, donc on a plus d'une chance sur deux de détecter que n n'est pas premier.

Application 2 : Blum-Blum-Shub

Il est possible à partir du problème de résiduosité quadratique, de fabriquer une suite de bits pseudo-aléatoire satisfaisant le test du bit suivant, autrement dit une suite pour laquelle il n'est pas possible de déduire en temps polynomial avec un avantage le bit i en fonction des $i-1$ précédents. On convient qu'une telle suite admet de bonnes propriétés statistiques puisqu'il n'est pas possible de la distinguer en temps polynomial d'une suite aléatoire uniforme sur $\{0;1\}$ (test de Yao).

La sortie de l'algorithme de Blum-Blum-Shub s'obtient en considérant le bit le moins significatif des termes de la suite (x_k) vérifiant

$$x_{k+1}^2 = x_k \bmod pq, \quad x_{k+1} \text{ résidu quadratique}$$

où p, q sont deux grands nombres premiers tels que $p = q = 3 \bmod 4$. L'hypothèse sur p et q permet de s'assurer que quelque soit $x \in (\mathbb{Z}/pq\mathbb{Z})^\times$, la suite définie à partir de $x_0 = x^2$ est bien définie. En effet, si l'on note C l'ensemble des résidus quadratiques modulo pq, alors l'application $x \in C \mapsto x^2$ est injective ($x^4 = y^4 \bmod pq \Rightarrow x^2 = \pm y^2 \bmod p, q \Rightarrow x^2 = y^2$ puisque (-1) n'est un carré ni modulo p, ni modulo q) donc bijective. Il suffit donc de choisir à chaque étape l'unique racine carrée qui est elle-même un carré (une seule parmi les quatre doit satisfaire cette condition).

On peut montrer facilement que si l'on peut deviner le bit $k+1$ de cette suite à partir des k précédents avec un avantage, alors on est capable de résoudre le problème de résiduosité quadratique décisionnel avec le même avantage. En effet, soit a un entier dont on souhaite déterminer si c'est un résidu quadratique ou pas modulo n. Sans perte de généralité, on peut supposer que $\left(\frac{a}{n}\right) = 1$ (sinon on peut déjà dire que ce n'est pas un carré !). On construit alors k termes consécutifs de la suite BBS en récupérant le bit de poids faible des entiers $x_k = a^2, x_{k-1} = a^4, \ldots, x_1 = a^{2^k} \bmod n$. Si l'on peut déduire $x_{k+1} = \begin{cases} a \text{ si } a \text{ résidu quadratique} \\ -a \text{ sinon} \end{cases}$ avec un avantage, alors on sait répondre au problème de résiduosité quadratique pour a avec le même avantage.

Application 3 : pile ou face par téléphone

Alice et Bob souhaitent jouer à pile ou face par téléphone. Ils utilisent le protocole cryptographique suivant :

1. Bob choisit p, q deux grands nombres premiers, calcule $n = pq$ et transmet n à Alice. Il choisit également x mod n tel que $\left(\frac{x}{n}\right) = 1$ aléatoire et l'envoie à Alice.
2. Alice choisit un élément $\epsilon \in \{1; -1\}$ au hasard et l'envoie à Bob.
3. Bob compare ϵ et $\left(\frac{x}{p}\right)$:
   — s'ils sont égaux, le résultat du tirage au sort est pile ;
   — sinon, le résultat est face.
   Bob transmet le résultat à Alice et justifie ce résultat en lui transmettant p et q.
4. Alice vérifie la primalité de p et q, calcule $\left(\frac{x}{p}\right)$ et $\left(\frac{x}{q}\right)$ afin de vérifier le résultat transmis par Bob.

Alice qui ne connait pas la factorisation de n n'est pas en mesure de dire si x est un carré ou pas, donc ne peut pas tricher.

5 Logarithme discret

La fonction exponentielle modulaire $x \mapsto g^x\ [p]$ (avec p premier) permet de définir une autre fonction à sens unique (qui, contrairement au cas de la fonction de chiffrement RSA, n'est pas à trappe). On doit cependant préciser un peu les paramètres utilisés pour s'assurer que cette fonction est bien bijective.

5.1 Définition

La fonction $x \mapsto g^x\ [p]$ n'est bijective que lorsque g est une *racine primitive* modulo p. On en rappelle la définition

Définition (Racine primitive)**.** *Soit $n \in \mathbb{N}$. Un entier $g \in \mathbb{N}$ tel que $1 < g < n$ est appelé* racine primitive *de n si*

$$\begin{cases} g \wedge n = 1 \\ g^d \neq 1\ [n],\ \forall d \in \{1, \ldots, \varphi(n) - 1\} \end{cases}$$

En particulier, si p est premier, alors $g \in \mathbb{N}$ tel que $1 < g < p$ est une **racine primitive** de p si

$$\forall d \in \mathbb{N} \text{ tel que } 1 \leq d < p - 1,\ g^d \neq 1\ [p].$$

Exemple : 3 est une racine primitive de 31
(les puissances de 3 sont : $1, 3, 9, 27, 19, 26, 16, 17, 20, 29, 25, 13, 8, 24, 10, 30, 28, 22, 4, 12, 5, 15, 14, 11, 2, 6, 18, 23, 7, 21, 1$)

Théorème. *Soit p un nombre premier, et g une racine primitive de p. Alors l'application*

$$\exp : x \in \mathbb{Z}/(p-1)\mathbb{Z} \mapsto g^x \in (\mathbb{Z}/p\mathbb{Z})^\times$$

est bijective de réciproque

$$\log_g : y \in (\mathbb{Z}/p\mathbb{Z})^\times \mapsto \log_g(y) \in \mathbb{Z}/(p-1)\mathbb{Z}$$

Démonstration.
— l'application exp est bien définie : $g \wedge p = 1 \Rightarrow g^x \wedge p = 1 \Rightarrow g^x \in (\mathbb{Z}/p\mathbb{Z})^\times$
— exp injective : soient $x < y < p$ tels que $g^x = g^y$ alors $g^{y-x} = 1$ et $y - x < p - 1$; comme g est une racine primitive $x - y = 0$.

— exp est bijective puisque $\#(\mathbb{Z}/(p-1)\mathbb{Z}) = \#(\mathbb{Z}/p\mathbb{Z})^\times = p-1$

□

Heureusement, il n'est pas nécessaire de lister toutes les puissances d'un entier modulo p pour savoir si ce nombre est un racine primitive. On dispose en effet du test suivant [6] :

— trouver les facteurs premiers de $\varphi(p) = p-1 : p_1, \dots p_k$

— $m \in (\mathbb{Z}/p\mathbb{Z})^\times$ passe le test si

$$m^{\frac{p-1}{p_i}} \neq 1, \ \forall i = 1, \dots, k$$

Exemple : 11 est-elle une racine primitive de 31 ?
$p = 31$, $\varphi(p) = 30 = 2 \times 3 \times 5$
$11^{2\times 3} = 4$, $11^{3\times 5} = 30$, $11^{2\times 5} = 5 \Rightarrow 11$ est une racine primitive.

5.2 Échange de clef Diffie-Hellman

Cette fonction à sens unique permet de résoudre un célèbre problème cryptographique, à savoir comment s'échanger un secret en utilisant un canal non sécurisé sans s'entendre préalablement ?

Voici comment Alice et Bob procèdent :

1. Étape 1 : Alice choisit un nombre premier p et une racine primitive g de $\mathbb{Z}/p\mathbb{Z}$ et envoie p et g à Bob par un canal non sécurisé. [7]
2. Étape 2 :
   — Alice choisit un nombre secret a
   — Bob choisit un nombre secret b
3. Étape 3 :
   — Alice calcule $c_a = g^a \ [p]$ et le transmet à Bob via un canal non sécurisé
   — Bob calcule $c_b = g^b \ [p]$ et le transmet à Alice via un canal non sécurisé
4. Étape 4 :
   — Alice reçoit c_b et calcule $K_{ab} = (c_b)^a \ [p]$
   — Bob reçoit c_a et calcule $K_{ab} = (c_a)^b \ [p]$

On peut vérifier qu'alors Alice et Bob disposent bien d'une clef commune K_{ab} dont ils peuvent se servir pour chiffrer leur correspondance :

$$c_b^a = (g^b)^a = g^{ab} = (g^a)^b = c_a^b.$$

Charlie qui écoute les échanges connaît : p, g, c_a, c_b, mais ne peut pas en déduire les valeurs de a et b (sinon il sait résoudre le problème du logarithme discret !) ni la valeur de la clef $K_{ab} = g^{ab}$. Ce dernier problème est appelé problème de "Diffie-Hellman" ; on ne sait toujours pas comment le résoudre efficacement (mais par contre, il est clair que si l'on sait résoudre le logarithme discret, on sait résoudre ce problème).

6. En fait, l'idée repose sur le fait que l'ordre d'un élément g divise l'ordre du groupe (ici $\varphi(p) = p_1^{\alpha_1} \dots p_k^{\alpha_k}$), donc $ord(x) = p_1^{\beta_1} \dots p_k^{\beta_k}$ où $\beta_i \leq \alpha_i$.

7. On remarquera que le fait de prendre pour g une racine primitive est primordial, sinon il est possible que la clef obtenu soit $K_{ab} = 1$...

5.3 Chiffrement Elgamal

La fonction exponentielle permet également de définir un chiffrement asymétrique appelé *chiffrement Elgamal*. Voici la liste des paramètres de ce chiffrement en supposant que c'est Alice qui veut envoyer un message chiffré à Bob :

- *publics : p* un nombre premier, g une racine primitive de p, $K_{pub} = g^s\ [p]$ la clé publique de Bob
- *privé : s* le secret de Bob
- *hypothèse sur les messages :* on considère les messages M comme des nombres aléatoires de $\mathbb{Z}/p\mathbb{Z}$.

Ainsi pour envoyer son message chiffré,

1. Alice génère un nombre aléatoire t tel que $1 < t < p-1$
2. puis elle calcule $\begin{cases} C_1 = g^t\ [p] \\ C_2 = M.K_{pub}^t \end{cases}$ et envoie le couple (C_1, C_2) à Bob.

Bob va pouvoir déchiffrer en calculant $C_2.(C_1^{-1})^s$ à l'aide de sa clef secrète s.

Exemple de chiffrement avec El Gamal $p = 2579$, $g = 2$, $s = 765$ (secret de Bob), $K_{pub} = 2^{765} = 949\ [2579]$, $M = 1299$ (message d'Alice à chiffrer pour envoi à Bob)

1. Alice tire un nombre aléatoire $t = 853 \in \mathbb{Z}/p\mathbb{Z}$. Elle calcule $C_1 = g^t = 2^{853} = 435\ [2579]$ et $C_2 = M.K_{pub}^t = 1299 \times 949^{853} = 2396\ [2579]$, puis envoie (C_1, C_2) à Bob.
2. Bob reçoit $(C_1, C_2) = (435, 2396)$ et déchiffre en calculant $C_2.(C_1^{-1})^s = 2396 \times (435^{-1})^{765} = 1299\ [2579]$

On peut vérifier facilement que l'algorithme est correct : $C_2.(C_1^{-1})^s = M.(g^s)^t.(g^{-t})^s = M.g^{st-ts} = M$.
Un attaquant passif doit être capable de retrouver $K_{pub}^t = g^{ts}$ connaissant $C_1 = g^t\ [p]$ et $K_{pub} = g^s$ (problème de "Diffie-Hellman").

5.4 Méthode "pas de bébé / pas de géant" pour attaquer le problème du logarithme discret

On rappelle que le problème est le suivant :
soient p premier, g racine primitive de p, $y \in (\mathbb{Z}/p\mathbb{Z})^\times$, on souhaite trouver x tel que

$$g^x = y\ [p].$$

Le but de la méthode pas de bébé / pas de géant est d'éviter le calcul de toutes les puissances de g^i pour $0 \leq i < p-1$ en écrivant x sous la forme

$$x = ix_0 + j \text{ où } x_0 = \lceil \sqrt{p-1} \rceil,\ 0 \leq i < x_0,\ 0 \leq j < x_0.$$

On peut alors trouver x tel que $g^x = y\ [p]$ en cherchant i, j tels que $g^j = y.(g^{-x_0})^i\ [p]$. Le reste de l'algorithme consiste alors à

1. précalculer les valeurs g^j où $0 \leq j < x_0$;
2. calculer g^{-x_0} ;
3. tester au fur et à mesure si $y.(g^{-x_0})^i\ [p]$ est égal à une des valeurs g^j.

Exemple : résoudre $11^x = 25$ [31]

On fait les précalculs suivants : $11^0 = 1$ [31]
$11^1 = 11$ [31]
$11^2 = 28$ [31]
$11^3 = 29$ [31]
$11^4 = 9$ [31]
$11^5 = 6$ [31]

$11^{-6} = 8$ [31]

On cherche ensuite une collision :

$25.(11^{-6})^0 = 25$ [31]
$25.(11^{-6})^1 = 14$ [31]
$25.(11^{-6})^2 = 19$ [31]
$25.(11^{-6})^3 = 28$ [31]

On déduit alors que $11^2 = 25.(11^{-6})^3$ [31] et donc que $11^{20} = 25$ [31]. Le logarithme discret de 25 en base 11 modulo 31 est donc 20.

Deuxième partie

Codes correcteurs d'erreurs

6 Introduction

La transmission de données

Le schéma suivant modélise une transmission de données classique :

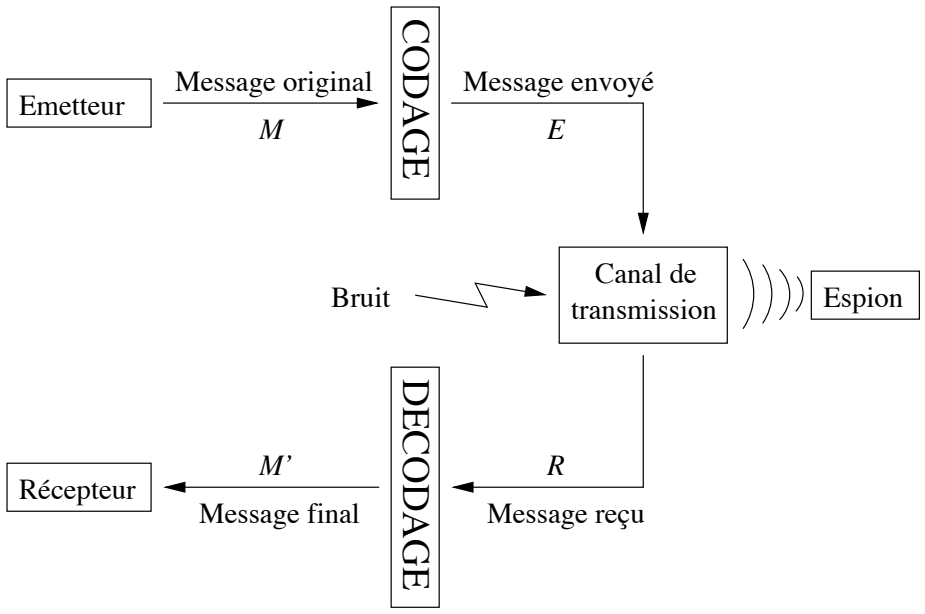


FIGURE 3 – Schéma général de communication

— Le message à transmettre M est d'abord **codé** en un message E. Le codage introduit un supplément d'information (redondances).
— Le message E est envoyé sur le canal de transmission.
— A cause de l'imperfection du canal, le message reçu R n'est pas forcément identique au message envoyé E : il peut contenir des erreurs.
— Le message reçu R est ensuite **décodé** : des erreurs éventuelles peuvent être détectées et corrigées.
— Le message décodé M' est identique au message d'origine M si il n'y a pas eu trop d'erreurs dans la transmission.

Ce schéma de communication est très général et recouvre des situations très différentes :
— transmissions inter-ordinateurs : liaisons ethernet, wifi, internet...
— transmissions intra-ordinateurs : disque dur ↔ mémoire vive ↔ processeur
— communications hertziennes (ondes électro-magnétiques) : radio, téléphones portables, mais aussi satellittes et sondes spatiales...
— transmissions de données via des supports physiques : disquettes, CD, DVD, lettres, livres...

Dans ces derniers cas, il est impossible ou trop difficile de demander une retransmission du message. On ne peut donc pas se contenter de détecter les erreurs, il faut aussi savoir les corriger.

Les erreurs de transmission : un problème très concret

Avec l'avènement des nouvelles technologies de communication, le nombre de données numériques échangées quotidiennement a augmenté exponentiellement ces dernières années, et cette évolution ne semble pas devoir s'arrêter, citons deux exemples :

— développement de l'Internet, avec des usages de plus en plus gourmands en bande passante (vidéo à la demande...)
— développement de la téléphonie portable, visioconférence...

Problème :

Les communications ne sont jamais parfaites. Quel que soit le canal utilisé, des erreurs de transmission se produisent inévitablement, avec des conséquences qui peuvent être graves si les données sont numériques. Il faut réaliser qu'une erreur d'un seul bit peut compromettre l'intégrité d'un système informatique si elle a lieu par exemple dans un fichier exécutable : instruction erronée, écriture au mauvais emplacement en mémoire...

Des chiffres :

Le taux d'erreur varie énormément suivant les canaux. En informatique, il est généralement compris entre 10^{-9} (un bit sur un milliard erroné, pour des communications intra-ordinateurs) et 10^{-4} (un bit sur dix mille erroné).

Exemple : avec un taux d'erreur de 10^{-6} et une connexion à 1 Mo/s, en moyenne 8 bits erronés sont transmis chaque seconde...

On peut envisager deux types de solutions pour s'assurer de l'intégrité des données transmises :

— On peut agir sur le canal de transmission, avec pour objectif de rendre le taux d'erreur négligeable. Mais il est impossible de remplacer du jour au lendemain toutes les infra-structures de réseaux existantes, sans parler de problèmes de coût. Et de toutes façons, le taux d'erreur nul n'existe pas : avec la miniaturisation, les micro-processeurs sont de plus en plus sensibles, certains rayons cosmiques pouvant perturber les transmissions à l'intérieur même des cartes-mères.
— On peut agir sur le message transmis, avec pour objectif d'être capable de détecter si des erreurs de transmission ont eu lieu, et éventuellement les corriger. C'est le principe des **codes correcteurs d'erreurs**.

Comment détecter et/ou corriger des erreurs ?

Un exemple simple :

Supposons qu'on veuille me transmettre un numéro de téléphone. Mon correspondant à deux possibilités :

1. Il me transmet `0169336000`. S'il y a des erreurs de transmission, par exemple si je reçois `0167336010`, je ne peux pas les détecter.
2. Il me transmet `zero un soixante-neuf trente-trois soixante zero zero`.
   S'il y a des erreurs de transmission, par exemple si je reçois `zero an sogxante-seuf trepte-troisksoixanqe zoro zerb`, je suis capable de corriger les erreurs et de retrouver le numéro.

Dans le premier cas, l'information est la plus concise possible.
Dans le deuxième cas au contraire, le message contient plus d'informations que nécessaire. C'est cette **redondance** qui permet la détection et la correction d'erreurs.

Exemples de redondance

L'utilisation de redondance pour améliorer une communication n'est pas vraiment une idée nouvelle...
Le type le plus simple de redondance est la répétition pure et simple du message.

- Comportement habituel au téléphone : répétition des numéros, épellation d'un nom ("I comme Irma, U comme Ursule, T comme Thérèse"...)
- Alphabet radio international : chaque lettre de l'alphabet est remplacée par un mot (Alpha Bravo Charlie Delta Echo ...)
- On constate aussi que l'orthographe traditionnelle comporte des redondances. Elle est donc moins sensible aux erreurs qu'une écriture phonétique (ou type SMS).

7 Principes généraux des codes détecteurs et correcteurs d'erreurs

7.1 Un peu de théorie

7.1.1 La notion de codage

Pour définir mathématiquement ce qu'est un codage, on introduit d'abord quelques notions :

Définition.
- *Un* alphabet *A est un ensemble fini, dont les éléments sont appelés* symboles.
- *Un* mot *de longueur n est un élément de A^n : c'est une suite de n symboles de A.*
- *Un* message *est une suite de symboles de A de longueur quelconque. On note A^* l'ensemble des messages.*

Si w est un élément de A^n (ou de A^*), on note w_i le i-ème symbole de w. On utilisera aussi la notation $w = w_1 w_2 \dots w_n$, où les w_i sont des symboles de A.

Le plus fréquemment on aura $A = \{0, 1\}$ (on parlera alors de mots et de codes *binaires*).

Définition. *Soient $w_1 \in A^n$ un mot de longueur n et $w_2 \in A^p$ un mot de longueur p. Le* concaténé *de w_1 et de w_2, noté $w_1.w_2$, est le mot de A^{n+p} obtenu en "écrivant à la suite" w_1 et w_2.*

Exemple : la concaténation des mots *azert* et *yuiop* est $azert.yuiop = azertyuiop$

Définition.
Un mot $u \in A^p$ est un préfixe *d'un mot $w \in A^n$ si u est "au début" de w, c'est-à-dire s'il existe $v \in A^{n-p}$ tel que $w = u.v$*
Un mot $u \in A^s$ est un suffixe *d'un mot $w \in A^n$ si u est "à la fin" de w, c'est-à-dire s'il existe $v \in A^{n-s}$ tel que $w = v.u$*

Exemple : *antic* est un préfixe et *lement* est un suffixe du mot *anticonstitutionnellement*

On peut maintenant définir ce qu'est un codage :

Définition. *Soient A et B deux alphabets. Un* codage *est une application injective $\phi : A^* \to B^*$.*

C'est donc simplement une règle spécifiant comment transformer un *message d'origine* M en un nouveau *message codé* $M' = \phi(M)$. Dans la suite du cours, les alphabets du message d'origine et du message codé seront le plus souvent les mêmes.

Etant donné un message $M' \in B^*$, le *décodage* consiste à rechercher un message $M \in A^*$ tel que $\phi(M) = M'$. S'il existe, un tel message M doit être unique : c'est pour cela que l'application ϕ doit être *injective*.

Les buts d'un codage peuvent être multiples :

— *Détection et correction d'erreurs* : le codage doit introduire de la redondance. C'est l'objet de ce cours.

— *Compression* (abréviations, code de Huffman...) – non abordé dans ce cours –

— *Changement d'alphabet* (code Morse, ASCII...).

— *Cryptographie* : le décodage doit être très difficile si on ne connaît pas ϕ.

7.1.2 Codage par blocs

Dans la pratique des codes correcteurs, on utilise principalement des *codages par blocs* :

1. Le message à transmettre est découpé en *blocs* (mots) de taille p fixée.
2. Chacun des blocs d'origine de taille p est codé séparément. Tous les blocs codés doivent avoir la même taille n.
3. Les blocs codés sont transmis successivement.

Exemple : le code Ascii transforme chaque caractère en un octet. C'est un donc un codage par blocs, et on a $p = 1$, $A =$l'ensemble des caractères habituellement utilisés pour écrire, $n = 8$, $B = \{0, 1\}$.
Le code Morse transforme chaque lettre en une suite de points et de tirets, mais ce n'est pas un codage par blocs : la longueur des lettres codées n'est pas la même.

L'étude d'un codage par blocs se restreint à l'étude du codage de mots de longueur p par des mots de longueur n. Un tel codage est entièrement décrit par l'*application de codage*

$$\phi : A^p \to B^n$$

L'application ϕ doit être injective, donc si $B = A$ on a nécessairement $n \geq p$.

Dans la suite de ce cours tous les codages seront des codages par blocs (sauf mention du contraire).

Terminologie

Soit $\phi : A^p \to B^n$ une application de codage.

Définition. — *On appelle* mot de code *tout élément de l'image de ϕ, i.e tout mot de B^n de la forme $\phi(w)$, $w \in A^p$.*
— *On appelle* code *l'ensemble des mots de code : $C = \{\phi(w),\ w \in A^p\}$. C'est l'image par ϕ de tous les mots de A^p.*

Comme ϕ est injective, il y a autant de mots de code que de mots dans A^p. Un code est donc juste un sous-ensemble à $card(A)^p$ éléments de B^n. Dans le cas où $A = B$, on parle de **code de taille** $(\mathbf{n}, \mathbf{p})$

Attention : ne pas confondre *code* (sous-ensemble de B^n) et *codage* (donné par l'application ϕ). On verra que la capacité de détection et de correction d'erreurs ne dépend que du code, et pas du codage, ce qui explique pourquoi on s'intéresse beaucoup plus au code qu'au codage.

Dans la suite de ce cours A et B seront toujours les mêmes (sauf mention du contraire).

7.1.3 Codage systématique

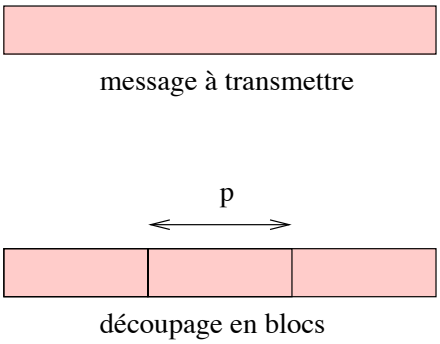
Définition. *Un codage $\phi : A^p \to A^n$ est dit* systématique *si le mot à coder se retrouve en tête du mot codé :*

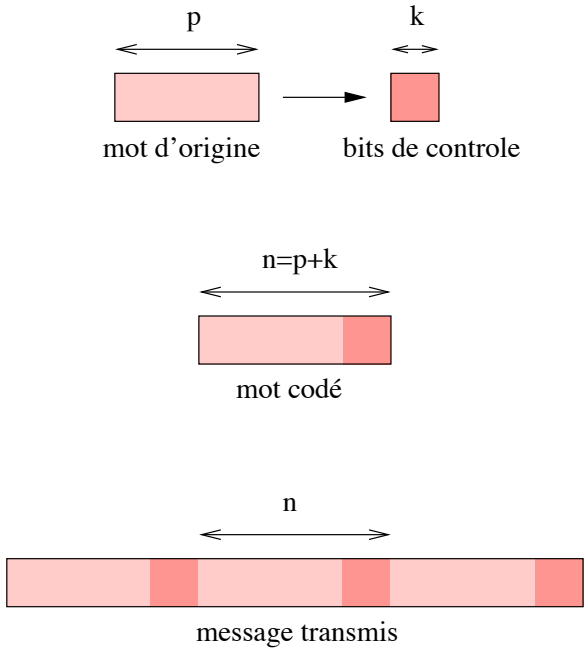
$$\forall w \in A^p,\ w \text{ est un préfixe de } \phi(w)$$

Un codage systématique consiste juste à rajouter $k = n - p$ *symboles* (ou bits) *de contrôle* à la fin du mot à coder. Les p premiers symboles du mot codé portent l'information ; les k derniers symboles du mot codé portent la redondance.

L'intérêt d'un codage systématique est que le décodage est immédiat : s'il n'y a pas d'erreurs de transmission il suffit de prendre les p premiers symboles du message reçu. Quasiment tous les codes peuvent être rendus systématiques

Codage systématique - illustration





7.1.4 Décodage et détection d'erreurs

Supposons que l'on ait reçu un mot w. Comment savoir si la transmission s'est bien déroulée ?

1. Si w n'est pas un mot du code C, on sait qu'il y a eu des erreurs dans la transmission et que le mot reçu est erroné.
2. Si w est un mot du code, on suppose qu'il n'y a pas eu d'erreurs de transmission et que le mot reçu est correct. On peut alors décoder w.

   En fait, deux cas peuvent s'être produits :
   — il n'y a pas eu d'erreur de transmission
   — il y a eu suffisamment d'erreurs de transmission, et bien positionnées, transformant le mot de code envoyé en un autre mot de code

   On rejette la deuxième possibilité comme étant la moins probable.

Il est bien sûr impossible de garantir avec certitude l'absence d'erreurs dans la transmission. Il existe cependant des codes permettant de réduire autant que voulu le risque d'erreurs accidentelles, au prix d'une augmentation considérable de la longueur du message envoyé.

7.2 Deux exemples fondamentaux

7.2.1 Codage par répétition simple

C'est le code détecteur d'erreurs le plus simple possible : on se contente de répéter chaque bit trois fois. Application de codage : $0 \mapsto 000$, $1 \mapsto 111$

C'est un codage systématique, de taille $(3, 1)$, sur l'alphabet $A = \{0, 1\}$. Les mots du code sont 000 et 111. Ce code peut détecter deux erreurs et corriger une erreur (cf TD).

7.2.2 Codage par bit de parité

Codage systématique de taille $(n, n-1)$ sur l'alphabet $A = \{0, 1\}$: on rajoute un bit de contrôle (*bit de parité*), de telle sorte que le nombre de 1 du mot codé (ou de manière équivalente la somme des chiffres) soit pair. Autrement dit :

— si le nombre de 1 du mot à coder est pair $\rightarrow$ bit de parité à 0
— si le nombre de 1 du mot à coder est impair $\rightarrow$ bit de parité à 1.

Exemples : 0110 donne 01100, 1101 donne 11011.

Les mots du code sont l'ensemble des $w \in A^n$ dont le nombre de 1 est pair. Ce code détecte un nombre impair d'erreurs mais ne peut pas en corriger (cf TD). En raison de sa simplicité et de sa concision (il ne rajoute qu'un bit par bloc) c'est un des codes les plus utilisées

7.3 Les erreurs de transmission

Pour pouvoir juger les performances d'un code détecteur / correcteur d'erreurs, il est nécessaire de savoir quelles genres d'erreurs sont susceptibles de se produire. Les erreurs de transmission ont naturellement un comportement aléatoire, mais qui peut se modéliser.

7.3.1 Modélisation des erreurs

Pour simplifier, on va supposer que :

1. les erreurs de transmission ne vont pas rajouter ou supprimer de symboles, seulement les modifier,
2. chaque symbole a la même probabilité p d'être modifié pendant la transmission,
3. si l'alphabet est de taille s, les $s-1$ possibilités de modification d'un symbole sont équiprobables,
4. les probabilités d'erreur sur chaque symbole transmis sont indépendantes.

Ces quatre hypothèses ne s'appliquent pas à l'ensemble des situations. On va les reprendre une par une.

1. les erreurs de transmission ne vont pas rajouter ou supprimer de symboles, seulement les modifier

Ce n'est pas le cas si le canal est un texte écrit. Erreurs de lecture classique : ri $\rightarrow$ n, rn $\rightarrow$ m...
Dans le cadre des transmissions informatiques, ce sont des erreurs peu fréquentes et relativement faciles à détecter au niveau matériel (problèmes de synchronisation d'horloges ou de longueur de trames) ; on ne s'en préoccupera donc pas.

2. chaque symbole a la même probabilité p d'être modifié pendant la transmission,
3. si l'alphabet est de taille s, les $s-1$ possibilités de modification d'un symbole sont équiprobables

Encore une fois, ce n'est pas le cas si le canal est un texte écrit. Certaines confusions sont plus fréquentes que d'autres : O et 0, I et l ...
Par contre c'est le cas pour une transmission informatique (en binaire, le point 3 est sans objet).

4. les probabilités d'erreur sur chaque symbole transmis sont indépendantes.

C'est sans doute le point le plus critiquable. Dans une transmission numérique, on distingue classiquement deux types d'erreurs :
- des erreurs apparaissant de façon *aléatoire*, liées à un bruit de fond électromagnétique. Ce sont celles que l'hypothèse modélise.
- des erreurs apparaissant *en rafale*, dont les causes peuvent être multiples (rayure sur un CD, perturbation électromagnétique, parasite...). Elles se manifestent par beaucoup de symboles erronés à très peu d'écart et mettent en défaut les codages par blocs classiques.

En règle générale, les hypothèses faites sur les erreurs modélisent convenablement un bruit de fond aléatoire lors de la transmission d'un signal binaire.

7.3.2 Nombre d'erreurs - loi binomiale

Avec les hypothèses que l'on a faites, le nombre d'erreurs de transmission (noté Z) suit une *loi binomiale* $B(n,p)$, où n est la taille d'un bloc et p le *taux d'erreurs*.

$$P(Z=k) = C_n^k\, p^k (1-p)^{n-k}$$

On supposera toujours que p est suffisament faible et n suffisament petit, de telle sorte que le cas le plus probable est qu'il n'y ait pas (ou peu) d'erreurs de transmission :

$$P(Z=0) > P(Z=1) > P(Z=2) > \cdots > P(Z=n)$$

Application : On utilise un canal sur lequel le taux d'erreurs est de 10^{-2} pour envoyer des messages de 64 bits.

Probabilité d'avoir :
- aucune erreur de transmission : $P(Z{=}0) = (1-p)^n = 0{,}99^{64} \simeq 0{,}526$
- une erreur de transmission : $P(Z{=}1) = np(1-p)^{n-1} \simeq 0{,}340$
- deux erreurs de transmission : $P(Z{=}2) = \frac{n(n-1)}{2} p^2 (1-p)^{n-2} \simeq 0{,}108$
- trois erreurs de transmission : $P(Z{=}3) = C_n^3 p^3 (1-p)^{n-3} \simeq 0{,}023$
- quatre erreurs de transmission : $P(Z{=}4) = C_n^4 p^4 (1-p)^{n-4} \simeq 0{,}003$
- plus de quatre erreurs : $P(Z{>}4) < 0{,}0005$

7.4 Distance de Hamming et correction d'erreurs

7.4.1 Distance entre deux mots

Définition. *Soit A un alphabet, et soient w et w' deux mots sur A de longueurs n. On appelle* distance de Hamming *(ou simplement distance) entre w et w', et on note $d(w,w')$, le nombre de positions où*

w et w' ont des symboles différents :

$$d(w, w') = card\{i \in [1, n] \ , \ w_i \neq w'_i\}$$

Alternativement, c'est aussi :
- le nombre de symboles à modifier pour passer de w à w'
- le nombre d'erreurs nécessaires pour confondre w et w'

Exemple : $d(1101, 1001) = 1$

On vérifie que la distance de Hamming est une *distance* sur A^n, c'est-à-dire qu'elle vérifie les trois propriétés suivantes :

1. $d(x, y) \geq 0$ pour tous éléments x et y de A^n, et $d(x, y) = 0$ si et seulement si $x = y$ (*positivité*)
2. $d(x, y) = d(y, x)$ quels que soient les éléments x et y de A^n (*symétrie*)
3. $d(x, y) \leq d(x, z) + d(z, y)$ quels que soient les éléments x, y et z de A^n (*inégalité triangulaire*).

7.4.2 Application à la correction d'erreurs

Supposons que l'on ait reçu un mot w. Comment retrouver le message transmis ?

1. Si w est un mot du code, on suppose qu'il n'y a pas eu d'erreurs de transmission (cas le plus probable) et que le mot reçu est correct. On peut alors décoder w.
2. Si w n'est pas un mot du code C, on sait qu'il y a eu des erreurs dans la transmission et que le mot reçu est erroné.
   On recherche alors le mot du code *le plus proche* de w pour la distance de Hamming.
   - Si le mot de code le plus proche de w est unique, on remplace w par ce mot de code v : on a *corrigé* les erreurs de transmission.
   - Sinon, on ne peut pas choisir parmi les mots de code les plus proches : la correction d'erreur échoue.

7.4.3 Distance minimale d'un code

Définition. *On appelle* distance minimale *(ou simplement distance) d'un code C, et on note $d(C)$, la plus petite distance de Hamming entre deux mots distincts de C :*

$$d(C) = \min\{d(x, y) \ ; \ x \in C, \ y \in C, \ x \neq y\}$$

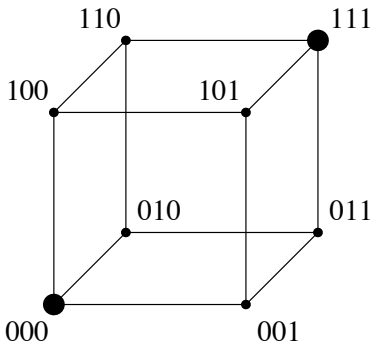
C'est donc le nombre minimum d'erreurs permettant de transformer un mot du code en un autre mot du code. Plus la distance minimale du code est grande, plus les mots du code sont "dispersés" dans A^n.

Exemples :
- Codage par répétition : $C = \{000, 111\}$
  La distance minimale du code est donc $d(000, 111) = 3$.

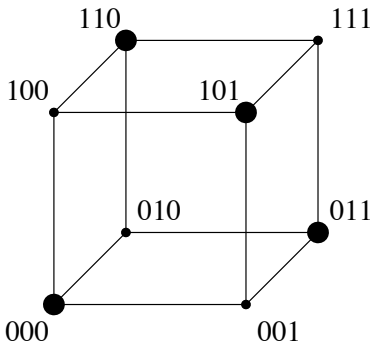
— Codage par bit de parité : $C = \{000, 011, 101, 110\}$
La distance minimale du code est 2.

On peut représenter les éléments $\{0,1\}^3$ par les sommets d'un cube ; les arêtes du cube relient les mots à distance 1 l'un de l'autre.

Codage par répétition :



Bit de parité :



La distance minimale du code se lit alors comme le nombre minimum d'arêtes entre deux sommets correspondant à des mots du code.

7.4.4 Capacité de correction d'un code

La distance minimale d'un code est directement liée à ses performances :

Théorème. *Soit C un code de distance minimale d, et soit w' un mot reçu, comportant e erreurs de transmission par rapport au mot envoyé w.*

— *Si $e < d$, le code C permet de détecter que le mot reçu w' est erroné.*

— *Si $e < d/2$, le code C permet de corriger le mot reçu.*

Un code C de distance minimale d permet donc de détecter jusqu'à $d-1$ erreurs et de corriger jusqu'à $E(\frac{d-1}{2})$ erreurs.

Démonstration.
Cas $e < d$: On suppose qu'il y a eu au moins une erreur de transmission, donc $w' \neq w$. Par définition de la distance minimale, pour tout mot du code v différent de w, on a $d(w,v) \geq d$. Or comme $d(w,w') = e < d$, w' ne peut pas être un mot du code. Le mot reçu w', n'étant pas un mot du code, est donc détecté comme erroné.

Cas $e < d/2$: Pour tout mot du code v différent de w, on a par définition $d \leq d(w,v)$. L'inégalité triangulaire donne : $d(w,v) \leq d(w,w') + d(w',v)$. Maintenant par hypothèse $d(w,w') = e < d/2$. Donc $d < d/2 + d(w',v)$, soit $d/2 < d(w',v)$. Et comme $d(w,w') < d/2$, on en déduit que pour tout mot du code v différent de w, $d(w',w) < d(w',v)$. Le mot reçu w' est bien corrigé, car w est le mot du code le plus proche de w'.

□

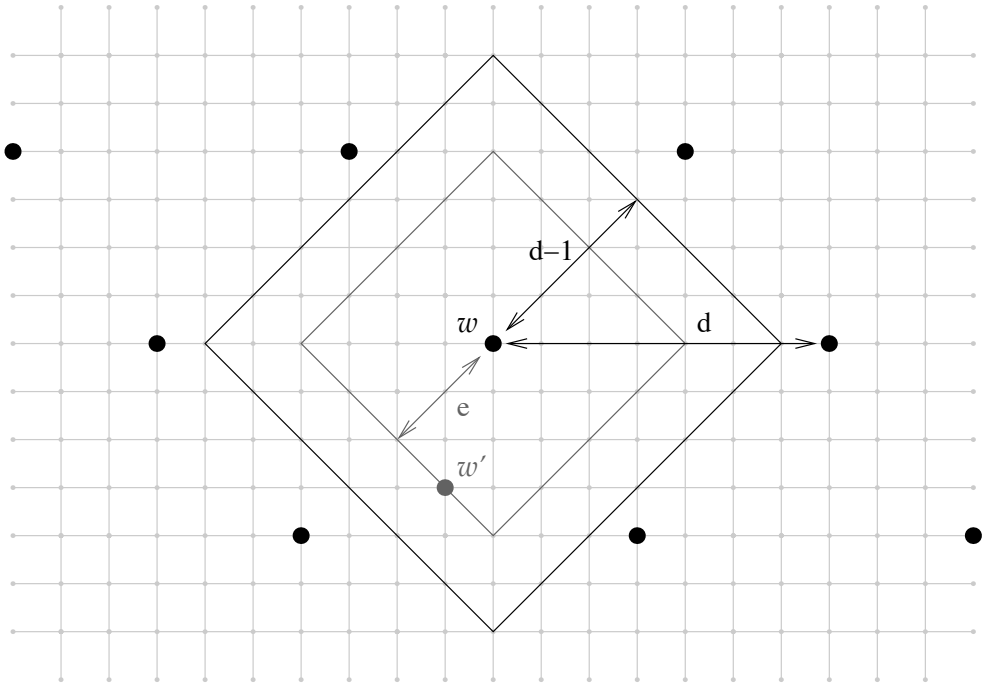


FIGURE 4 – A distance $e \leq d-1$ de w, il n'y a pas de mot du code.

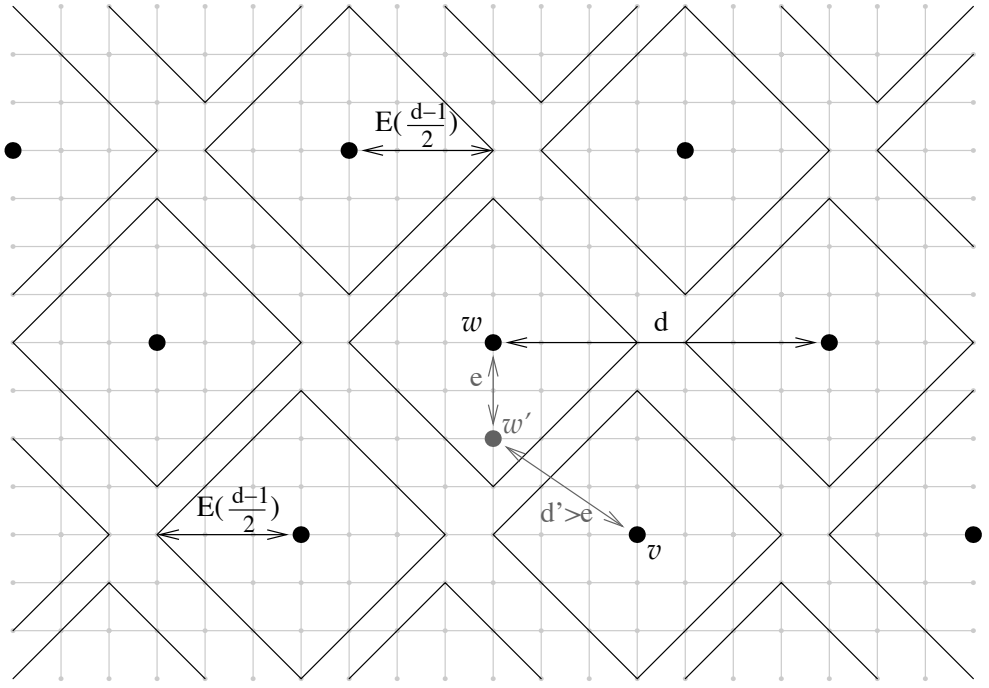


FIGURE 5 – Si $e \leq E(\frac{d-1}{2})$, w est le mot du code le plus proche de w'.

Remarques :
Dès que le nombre d'erreurs est supérieur ou égal à d, on peut trouver des messages envoyés w tels que le message reçu soit aussi un mot du code.
Dès que le nombre d'erreurs est supérieur ou égal à $d/2$, on peut trouver des messages envoyés w tels que le message reçu soit plus proche d'un autre mot du code que de w.

Exemples :
- Codage par répétition : on a $d = 3$, donc ce code détecte jusqu'à 2 erreurs et corrige une erreur.
- Codage par bit de parité : on a $d = 2$, donc ce code détecte une erreur et ne peut pas en corriger.

7.4.5 Performance d'un code correcteur

La performance d'un code détecteur/correcteur d'erreurs se juge par :

- sa capacité de détection et de correction d'erreurs, mesurée par la distance minimale d du code, d'une part
- l'augmentation de la taille des messages, mesurée par le rapport n/p, d'autre part.

Un bon code est un code qui, à p et n fixé, maximise d (ou qui, à p et d fixé, minimise n).

Il y a un compromis à trouver entre ces deux critères. Par exemple, si le taux d'erreurs est très faible et les messages faciles à réexpédier, savoir détecter une erreur peut suffire. Par contre si le taux d'erreurs est élevé et/ou les messages couteux à réexpédier, on va privilégier la capacité de correction au débit.

7.4.6 Inégalité et borne de Hamming

La taille (n, p) d'un code et sa distance minimale d sont reliées par l'*inégalité de Hamming* :

Théorème. *Soit A un alphabet ayant s symboles distincts et C un code de taille (n, p), de distance minimale d. On note $t = E((d-1)/2)$ la capacité de correction d'erreurs de C. Alors*

$$\sum_{i=0}^{t} C_n^i (s-1)^i \leq s^{n-p}.$$

En particulier si $A = \{0, 1\}$ on a $s = 2$ et l'inégalité devient

$$\sum_{i=0}^{t} C_n^i \leq 2^{n-p}.$$

Cette inégalité permet de :
- majorer d en connaissant p et n (ce qu'on appelle la *borne de Hamming*)
- majorer p en connaissant d et n
- minorer n en connaissant d et p

L'interprétation de l'inégalité de Hamming est que les boules de rayon t centrées en les mots du code doivent être disjointes. Un code pour lequel on a l'égalité $sum_{i=0}^{t} C_n^i (s-1)^i = s^{n-p}$ est un *code parfait* : les boules de rayon t forment une partition de A^n, et on a alors $d = 2t + 1$. En dehors des codes de Hamming que l'on verra au chapitre suivant, on connaît très peu de codes parfaits non triviaux.

8 Codes binaires linéaires

Dans la suite du cours, on se placera toujours sur l'alphabet $A = \{0, 1\}$, sauf mention du contraire.

La notion de code telle que définit auparavant est trop générale pour être utile. On va se restreindre à une classe de code particulière, les codes binaires linéaires, qui ont les avantages suivants :

— applicables aux transmissions numériques : l'alphabet utilisé est $\{0, 1\}$
— réguliers : les mots du code sont "régulièrement" espacés dans A^n (on verrra ce que cela signifie)
— faciles à décrire : la donnée d'une matrice détermine le code.

8.1 Arithmétique dans $\mathbb{F}_2^n$

8.1.1 Règles de calculs dans $\mathbb{F}_2$

On note $\mathbb{F}_2$ l'ensemble $\{0, 1\}$ muni des deux lois internes suivantes :

Addition, notée $+$, correspond au "ou exclusif" en logique booléenne :

$+$	0	1
0	0	1
1	1	0

Multiplication, notée $\times$, correspond au "et" en logique booléenne :

$\times$	0	1
0	0	0
1	0	1

Ces deux opérations ont les propriétés usuelles de l'addition et de la multiplication : associativité, commutativité, distributivité de la multiplication par rapport à l'addition. Mathématiquement, on dit que $\mathbb{F}_2$ est un *corps*, c'est en fait exactement l'ensemble $\mathbb{Z}/2\mathbb{Z}$ que l'on a défini en première partie de ce cours.

8.1.2 Règles de calculs dans $\mathbb{F}_2^n$

La notation $\mathbb{F}_2^n$ désigne l'ensemble des n-uplets d'éléments de $\mathbb{F}_2$, identifié avec l'ensemble des mots binaires de longueur n.

Sur $\mathbb{F}_2^n$, on définit deux lois :

— l'addition, noté $+ : \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{F}_2^n$

$$(v_1, v_2, \ldots, v_n) + (w_1, w_2, \ldots, w_n) = (v_1 + w_1, v_2 + w_2 \ldots, v_n + w_n)$$

— la multiplication scalaire, noté $\times$ ou $\cdot : \mathbb{F}_2 \times \mathbb{F}_2^n \to \mathbb{F}_2^n$

$$\lambda \cdot (v_1, v_2, \ldots, v_n) = (\lambda \cdot v_1, \lambda \cdot v_2, \ldots, \lambda \cdot v_n)$$

On utilise la notation 0_n pour désigner le n-uplet $(0, 0, \ldots, 0) = 00 \ldots 0$.

Remarque : l'addition dans $\mathbb{F}_2^n$est une opération différente de la somme de deux nombres écrits en binaire. En particulier, **on ne fait pas de report de retenue.**

L'addition et la multiplication par un scalaire ont les mêmes propriétés dans $\mathbb{F}_2^n$ que dans $\mathbb{R}^n$. Mathématiquement, on dit que $\mathbb{F}_2^n$ est un $\mathbb{F}_2$-*espace vectoriel*. Toutes les notions vues sur les espaces vectoriels réels s'appliquent à $\mathbb{F}_2^n$ (combinaisons linéaires, sous-espaces vectoriels, bases, applications linéaires, matrices ...).

Attention : Pour tout $x \in \mathbb{F}_2^n$, on a $x + x = 0_n$, c'est-à-dire $x = -x$! En particulier, $x + y = z \Leftrightarrow x = y + z$

8.1.3 Matrices

On note $\mathcal{M}_{p,n}(\mathbb{F}_2)$ l'ensemble des matrices de taille $p \times n$ à coefficients dans $\mathbb{F}_2$.

La multiplication matricielle $\mathcal{M}_{p,k}(\mathbb{F}_2) \times \mathcal{M}_{k,n}(\mathbb{F}_2) \to \mathcal{M}_{p,n}(\mathbb{F}_2)$ s'effectue comme pour les matrices réelles (en utilisant les opérations de $\mathbb{F}_2$).

Exemple :

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

Convention : L'usage est d'écrire les éléments de $\mathbb{F}_2^n$ "en ligne", comme des mots (ex : $(0,1,1) \leftrightarrow 011$), et pas "en colonne", comme des vecteurs.

On **identifie** donc $\mathbb{F}_2^n$ avec $\mathcal{M}_{1,n}(\mathbb{F}_2)$, l'ensemble des matrices lignes de longueur n.

Avec cette convention, toute matrice $M \in \mathcal{M}_{p,n}(\mathbb{F}_2)$ donne une application $f_M : \mathbb{F}_2^p \to \mathbb{F}_2^n$.

Exemple : $M = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}$

$$f_M(1101) = (1\ 1\ 0\ 1) \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix} = 111100$$

On rappelle brièvement les notions d'algèbre linéaire utilisées dans la suite du cours.

Définition. *Une application $f : \mathbb{F}_2^p \to \mathbb{F}_2^n$ est* linéaire *si pour tous mots x, y de $\mathbb{F}_2^p$, on a $f(x + y) = f(x) + f(y)$.*

Théorème.
Pour toute application linéaire $f : \mathbb{F}_2^p \to \mathbb{F}_2^n$, il existe une unique matrice $M \in \mathcal{M}_{p,n}(\mathbb{F}_2)$ telle que $f = f_M$.
Réciproquement, pour toute matrice $M \in \mathcal{M}_{p,n}(\mathbb{F}_2)$, l'application f_M est linéaire.

Définition. — *Une partie non vide E de $\mathbb{F}_2^n$ est un* sous-espace vectoriel *si elle est stable par addition :*

$$\forall x, y \in E,\ x + y \in E$$

— *Une* base *d'un sous-espace vectoriel E est une famille $(e_1, e_2, \dots e_p)$ d'éléments de E, telle que tout élément de E s'écrive de façon unique comme combinaison linéaire des $e_1, \dots, e_p$:*

$$\forall x \in E,\ \exists! \lambda_1, \dots, \lambda_p \in \mathbb{F}_2,\ x = \lambda_1 e_1 + \dots + \lambda_p e_p$$

Théorème. *Soit E un sous-espace vectoriel de $\mathbb{F}_2^n$.*

— *Toutes les bases de E ont le même nombre d'éléments ; ce nombre est appelé la* dimension *de E.*

— *Si B est une base de E, l'ensemble des combinaisons linéaires des éléments de B est égal à E.*

— *Le cardinal de E est 2^p, où p est la dimension de E.*

Exemples :

— $\mathbb{F}_2^n$ est un sous-espace vectoriel trivial de $\mathbb{F}_2^n$, sa dimension est n. Une base de $\mathbb{F}_2^n$ est 100...0, 010...0, 001...0, ..., 000...1.
Exemple de décomposition : $1011 = 1 \times 1000 + 0 \times 0100 + 1 \times 0010 + 1 \times 0001$

— $E = \{10, 01, 11\}$ n'est pas un sous-espace vectoriel de $\mathbb{F}_2^2$: en effet $10 + 10 = 00 \notin E$.

— L'ensemble des mot binaires de longueur n ayant un nombre pair de 1 est un sous-espace vectoriel de $\mathbb{F}_2^n$, de dimension $n - 1$. Une base en est 100...01, 010...01, 001...01, ..., 000...11.
Exemple de décomposition : $1010 = 1 \times 1001 + 0 \times 0101 + 1 \times 0011$

8.1.4 Distance de Hamming entre mots binaires

Si $A = \mathbb{F}_2$, on vérifie que la distance de Hamming est *invariante par translation* :

$$d(x, y) = d(x + z, y + z) \quad \forall x, y, z \in \mathbb{F}_2^n$$

En particulier, $d(x, y) = d(x + y, y + y) = d(x + y, 0_n)$

Définition. *Le* poids $p(w)$ *d'un mot binaire w est son nombre bits égaux à 1.*

Le poids d'un mot binaire w est égal à sa distance au mot nul : $p(w) = d(w, 0_n)$. On a alors pour tous mots binaires x et y de même longueur,

$$d(x, y) = d(x + y, 0_n) = p(x + y).$$

8.2 Généralités sur les codes linéaires

8.2.1 Définition et premières propriétés

Définition. *Un code binaire C est* linéaire *si la somme de deux mots quelconques du code est encore un mot du code :*

$$\forall w_1, w_2 \in C,\ w_1 + w_2 \in C$$

Un code linéaire est donc un *sous-espace vectoriel* de $\mathbb{F}_2^n$. En particulier le nombre de mots d'un code linéaire est de la forme 2^p, où p est la dimension de C.

Remarque : dans un code linéaire, le mot 0_n est toujours un mot du code :
en effet si x est un mot du code quelconque, alors $x + x = 0_n$ est aussi un mot du code.

Un des intérêts des codes linéaires est la propriété suivante, qui montre que la distance minimale d'un code est beaucoup plus facile à déterminer quand le code est linéaire.

Théorème. *Soit C un code linéaire.*

— *La distance minimale du code est égale au poids du mot du code non nul de plus petit poids :*

$$d(C) = \min\{p(w) \ ; \ w \in C, \ w \neq 0_n\}$$

— *Pour tout mot du code w, la plus petite distance entre w et un autre mot du code est toujours égale à $d(C)$.*

Démonstration. On rappelle que $p(w) = d(0_n, w)$, et que 0_n est un mot du code. Donc :

$$\begin{aligned} \min\{p(w) \ ; \ w \in C, \ w \neq 0_n\} &= \min\{d(0_n, w) \ ; \ w \in C, \ w \neq 0_n\} \\ &\leq \min\{d(v, w) \ ; \ v, w \in C, \ w \neq v\} \\ &\leq d(C) \end{aligned}$$

Réciproquement, si x, y sont deux mots (distincts) du code tels que $d(x, y) = d(C) = \min\{d(v, w) \ ; \ v, w \in C, \ w \neq v\}$, alors $d(C) = d(x, y) = p(x + y)$ et $x + y$ est un mot du code non nul. Donc :
$d(C) = p(x + y) \leq \min\{p(w) \ ; \ w \in C, \ w \neq 0_n\}$.

Pour le deuxième point : soit x un mot du code tel que $d(C) = p(x) = \min\{p(w) \ ; \ w \in C, \ w \neq 0_n\}$.
Alors $w + x$ est un mot du code, et $d(w, w + x) = p(w + w + x) = p(x) = d(C)$.

□

8.2.2 Matrice génératrice

Définition. *On appelle* matrice génératrice *d'un code linéaire C toute matrice dont les lignes forment une base de C.*

Un code linéaire est complètement décrit par une matrice génératrice.

Exemple :

$G = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$ est la matrice génératrice du code $\{00000, 00110, 11000, 11110, 10111, 10001,$
$01111, 01001\}$. La distance de ce code est 2.

8.2.3 Codages linéaires

Définition. *Un codage est* linéaire *si l'application de codage $\phi : A^p \to A^n$ est linéaire, c'est-à-dire si $\phi(x+y) = \phi(x) + \phi(y)$ pour tous mots x, y de A^p.*

Le code correspondant à un codage linéaire (i.e. l'image de ϕ) est un code linéaire.

L'application ϕ est linéaire donc peut s'écrire sous forme *matricielle*. La matrice correspondant à ϕ est une matrice génératrice du code.

Les codages linéaires sont donc faciles à décrire : ils sont explicités par la donnée d'une matrice.

Exemples :

- Le code $\{000, 111\}$ est linéaire. Sa matrice génératrice est $G = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$
- Le code $\{101, 010\}$ n'est pas linéaire.
- Le codage par bit de parité est linéaire. La matrice génératrice correspondante (en taille $(4,3)$) est $G = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$

Proposition. *Un codage linéaire est systématique si et seulement si la matrice génératrice G correspondante est la juxtaposition de la matrice identité I_p et d'une* matrice de parité $P \in \mathcal{M}_{p,n-p}(\mathbb{F}_2)$ *:*

$$G = (I_p \dot{:} P)$$

Rappel : un codage est systématique si le mot d'origine est un préfixe du mot codé.

Exemples :

- Codage $(3,1)$ par répétition : $G = \left(\begin{array}{c|cc} 1 & 1 & 1 \end{array}\right)$, $P = \begin{pmatrix} 1 & 1 \end{pmatrix}$
- Codage $(4,3)$ par bit de parité : $G = \left(\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{array}\right)$, $P = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$

Etant donné un code linéaire C, on aimerait en connaître un codage systématique.
Cela revient à chercher une *matrice génératrice standard* du code C, c'est-à-dire une matrice génératrice de la forme $G = (I_p \dot{:} P)$.

Si elle existe, la matrice génératrice standard est unique. On l'obtient en appliquant l'*algorithme de Gauss* à une matrice génératrice quelconque de C.

Exemple : Soit C le code définit par sa matrice génératrice standard $G = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$. On applique l'algorithme de Gauss à G :

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$$\to \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix} \quad L_2 \leftarrow L_2 + L_1$$

$$\to \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \begin{matrix} L_1 \leftarrow L_1 + L_2 \\ \\ L_3 \leftarrow L_3 + L_2 \end{matrix}$$

$$\to \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \quad L_1 \leftarrow L_1 + L_3$$

Certains codes linéaires n'admettent pas de matrices génératrices standard, par exemple $\{000, 001, 010, 011\}$.

8.3 Correction d'erreur : méthode du tableau standard

8.3.1 Classes d'équivalence

Le décodage pour un code "quelconque" est fastidieux : il faut déterminer, pour chaque mot susceptible d'être reçu, le mot du code le plus proche. La structure régulière d'un code linéaire permet de simplifier en partie ce travail.

Soit C un code linéaire de taille (n, p). On définit sur $\mathbb{F}_2^n$ la **relation binaire** $\mathcal{C}$:

$$u \,\mathcal{C}\, v \Leftrightarrow u + v \in C$$

Proposition.
La relation $\mathcal{C}$ est une relation d'équivalence*, possédant 2^{n-p} classes d'équivalence. Le code C est la classe d'équivalence de 0_n.*

Démonstration. Montrons d'abord que $\mathcal{C}$ est une relation d'équivalence :
- *Réflexivité :* pour tout $x \in \mathbb{F}_2^n$, on a $x + x = 0_n$, et 0_n est un mot du code car le code est linéaire ; donc $x \,\mathcal{C}\, x$.
- *Symétrie :* si $x \,\mathcal{C}\, y$, alors par définition $x + y \in C$, et on a directement $y + x = x + y \in C$, soit $y \,\mathcal{C}\, x$.
- *Transitivité :* si $x \,\mathcal{C}\, y$ et $y \,\mathcal{C}\, z$, alors par définition $x + y \in C$ et $y + z \in C$. On a $x + z = x + (y + y) + z = (x + y) + (y + z)$; comme le code est linéaire et que $x + y$ et $y + z$ sont des mots du code, $x + z$ est un mot du code : $x \,\mathcal{C}\, z$.

La relation $\mathcal{C}$ est donc bien une relation d'équivalence. On note $[x]$ la classe d'équivalence d'un mot x pour la relation $\mathcal{C}$.

Si w est un mot du code alors $(x + w) \,\mathcal{C}\, x$, c'est-à-dire $x + w \in [x]$: en effet $(x + w) + x = w \in C$. On peut alors regarder l'application $\tau_x : C \to [x]$, $w \mapsto x + w$.

— Elle est *injective* : $\tau_x(w) = \tau_x(w') \Rightarrow x + w = x + w' \Rightarrow w = w'$.
— Elle est *surjective* : si $y \in [x]$, alors $x+y \in C$ est un antécédent de y : $\tau_x(x+y) = x+x+y = y$.
L'application τ_x est donc bijective. On en déduit que chaque classe d'équivalence a le même nombre d'éléments que C, c'est-à-dire 2^p.

Les classes d'équivalence forment une partition de $\mathbb{F}_2^n$ par des sous-ensembles de cardinal 2^p. Il y a donc $card(\mathbb{F}_2^n)/2^p = 2^{n-p}$ classes d'équivalence.

Enfin, il est clair que la classe de 0_n est C : en effet $w\,\mathcal{C}0_n \Leftrightarrow w + 0_n \in C \Leftrightarrow w \in C$. □

8.3.2 Vecteur d'erreur

On se place dans le cadre d'une transmission utilisant un code linéaire C.

Définition. *Soit $w \in C$ le mot du code envoyé et soit $w' \in \mathbb{F}_2^n$ le mot reçu. On appelle* vecteur d'erreur *de la transmission le mot $e = w + w'$.*

Le vecteur d'erreur vérifie $w = w' + e$: il indique précisément quels sont les erreurs ayant eu lieu pendant la transmission.

Propriété. *Le vecteur d'erreur e appartient à la classe d'équivalence du mot reçu w'.*

En pratique, on ne connait que le mot reçu w', et on veut trouver le mot du code le plus proche de w'.

Cela revient à rechercher, parmi tous les vecteurs d'erreur possibles, *c'est-à-dire parmi tous les éléments de la classe d'équivalence de w'*, celui de plus petit poids.

Une première méthode de décodage consiste donc à déterminer, préalablement, un élément de plus petit poids dans chacune des classes d'équivalence du code : c'est la méthode du *tableau standard*.

8.3.3 Construction du tableau standard

Un tableau standard pour le code C contient tous les mots de $\mathbb{F}_2^n$. Sur chaque ligne sont rangés tous les éléments d'une même classe d'équivalence.

On procède de la façon suivante :
— première ligne : on liste les mots de C, en commençant par $a_1 = 0_n$
— deuxième ligne : on choisit un mot a_2, de poids minimum, qui n'est pas déjà dans le tableau (on obtient a_2 en parcourant tous les mots de poids 1, puis 2, 3...). On remplit alors la ligne en inscrivant $a_2 + x$ dans la colonne ayant au sommet le mot du code x.
— troisième ligne : on choisit un mot a_3, de poids minimum, qui n'est pas déjà dans le tableau. On remplit alors la ligne en inscrivant $a_3 + x$ dans la colonne ayant au sommet le mot du code x.
— On continue de la même façon jusqu'à que tous les mots du code soient inscrits et que les 2^{n-p} lignes soient remplies.

Exemple : Soit C le code linéaire de taille $(4, 2)$, de matrice génératrice $G = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$, c'est-à-dire le code $C = \{0000, 1011, 0101, 1110\}$.

Construction du tableau standard :

0000	1011	0101	1110
1000	0011	1101	0110
0100	1111	0001	1010
0010	1001	0111	1100

Remarque : le tableau standard n'est pas unique. En particulier, si dans une classe d'équivalence le mot de plus petit poids n'est pas unique, le choix de celui qui est en tête de ligne dans le tableau aura une influence sur le décodage.

8.3.4 Utilisation du tableau standard

Soit w' le mot reçu. On cherche sa position dans le tableau, puis on le corrige par le mot w situé en haut de la même colonne. Cela revient à ajouter à w' le vecteur d'erreur a_i situé en tête de sa ligne.

Par construction du tableau, a_i est un élément de poids minimum dans sa classe d'équivalence. Donc w est bien un mot du code le plus proche de w', ce qui justifie cette méthode de décodage.

Exemple : On utilise le tableau standard du paragraphe précédent.

— Si on reçoit le mot 0111 : on le corrige en 0101. Le vecteur d'erreur est 0010.

— Si on reçoit le mot 0110 : on le corrige en 1110. Le vecteur d'erreur est 1000.

8.3.5 Efficacité de la méthode

Préliminaire : Soit w un message envoyé, on note p le taux d'erreur (à ne pas confondre avec le poids d'un mot, aussi noté p). La probabilité que le message reçu soit un mot donné w' dépend uniquement du vecteur d'erreur $e = w' + w$, qui indique quelles erreurs de transmission doivent se produire.

$$\begin{aligned} P(\text{message reçu}=w') &= P(\text{vecteur d'erreur}=e) \\ &= p^{p(e)}\,(1-p)^{n-p(e)} \end{aligned}$$

Exemple : Si le mot envoyé est 1101, la probabilité que le mot reçu soit 0111 est

$$\begin{aligned} P(\text{message reçu}=0111) &= P(\text{vecteur d'erreur}=1010) \\ &= P(1^{\text{er}} \text{ bit erroné}) \times P(2^{\text{e}} \text{ bit correct}) \\ &\quad \times P(3^{\text{e}} \text{ bit erroné}) \times P(4^{\text{e}} \text{ bit correct}) \\ &= p^2(1-p)^2 \end{aligned}$$

Soit w le mot du code envoyé. Le mot reçu w' est corrigé en w si et seulement si w et w' sont dans la même colonne. Or par construction du tableau standard, w et w' sont dans la même colonne si et seulement si leur somme $w + w'$ est dans la première colonne du tableau.

La correction fonctionne donc quand le vecteur d'erreur est dans la première colonne.

Proposition. *On note a_i les mots de la 1ère colonne du tableau standard, w le mot du code envoyé, w' le message reçu et $e = w + w'$ le vecteur d'erreur. Alors*

$$P(w' \text{ corrigé en } w) = P(e \text{ est dans la } 1^{re} \text{ colonne}) = \sum_{i=0}^{2^{n-p}} P(e{=}a_i) = \sum_{i=0}^{2^{n-p}} p^{p(a_i)}\,(1-p)^{n-p(a_i)}$$

Application :
On reprend le tableau standard construit précédemment. La probabilité que le décodage soit correct est

$$P(e{=}0000) + P(e{=}1000) + P(e{=}0100) + P(e{=}0010) = (1-p)^4 + 3\,p(1-p)^3$$

Par exemple avec $p = 10^{-4}$, la probabilité que le décodage soit correct est environ 0,9999.

La méthode de correction par le tableau standard présente plusieurs inconvénients :

- Le tableau est long à construire.
- Dès que la taille des blocs est relativement importante ($n \geq 30$ environ), le tableau devient beaucoup trop gros pour être utilisable.
- La recherche du message reçu dans le tableau est lente.

On va voir une deuxième méthode, plus algébrique, de correction des messages.

8.4 Correction d'erreurs : syndrômes

8.4.1 Un exemple de détection d'erreurs

On utilise pour la transmission un codage linéaire systématique de taille $(6, 3)$, donné par la matrice génératrice standard $\begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$.

On reçoit le message 011111. Comment savoir si ce message est correct ?

Le codage est systématique, donc si ce message est correct, le mot d'origine était 011, et on devrait retrouver le message reçu en codant 011. On compare donc le message reçu 011111 avec celui obtenu en codant les trois premiers bits $011 \mapsto 011110$

Seule la comparaison des trois bits de contrôle est pertinente. On a trouvé 110 à la place des bits reçus 111, l'erreur entre les deux est $110 + 111 = 001 \neq 000$, donc le message reçu n'est pas correct.

Cette opération

- garder les p premiers bits du message reçu, les recoder,
- prendre les $n-p$ bits de contrôle du résultat et les additionner aux bits de contrôles du message reçu,

— comparer le résultat à 0,

revient à faire le produit du message reçu avec la matrice $H = \begin{pmatrix} P \\ I_{n-p} \end{pmatrix}$, où P est la matrice de parité du codage.

Dans l'exemple précédent, on a $P = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$, donc $H = \left(\begin{array}{ccc} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ \hline 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right)$.

On vérifie qu'on a bien $011111 \times H = 001$.

8.4.2 Matrice de contrôle et syndrômes

Définition.
Soit C un code linéaire de taille (n, p) ayant une matrice génératrice standard $G = (I_p\, P)$, P étant la matrice de parité.
La matrice de contrôle *du code C est la matrice de taille $n \times (n-p)$:*

$$H = \begin{pmatrix} P \\ I_{n-p} \end{pmatrix}.$$

Définition. *Soit m un mot binaire de taille n. On appelle* syndrôme *de m, et on note $\sigma(m)$, le produit mH. C'est un mot de longueur $k = n - p$.*

Le syndrôme correspond à la somme des bits de contrôles du message reçu et des bits de contrôle recalculés.

Exemples :

— Codage $(3, 1)$ par répétition : $G = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$, donc $H = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$. Exemple de calcul de syndrôme : $\sigma(101) = 10$, $\sigma(111) = 00$.

— Codage $(4, 3)$ par parité : $G = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$, donc $H = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$. Exemple de calcul de syndrôme : $\sigma(1011) = 1$, $\sigma(1001) = 0$.

— Autre code : $G = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}$, donc $H = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$. Exemple de calcul de syndrôme : $\sigma(01010) = 00$, $\sigma(11111) = 11$.

8.4.3 Matrice de contrôle et détection d'erreurs

Théorème. *Soient C un code linéaire et H sa matrice de contrôle. Un message m est un mot du code si et seulement si son syndrôme est nul :*

$$m \in C \iff mH = 0_{n-p}$$

La matrice de contrôle fournit donc un moyen efficace de détection des erreurs.

Démonstration. Soit $m \in \mathbb{F}_2^n$ un mot de longueur n. On note $m_I \in \mathbb{F}_2^p$ le mot formé des p premiers bits de m (bits d'informations), et $m_C \in \mathbb{F}_2^k$ le mot formé des $k = n - p$ dernier bits de m (bits de contrôle) : $m = m_I.m_C$ (concaténation).

On utilise le fait que H s'écrit $\begin{pmatrix} P \\ I_{n-p} \end{pmatrix}$.

$$\begin{aligned} mH = 0_{n-p} &\Leftrightarrow (m_I.m_C) \times \begin{pmatrix} P \\ I_{n-p} \end{pmatrix} = 0_k \\ &\Leftrightarrow m_I \times P + m_C = 0_k \\ &\Leftrightarrow m_C = m_I \times P \\ &\Leftrightarrow m_I.m_C = m_I \times (I_p\, P) \\ &\Leftrightarrow m = m_I \times G \\ &\Leftrightarrow m \in C \end{aligned}$$

□

Applications : Soit C le code linéaire donné par sa matrice génératrice standard

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Les mots $m_1 = 010101010$, $m_2 = 111111111$, $m_3 = 011010000$ font-ils partie du code ?

La matrice de contrôle est $H = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$.

On calcule les syndrômes :

$m_1 H = 1100$,

$m_2H = 0110$,

$m_3H = 0000$,

donc seul m_3 fait partie du code.

8.4.4 Syndrômes et classes d'équivalence du code

On rappelle la définition de la relation d'équivalence $\mathcal{C}$ associée à un code linéaire C :

$$u\,\mathcal{C}v \iff u+v \in C$$

Théorème. *Deux messages sont équivalents pour la relation $\mathcal{C}$ si et seulement si ils ont le même syndrôme :*

$$u\,\mathcal{C}v \iff \sigma(u) = \sigma(v)$$

Démonstration. $u\,\mathcal{C}v \Leftrightarrow u+v \in C \Leftrightarrow \sigma(u+v) = 0_{n-p} \Leftrightarrow \sigma(u)+\sigma(v) = 0_{n-p} \Leftrightarrow \sigma(u) = \sigma(v)$ □

8.4.5 Liste des syndrômes

On a vu que le principe de la correction d'erreurs pour un code linéaire consiste à déterminer, dans chaque classe d'équivalence du code, un mot de plus petit poids (le vecteur d'erreurs).

On va utiliser le fait que les classes d'équivalences du code sont en bijection avec l'ensemble des syndrômes possibles pour construire un tableau, appelé *liste des syndrômes*, associant à chaque syndrôme possible un mot de plus petit poids ayant ce syndrôme.

Méthode de construction de la liste des syndrômes :
On se donne un code linéaire C, de taille (n, p), dont on connaît une matrice de contrôle H.

- On commence par écrire dans une première colonne tous les syndrômes possibles, c'est-à-dire tous les mots de longueur $k = n - p$.
- On parcourt ensuite l'ensemble des mots de longueur n de poids 0, 1, puis 2, 3 etc. Pour chaque mot on calcule son syndrôme. Si c'est la première fois que ce syndrôme apparaît, on écrit le mot dans la deuxième colonne, en face de son syndrôme.
- On continue jusqu'à ce qu'il y ait un mot en face de chaque syndrôme.

Exemple : Soit C le code linéaire de taille $(4, 2)$, de matrice génératrice standard $G = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$ (même code que dans l'exemple du tableau standard).

La matrice de contrôle est $H = \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$

Construction de la liste des syndrômes :

syndrômes	vecteurs d'erreurs
00	0000
01	0001
10	0010
11	1000

$\sigma(0000) = 00$

$\sigma(0001) = 01$

$\sigma(0010) = 10$

$\sigma(0100) = 01$, déjà présent

$\sigma(1000) = 11$

8.4.6 Utilisation de la liste des syndrômes :

Soit w' le mot reçu. On calcule son syndrôme $\sigma(w')$.

On regarde dans la liste le vecteur d'erreurs e correspondant à ce syndrôme, puis on corrige le mot reçu en le remplaçant par $w = w' + e$.

Par construction de la liste, e est un élément de poids minimum dans la classe d'équivalence de w'. Donc w est bien un mot du code le plus proche de w', ce qui justifie cette méthode de correction.

Remarque : la liste des syndrômes n'est pas unique. En particulier, si plusieurs mots de plus petit poids ont le même syndrôme, le choix de celui qui apparaît dans le tableau aura une influence sur la correction.

Exemple : On avait obtenu la liste des syndrômes suivante :

syndrômes	vecteurs d'erreurs
00	0000
01	0001
10	0010
11	1000

avec la matrice de contrôle $H = \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$

Si on reçoit le mot 0111 : $\sigma(0111) = 10$,
on le corrige en $0111 + 0010 = 0101$.

Si on reçoit le mot 0110 : $\sigma(0110) = 11$,
on le corrige en $0110 + 1000 = 1110$.

Remarques :

— Attention : pour la correction, il faut disposer de la liste des syndrômes *et* de la matrice de contrôle (pour calculer les syndrômes).

— La correction par syndrôme est un peu plus compliquée mais plus rapide et moins encombrante que la correction par tableau standard : elle est donc à privilégier.

— La construction de la liste des syndrômes peut quand même être longue et fastidieuse. Pour certains codes linéaires particuliers, il existe des méthodes de correction plus rapides.

8.5 Codes de Hamming

8.5.1 Matrice de contrôle et distance minimale du code

La matrice de contrôle donne directement des informations sur la capacité de correction d'un code linéaire.

Théorème. *Soit C un code linéaire, de distance minimale $d(C)$, et ayant une matrice de contrôle H. On a $d(C) \geq 3$ si et seulement si les lignes de H sont toutes distinctes et non nulles.*

On se servira de ce résultat pour construire des codes linéaires pouvant corriger une erreur.

Démonstration. On remarque que le syndrôme d'un mot de poids 1, 0...010...0, ayant exactement un 1 en i-ème position, est la i-ème ligne de H : les lignes de la matrice de contrôle correspondent aux syndrômes des mots de poids 1.
Donc une ligne de H est nulle si et seulement si il existe un mot de poids 1 de syndrôme nul, c'est-à-dire un mot du code de poids 1, ce qui équivaut à $d(C) = 1$.

De la même façon, les syndrômes des mots de poids 2 correspondent aux sommes de deux lignes distinctes de H.
Deux lignes de H sont identiques si et seulement si leur somme est nulle, donc si et seulement si il existe un mot de poids 2 de syndrôme nul, c'est-à-dire un mot du code de poids 2. □

8.5.2 Construction de codes linéaires

Pour construire un code linéaire C de taille (n, p) pouvant corriger au moins une erreur (i.e. $d(C) \geq 3$) :

- on construit la matrice de contrôle H de taille $n \times k$ (avec $k = n - p$), en commençant par l'identité en bas, puis en ajoutant des lignes non nulles toutes différentes (possible uniquement si $n \leq 2^k - 1$)
- connaissant $H = \begin{pmatrix} P \\ I_{n-p} \end{pmatrix}$, on connaît la matrice de parité P, et donc la matrice génératrice standard $G = (I_p P)$ du code.

8.5.3 Codes de Hamming

Définition. *Un* code de Hamming *est un code linéaire dont la matrice de contrôle H est constitué de tous les mots binaires non nuls de longueur k.*
C'est un code parfait, de taille $(2^k - 1, 2^k - 1 - k)$, et de distance 3 (exercice !).

Les codes de Hamming sont simples à construire et permettent de corriger exactement une erreur, ce qui justifie leur utilité.

Exemples de codes de Hamming :

— Si $k = 2$: alors $H = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$, donc $G = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$.
On retrouve le code $(3, 1)$ par répétition : c'est le plus petit code de Hamming.

— Si $k = 3$: on construit $H = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ par exemple, donc $G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$.
C'est un code de Hamming de taille $(7, 4)$.

9 Codes polynomiaux

On a vu qu'avec les codes binaires linéaires on disposait d'un outil efficace pour la correction d'erreurs. On peut cependant trouver que manipuler des matrices n'est pas toujours très pratique, surtout si leurs dimensions sont un peu importantes. On va voir qu'en utilisant un peu d'arithmétique sur les polynômes, on peut définir des codes linéaires particuliers, les codes polynomiaux, qui permettent de s'affranchir (presque) complètement du calcul matriciel, et qui sont entièrement déterminés par la donnée d'un simple polynôme.

9.1 Polynômes, division euclidienne

9.1.1 Polynômes à coefficients dans $\mathbb{F}_2$

Un *polynôme* à coefficient dans $\mathbb{F}_2$ se définit de la même façon qu'un polynôme réel : c'est une expression de la forme

$$a_n X^n + a_{n-1} X^{n-1} + \ldots + a_1 X + a_0$$

où a_n, a_{n-1}, …, a_1, a_0 sont des éléments de $\mathbb{F}_2$.

On note $\mathbb{F}_2[X]$ l'ensemble des polynômes à coefficients dans $\mathbb{F}_2$
On note $\mathbb{F}_{2,n}[X]$ l'ensemble des polynômes à coefficients dans $\mathbb{F}_2$, de degré *strictement inférieur* à n.

L'addition et la multiplication de polynômes se font comme dans le cas réel, en utilisant les règles de calcul de $\mathbb{F}_2$.

Exemples :

— $P(X) = X^4 + X^2 + 1 = 1.X^4 + 0.X^3 + 1.X^2 + 0.X^1 + 1.X^0$

— $Q(X) = X^5 + X^3 + X^2 + 1 = 1.X^5 + 0.X^4 + 1.X^3 + 1.X^2 + 0.X^1 + 1.X^0$

— $P(X) + Q(X) = (X^4 + \cancel{X^2} + \cancel{1}) + (X^5 + X^3 + \cancel{X^2} + \cancel{1}) = X^5 + X^4 + X^3$

— $P(X)Q(X) = (X^4 + X^2 + 1)(X^5 + X^3 + X^2 + 1)$
$= (X^9 + \cancel{X^7} + X^6 + \cancel{X^4}) + (\cancel{X^7} + \cancel{X^5} + \cancel{X^4} + \cancel{X^2}) + (\cancel{X^5} + X^3 + \cancel{X^2} + 1)$
$= X^9 + X^6 + X^3 + 1$

On définit le *degré* d'un polynôme non nul comme dans le cas réel : c'est l'entier correspondant à la plus grande puissance de X ayant un coefficient non nul.

Exemples : $d^\circ(X^4+X^2+1)=4$, $d^\circ(X^5+X^3+X^2+1)=5$, $d^\circ(X)=1$, $d^\circ(1)=0$.

Le degré vérifie les deux propriétés suivantes :
- $d^\circ(P_1P_2)=d^\circ(P_1)+d^\circ(P_2)$
- $d^\circ(P_1+P_2)\leq max(d^\circ(P_1),d^\circ(P_2))$

9.1.2 Rappels sur la division euclidienne d'entiers

Exemple de division "à la main", avec quotient et reste :

	2	7	5	3	4	8	27
−	2	7					10198
		0	5				
		−	0				
			5	3			
		−	2	7			
			2	6	4		
		−	2	4	3		
				2	1	8	
			−	2	1	6	
						2	

$\Rightarrow 275348 = 27 \times 10198 + 2$

Théorème. *Soient a, $b\in\mathbb{Z}$ deux entiers. Alors il existe un unique couple (q,r), $q\in\mathbb{Z}$, $0\leq r<|b|$, tel que $a=b\times q+r$.*
On appelle q le quotient *et r le* reste *de la division euclidienne de a par b.*

9.1.3 Division euclidienne dans $\mathbb{R}[X]$

On peut procéder de la même façon avec des polynômes. On procède en éliminant à chaque étape la plus grande puissance du dividande.

Exemple :

$3X^5$	$+2X^4$	$-X^3$		$-7X$	$+5$	X^2-X+2
$3X^5$	$-3X^4$	$+6X^3$				$3X^3+5X^2-2X-12$
	$5X^4$	$-7X^3$		$-7X$	$+5$	
	$5X^4$	$-5X^3$	$+10X^2$			
		$-2X^3$	$-10X^2$	$-7X$	$+5$	
		$-2X^3$	$+2X^2$	$-4X$		
			$-12X^2$	$-3X$	$+5$	
			$-12X^2$	$+12X$	-24	
				$-15X$	$+29$	

$\Rightarrow 3X^5+2X^4-X^3-7X+5=(X^2-X+2)(3X^3+5X^2-2X-12)-15X+29$

Théorème. *Soient $P_1(X)$ et $P_2(X)$ deux polynômes à coefficients réels.*
Il existe un unique couple $(Q(X), R(X)) \in \mathbb{R}[X] \times \mathbb{R}[X]$, avec $d^\circ R(X) < d^\circ P_2(X)$, tel que $P_1(X) = P_2(X)Q(X) + R(X)$.

9.1.4 Division euclidienne dans $\mathbb{F}_2[X]$

Cet énoncé reste vrai dans $\mathbb{F}_2[X]$:

Théorème. *Soient $P_1(X)$ et $P_2(X)$ deux polynômes à coefficients dans $\mathbb{F}_2$.*
Il existe un unique couple $(Q(X), R(X)) \in \mathbb{F}_2[X] \times \mathbb{F}_2[X]$, avec $d^\circ R(X) < d^\circ P_2(X)$, tel que $P_1(X) = P_2(X)Q(X) + R(X)$.
On appelle $Q(X)$ le quotient *et $R(X)$ le* reste *de la division euclidienne de $P_1(X)$ par $P_2(X)$.*

La division euclidienne se fait exactement pareil dans $\mathbb{R}[X]$ et dans $\mathbb{F}_2[X]$.

Exemple : Soient $P_1(X) = X^4 + X^3 + X^2 + 1$ et $P_2(X) = X^2 + 1$

X^4	$+X^3$	$+X^2$		$+1$	X^2+1
X^4		$+X^2$			X^2+X
	X^3			$+1$	
	X^3		X		
			X	$+1$	

$\Rightarrow P_1(X) = P_2(X)Q(X) + R(X)$, avec $Q(X) = X^2 + X$ et $R(X) = X + 1$.

Si le reste de la division euclidienne de P_1 par P_2 est nul, c'est-à-dire si $P_1(X) = P_2(X)Q(X)$, on dit que P_1 est un *multiple* de P_2, ou que P_2 *divise* P_1, et on note $P_2|P_1$ (lire : "P_2 divise P_1").

9.1.5 Mots et polynômes

Dans toute la suite du cours, on va identifier $\mathbb{F}_2^n$ (les mots binaires de longueur n) et $\mathbb{F}_{2,n}[X]$ (les polynômes de degré strictement inférieur à n à coefficients dans $\mathbb{F}_2$). On associe à un polynôme la suite de ses coefficients, par ordre *décroissant* des puissances de X :

$$P = \sum_{i=0}^{n-1} a_i X^i \iff w = a_{n-1}a_{n-2}...a_1a_0$$

Exemples : (avec $n = 6$)

$101101 \iff 1.X^5 + 0.X^4 + 1.X^3 + 1.X^2 + 0.X^1 + 1.X^0 = X^5 + X^3 + X^2 + 1$

$X^4 + X^2 + X = 0.X^5 + 1.X^4 + 0.X^3 + 1.X^2 + 1.X^1 + 0.X^0 \iff 010110$

9.2 Codes polynomiaux

9.2.1 Définition et construction

Définition. *Un sous-ensemble C de $\mathbb{F}_{2,n}[X]$ est un* code polynomial *si il existe un polynôme $g \in \mathbb{F}_{2,n}[X]$, appelé* polynôme générateur *de C, tel que les éléments de C sont exactement les polynômes de $\mathbb{F}_{2,n}[X]$ qui sont multiples de g :*

$$C = \{h \in \mathbb{F}_{2,n}[X] \ ; \ g|h\}$$

Etant donné un polynôme $g \in \mathbb{F}_{2,n}[X]$, le code polynomial de polynôme générateur g (on dit aussi *engendré* par g) se construit en regardant l'ensemble des multiples de g, de degré strictement inférieur à n.

Exemple : Pour $n = 5$, le code polynomial engendré par $X^2 + X + 1$ contient les mots suivants :

$$\begin{array}{rcrcll}
0 & \times & (X^2+X+1) & = & 0 & \longrightarrow \ 00000 \\
1 & \times & (X^2+X+1) & = & X^2+X+1 & \longrightarrow \ 00111 \\
X & \times & (X^2+X+1) & = & X^3+X^2+X & \longrightarrow \ 01110 \\
(X+1) & \times & (X^2+X+1) & = & X^3+1 & \longrightarrow \ 01001 \\
X^2 & \times & (X^2+X+1) & = & X^4+X^3+X^2 & \longrightarrow \ 11100 \\
(X^2+1) & \times & (X^2+X+1) & = & X^4+X^3+X+1 & \longrightarrow \ 11011 \\
(X^2+X) & \times & (X^2+X+1) & = & X^4+X & \longrightarrow \ 10010 \\
(X^2+X+1) & \times & (X^2+X+1) & = & X^4+X^2+1 & \longrightarrow \ 10101
\end{array}$$

Si le polynôme générateur g est de degré d, on obtient le code polynomial C engendré par g en multipliant g par tous les polynômes de degré strictement inférieur à $n-d$. Le code polynomial C est donc l'image de l'application

$$\begin{array}{rcl}
\phi : \mathbb{F}_{2,n-d}[X] & \to & \mathbb{F}_{2,n}[X] \\
P(X) & \mapsto & g(X)P(X)
\end{array}$$

C'est une application linéaire, donc C est un code *linéaire*, de taille $(n, n-d)$.

L'application ϕ est une application de codage simple, mais qui n'est *pas* systématique. On verra plus loin comment coder systématiquement un code polynomial.

9.2.2 Décodage

Soit C un code polynomial de polynôme générateur g. Un mot de $\mathbb{F}_2^n$ est un mot du code si et seulement si le polynôme correspondant est un multiple de g, c'est-à-dire si et seulement si *reste de sa division par g est nul.*

Exemple : les mots suivants : 1010101, 1111111, 0011001 font-ils partie du code polynomial $(7, 4)$ engendré par $X^3 + X + 1$?

On calcule les restes :

1010101 : $X^6 + X^4 + X^2 + 1 = (X^3 + X + 1)(X^3 + 1) + X^2 + X$, le reste n'est pas nul donc le mot n'appartient pas au code.

1111111 : $X^6 + X^5 + X^4 + X^3 + X^2 + X + 1 = (X^3 + X + 1)(X^3 + X^2 + 1)$, le reste est nul donc le mot appartient au code.

0011001 : $X^4 + X^3 + 1 = (X^3 + X + 1)(X + 1) + X^2$, le reste n'est pas nul donc le mot n'appartient pas au code.

En fait pour décoder un code polynomial on n'a pas besoin de déterminer une matrice de contrôle, car le calcul du syndrôme d'un mot revient au calcul du reste :

Proposition. *Soit C un code polynomial de taille $(n, n-d)$, de polynôme générateur g de degré d. Alors le syndrôme d'un mot $w \in \mathbb{F}_2^n$ s'obtient en calculant le reste de la division euclidienne par g du polynôme correspondant à w.*

On peut ensuite appliquer la méthode de la liste des syndrômes pour corriger les erreurs.

Exemple : en reprenant les calculs de l'exemple précédent, toujours pour le code polynomial $(7, 4)$ engendré par $X^3 + X + 1$, on obtient les syndrômes suivants :

$\sigma(1010101) = 110$ $(\leftrightarrow X^2 + X)$
$\sigma(1111111) = 000$ $(\leftrightarrow 0)$
$\sigma(0011001) = 100$ $(\leftrightarrow X^2)$

9.2.3 Codage systématique

Considérons un polynôme $g \in \mathbb{F}_{2,d}[X]$ et C le code polynomial de taille $(n, n-d)$ qu'il engendre ; on va voir comment construire un codage systématique de C.

Soit $w = a_1 a_2 ... a_{n-d}$ un mot de longueur d. Le polynôme correspondant est (attention à l'ordre des puissances !)

$$P_w(X) = a_1 X^{n-d-1} + a_2 X^{n-d-2} + \ldots + a_{n-d-1} X + a_{n-d} = \sum_{i=1}^{n-d} a_i X^{n-d-i}.$$

Pour coder systématiquement w, il faut lui rajouter d bits de contrôle : le mot codé de longueur n est de la forme $w' = a_1 a_2 ... a_{n-d}\, c_1 c_2 ... c_d$. Le polynôme correspondant est alors (attention aux puissances !)

$$\begin{aligned} P_{w'}(X) &= a_1 X^{n-1} + a_2 X^{n-2} + \ldots + a_{n-d-1} X^{d+1} + a_{n-d} X^d + c_1 X^{d-1} + c_2 X^{d-2} + \ldots + c_{d-1} X + c_d \\ &= X^d P_w(X) + \sum_{i=1}^{d} c_i X^{d-i}. \end{aligned}$$

Notons $C(X) = \sum_{i=1}^{d} c_i X^{d-i}$ le polynôme correspondant aux bits de contrôle. Le mot codé w' fait partie du code si et seulement c'est un multiple de g, c'est-à-dire si il existe $Q \in \mathbb{F}_{2,n-d}[X]$ tel que $P_{w'}(X) = g(X)Q(X)$, ce qui s'écrit encore

$$X^d P_w(X) = g(X)Q(X) - C(X).$$

Mais comme le degré de $C(X)$ est strictement inférieur à d, cette dernière égalité implique que $-C(X)$ est le reste de la division euclidienne de $X^d P_w(X)$ par $g(X)$!

Proposition. *Soit C un code polynomial de taille $(n, n-d)$ engendré par un polynôme $g \in \mathbb{F}_{2,d}[X]$. Pour coder un mot $w \in \mathbb{F}_2^{n-d}$:*
- *on écrit le polynôme P_w correspondant à w*
- *on calcule le reste $R(X)$ de la division euclidienne de $X^d P_w(X)$ par g*
- *les bits de $R(X)$ correspondent aux bits de contrôle*

Plus précisément, le mot codé est le mot correspondant au polynôme $X^d P_w(X) - R(X)$.

Exemple : On considère encore le code polynomial $(7, 4)$ engendré par $X^3 + X + 1$, et on veut coder le mot 0110. Le polynôme correspondant est $X^2 + X$. On fait la division euclidienne de $X^3(X^2 + X)$ par $X^3 + X + 1$:

$X^3(X^2 + X) = X^5 + X^4 = (X^3 + X + 1)(X^2 + X + 1) + 1$

Le reste est $R(X) = 1$, On calcule que $X^3(X^2 + X) - R(X) = X^5 + X^4 + 1$, le mot codé est alors 01100001.

On vérifie que le mot d'origine 0110 est bien préfixe du mot codé et que le mot codé fait bien partie de C.

9.2.4 Comment détecter si un code linéaire est polynomial ?

On peut remarquer dans un premier temps que si $g \in \mathbb{F}_{2,n}[X]$ engendre un code polynomial C, alors g est le polynôme de plus petit degré de C (puisqu'il divise tous les polynômes du code). En particulier, le mot associé à g est le mot du code commençant par le plus de zéros (soit exactement $n - \deg(g)$ zéros).

On suppose donné C est un code linéaire admettant une matrice standard. Pour savoir si le code est polynomial, on commence par choisir un candidat g pour le polynôme générateur en considérant le polynôme associé à la dernière ligne de la matrice standard, puis on vérifie que les autres polynômes de cette matrice sont des multiples du candidat g. Si tel est le cas, le code est bien polynomial.

Exemple :
Soit $G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$ la matrice génératrice d'un code linéaire C de taille $(6, 3)$. Ce code n'est pas polynomial puisque le polynôme $g(X) = X^3 + X^2 + X$ n'est pas un diviseur du polynôme associé à la deuxième ligne de la matrice $P(X) = X^4 + X^2 + 1$.

10 Codes cycliques

Une famille importante (dont fait partie le célèbre code de Reed-Solomon) est celle des codes cycliques. On détaille ici quelques propriétés de ces codes, ainsi que leur lien avec les codes polynomiaux.

Définition. *On appelle* fonction de décalage à gauche *la fonction suivante*

$$D : \{0;1\}^n \to \{0;1\}^n, (a_1 \dots a_n) \mapsto (a_2 \dots a_n a_1).$$

Par exemple, si $w = 01000111$ alors $Dw = 10001110$. On remarque aussi que D itérée n fois est égale à l'identité, en particulier la fonction D est bijective d'inverse $D^{-1} = D^{n-1} = D \circ \cdots \circ D$ (composée $n-1$ fois).

Définition. *Un code* cyclique *est un code linéaire qui est stable par décalage :*

$$\forall w \in C, Dw \in C.$$

Exemples :
- Le code trivial $C = \{0;1\}^n$ est un code cyclique.
- Le code $C = \{0000; 0101; 1010\} \subset \mathbb{F}_2^4$ est stable par décalage mais n'est pas cyclique (car non linéaire). Par contre le code $C = \{0000; 0101; 1010; 1111\}$ est bien un code cyclique.

Le lien entre les polynômes associés à un mot ainsi qu'à son mot décalé s'obtiennent aisément avec le calcul suivant :

Lemme. *Si w admet pour polynôme associé $P_w = a_1X^{n-1} + \cdots + a_n$, alors Dw admet pour polynôme associé*

$$P_{D_w} = a_2X^{n-1} + \cdots + a_nX + a_1 = XP_w - a_1(X^n - 1)$$

On peut déterminer facilement dans quels cas un code cyclique est polynomial. En particulier, on peut alors appliquer les méthodes de détections et corrections d'erreurs vues précédemment.

Théorème. *Soit $C \subset \mathbb{F}_2^n$ un code linéaire.*
Alors le code C est cyclique si et seulement si il est polynomial et que son polynôme générateur g est un diviseur de $X^n - 1$.

Démonstration.
- Supposons que C est un code polynomial de polynôme générateur g tel que $g|(X^n - 1)$. Soit $w = a_1 \dots a_n \in C$ de polynôme associé P_w, alors $P_{D_w} = XP_w - a_1(X^n - 1)$. Comme $g|P_w$ et $g|(X^n - 1)$, $g|P_{D_w}$ et D_w est bien un mot du code.
- Réciproquement, on suppose que C un code cyclique.
  Soit $w_g \in C$ le mot du code non nul commençant par le plus de zéros et g son polynôme associé de degré noté d. Alors w_g est nécessairement de la forme $0 \dots 01 * \cdots * 1$ avec $n-d$ symboles 0 en bits de poids forts et un symbole 1 en bit de poids faible ; en effet si le bit de poids faible de w_g était 0, alors D_{w_g} serait un mot du code qui commencerait par plus de zéros... Par conséquent

$$\begin{aligned} P_{Dw_g} &= XP_{w_g} - a_1(X^n - 1) = XP_w \\ P_{D^2w_g} &= XP_{Dw_g} - a_2(X^n - 1) = X^2P_w \\ &\vdots \\ P_{D^{n-d-1}w_g} &= XP_{w_g} - a_1(X^n - 1) = XP_{w_g} \end{aligned}$$

  représentent tous des mots du code. On montre alors que tout mot w dans le code est forcément divisible par g en considérant la division euclidienne du polynôme associé P_w par g :

$$P_w = gQ_w + R_w, \text{ avec } \deg(R_w) < d,$$

  comme $\deg(Q_w) \leq n - d - 1$, le mot associé à gQ_w est nécessairement dans le code d'après ce qui précède (comme combinaison linéaire des mots $Dw_g, D^2w_g, \dots, D^{n-d-1}w_g$ qui sont tous dans le code C). En particulier, comme g est le polynôme de C de plus petit degré d et que $R_w \in C$ est de degré strictement inférieur, on a nécessairement $R_w = 0$ et $g|P_w$.

Il ne reste plus qu'à vérifier que $g|(X^n - 1)$. Pour cela, on considère un mot du code w ayant un symbole 1 en bit de poids fort (un tel mot existe forcément, il suffit pour s'en convaincre de considérer $D^{-1}w_g$). Par conséquent, g divise le polynôme $P_w = X^{n-1} + a_2X^{n-2} + \cdots + a_n$ associé à w. On conclut alors facilement en remarquant que $P_{Dw} = XP_w - (X^n - 1)$ est divisible par g.

□