Théorie des codes

Compression, cryptage, correction

Jean-Guillaume Dumas
Professeur à l'université Grenoble Alpes

Jean-Louis Roch
Maitre de conférences à L'ENSIMAG

Éric Tannier
Chercheur de l'INRIA Rhône Alpes à l'université Lyon 1

Sébastien Varrette
Chercheur à l'université du Luxembourg

3e édition

DUNOD

Le **code international des signaux flottants maritimes** est un système mis en place dans toutes les marines du monde permettant de converser quelle que soit la langue parlée par le bâtiment origine du message et son (ou ses) destinataire(s). Il est composé des lettres de l'alphabet et de chiffres à l'aide de différents pavillons, flammes ou triangles, appelés *flottants*, et qui peuvent être utilisés de plusieurs manières :

- Chaque flottant peut représenter une lettre d'un message,
- Chaque flottant peut avoir une signification propre,
- Un ou plusieurs flottants peuvent former un mot codé (ou signal) qui peut être décodé à l'aide d'un document détenu par les deux bateaux.

© Dunod, Paris, 2007, 2018
11 rue Paul Bert, 92240
Malakoff
www.dunod.com
ISBN 978-2-10-078109-6

Préambule

Ce livre est un manuel pour étudiants en master, en mathématiques appliquées ou informatique. Il peut être utilisé comme référence par des enseignants, chercheurs ou ingénieurs, par des entreprises impliquées dans la télécommunication et la sécurité des informations.

Nous avons veillé à ce que cet ouvrage puisse être le plus indépendant possible, en particulier tous les termes utilisés sont définis, les notions de bases rappelées. Néanmoins ces rappels ne remplacent pas un cours complet d'algèbre, d'algorithmique ou de probabilités, et la maîtrise préalable de ces notions sera utile à la lecture.

Cet ouvrage a été initié au printemps 2000, avec la création du département ENSIMAG-ENSERG télécommunications à l'Institut national polytechnique de Grenoble (INPG) et la construction d'un cours général d'introduction aux codes et à leurs applications. Il a évolué en devenant un support de référence pour différents cours des universités de Grenoble, à la fois à l'INPG et à l'université Joseph Fourier, puis l'université Grenoble Alpes (UGA) pour paraître sous forme de livre chez Dunod en 2007, en version corrigée et augmentée en 2009, puis dans une nouvelle édition en 2013. Il a été traduit en anglais et est paru dans cette langue chez Wiley en 2015 sous le titre "*Foundations of Coding*".

Nous remercions chaleureusement nos collègues Gilles Debunne, Yves Denneulin, Dominique Duval, Grégory Mounié et Karim Samaké qui ont participé à ces enseignements et ont contribué par leurs commentaires à améliorer notre support de cours. Mouloud Aït-Kaci, Éric Bourre, Cécile Canovas-Dumas, Rodney Coleman, Mélanie Favre, Françoise Jung, Madeline Lambert, Benjamin Mathon, Clément Pernet, Marie-Aude Steineur, Antoine Taveneaux et Romain Xu ont également relu des parties de cet ouvrage et nous ont aidé à corriger plusieurs erreurs.

Pour la seconde, puis la présente édition, nous avons actualisé l'ouvrage en intégrant des techniques récentes, comme les courbes elliptiques, les codes à faible densité ou les codes-barres bidimensionnels, nous avons abordé l'utilisation de la cryptographie dans les applications mobiles (Facebook Messenger,

WhatsApp, Telegram) et les réseaux sociaux (Keybase), et nous avons élaboré des guides de bonnes pratiques de sécurité pour les utilitaires dorénavant incontournables comme GPG ou SSH. Nous avons également mis à jour plusieurs parties, incluant la stéganographie et le tatouage d'images, les attaques par saturations, la cryptographie post-quantique, les chiffrements homomorphes, les preuves à divulgation nulle de connaissance, le partage de secret et les codes d'interpolation, ou encore les standards récents comme le hachage de Galois, le portfolio e-stream et le standard de hachage SHA-3 (Keccak) ainsi que le règlement européen eIDAS.
Par ailleurs, 26 nouveaux exercices complètent les 120 exercices originaux et, pour plusieurs d'entre eux, des solutions interactives utilisant le logiciel libre de calcul mathématique Sage sont dorénavant disponibles sur le site internet compagnon du livre : `https://theoriedescodes.imag.fr`.

Grenoble, Lyon, Luxembourg, le 23 avril 2018.
Jean-Guillaume Dumas, Jean-Louis Roch, Éric Tannier, Sébastien Varrette.

Ce livre est accompagné d'un site internet :

`https://theoriedescodes.imag.fr`

Ce site donne plus de détails sur les auteurs, notamment des liens pour les contacter, leurs pages internet personnelles, etc.
Le sujet étant en constante évolution, on trouvera également sur le site une bibliographie plus détaillée.
Enfin, pour plusieurs des exercices de livre, la solution de fin d'ouvrage pourra s'accompagner d'une solution interactive avec le logiciel libre de calcul mathématique Sage (`http://www.sagemath.org`). La présence d'une telle solution interactive est repérée à la page de la solution par la mention du site internet, comme dans l'exemple : Exercice 4.43, page 303 (theoriedescodes.imag.fr/Ex/#4.43).
L'utilisateur pourra consulter les captures d'écran en pdf, ou utiliser directement le code source de chaque exercice, soit dans sa propre installation du logiciel Sage, soit au travers d'un serveur Sage de calcul collaboratif, comme `https://cocalc.com`.

Sommaire

Introduction

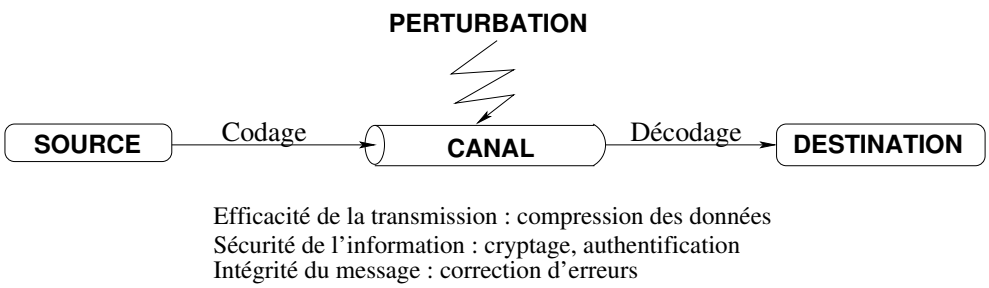

Figure 1 – Schéma fondamental du codage

Ce livre a pour objet la transmission automatique d'informations numériques, du point de vue de la structure de l'information elle-même, indépendamment du type de support de transmission. L'information peut être de n'importe quel type pourvu qu'on puisse en donner une représentation numérique : textes, images, sons, vidéos par exemple. La transmission de ces types de données est omniprésente dans la technologie, et spécialement dans les télécommunications. Il est donc nécessaire de s'appuyer sur de bonnes bases théoriques pour que cette transmission soit fiable, ce dernier terme se voyant donner plusieurs significations qui sont les objectifs qui nous guideront tout au long de l'ouvrage.

Les canaux de transmission peuvent également être de tous types (réseaux de câbles ou d'ondes), via éventuellement un support de stockage. Nous ne considérons pas les problèmes physiques que posent la transmission, qui font l'objet de la théorie dite « du signal ». La théorie des codes, elle, traite de la forme de l'information elle-même quand elle doit être transmise, ou stockée.

La communication d'une information commence par sa formulation par un émetteur, se poursuit par un transit via un canal, et se termine par la reconstitution du message par le destinataire. Parfois l'émetteur et le destinataire sont confondus, s'il s'agit d'un processeur ou d'une personne qui enregistre des

données dans un registre, une mémoire ou un disque puis les lit ultérieurement. L'information doit être reçue par son destinataire dans son intégralité, en sécurité, et le plus rapidement possible. Or quel que soit le canal considéré, avec des variations selon le support, on ne peut jamais le supposer « sûr », dans plusieurs sens du terme : il y a toujours des erreurs au cours des transmissions, et le message est susceptible d'être lu, voire altéré, par des tiers plus ou moins bien intentionnés. Il s'agit alors pour l'émetteur d'envoyer ses messages sous une forme qui permette au destinataire de les reconstituer en tenant compte d'éventuelles altérations au cours du transfert ou de la confidentialité de certaines informations, tout en minimisant la taille des données. Ces contraintes sont les points de départs de plusieurs champs d'étude en mathématiques et en informatique, souvent développés indépendamment les uns des autres bien qu'ils traitent des mêmes objets. Le but de ce livre est de les rassembler en un seul volume afin de présenter une seule théorie, dont l'objet est la forme donnée à l'information au cours de sa transmission, c'est-à-dire une théorie générale des *codes*.

En 1948, Claude Shannon posait la première pierre de ce qu'il a appelé une « théorie mathématique de l'information ». On raconte que sa théorie est née de réflexions sur la langue (l'anglais, en l'occurrence) : Shannon s'amusait à masquer une certaine proportion des textes qu'il lisait, et à les reconstituer à partir de la partie visible. En enlevant quelques mots, on pouvait reconstituer le sens de la phrase de façon certaine. C'est que ces mots cachés étaient *redondants*, ils n'apportaient rien au sens du message. Si on en enlevait trop, il était impossible de reconstituer le message avec certitude. Shannon mit alors au point une théorie qui serait à même de calculer la « quantité d'information » dans n'importe quel type de message, ce qui revient à déterminer son taux de redondance. Réduire cette redondance est ce qu'aujourd'hui nous appelons la *compression*. Par fidélité historique, la compression est parfois également appelée « théorie de l'information », bien que ce titre puisse convenir à l'ouvrage entier, qui lui-même dépasse largement cette théorie. Dans un souci d'efficacité, qui relève de l'optimisation de l'espace de stockage, et plus encore de la rapidité de transmission, il s'agit de rendre le message le plus court possible, en ne gardant que ce qui est indispensable à sa compréhension, ou mieux, en le reformulant sans redondance.

La confidentialité dont on veut entourer la transmission d'une information est une préoccupation beaucoup plus ancienne que celle de l'efficacité. Partant du principe que les routes (comme les canaux numériques) ne sont pas sûres, et qu'un message peut être intercepté au cours de sa transmission, il faut transformer le texte, le décorréler de sa signification, en ne laissant qu'à ses destinataires la clef de son déchiffrement. Les histoires de codes secrets, et des batailles entre inventeurs de codes et briseurs de codes (qui cherchent à

connaître la signification d'un message sans en avoir la clef) parsèment l'histoire des sociétés et des états. Shannon contribua également à ce domaine en apportant pour la première fois, en 1949, une preuve théorique de confidentialité. Une discipline scientifique est aujourd'hui consacrée aux codes secrets, la *cryptologie*. Non seulement les méthodes actuelles garantissent le *secret* sur la signification d'un message, mais elles permettent aussi de *signer* des documents ou d'*identifier* un émetteur.
Tous les canaux permettant de transmettre une information numérique peuvent être soumis à des perturbations qui sont susceptibles d'altérer une partie des messages, et donc d'en modifier le sens. Si les informations sont envoyées sans redondance, la plus petite altération peut en effet entraîner des fausses interprétations à l'arrivée. Dans le cas d'un texte en langue naturelle, la plupart des erreurs n'altéreront pas la perception du lecteur, car la redondance lui permettra de reconstituer le message original. Shannon, lui encore, a montré un résultat sensationnel en 1948 : même sur des canaux dont le taux d'erreur est très important, il est possible d'ajouter suffisamment de redondance pour que le message soit reçu dans son intégralité. Mais sa démonstration n'est pas constructive et ce théorème motive toujours la construction de méthodes incluant de façon ordonnée et optimisée de la redondance afin que le destinataire puisse soit s'apercevoir que le message a été altéré (*codes détecteurs*), soit corriger lui-même les erreurs éventuelles (*codes correcteurs*). Toutes ces méthodes sont paramétrables, c'est-à-dire adaptables au type de support considéré, et à son taux d'erreur.

Ce sont ces trois préoccupations — l'*efficacité*, la *sécurité*, l'*intégrité* — qui retiennent l'attention des concepteurs de méthodes de transmission de l'information. L'intérêt de les présenter ensemble repose sur leur objet commun, le *code*, qui structure l'information dans tous les supports technologiques actuels, et sur l'utilité de disposer dans un seul volume de toutes les façons de manipuler l'information avant de l'envoyer ou de la recevoir.
Le socle sur lequel s'appuie la théorie des codes est issu de l'algèbre linéaire, de l'arithmétique, des probabilités, de l'algorithmique et de la combinatoire. Les modèles mathématiques et les premiers développements algorithmiques qui structurent la notion de code sont introduits dans le premier chapitre de cet ouvrage. La présentation de ces modèles est assortie d'introductions des mathématiques utiles à leur manipulation, ainsi que de notions générales sur l'efficacité des méthodes de calcul, dont nous ferons un usage intensif tout au long de l'ouvrage. Il sera utile pour lire ce chapitre de disposer d'un bagage théorique de base en algèbre linéaire, ce qui rend ce livre accessible après une licence scientifique. Certains éléments dépassent les connaissances classiques des étudiants non mathématiciens, ils sont donc présentés en détails ici. Le lecteur se rendra rapidement compte de leur réelle efficacité pratique, le plus

souvent au moment même de leur introduction. Pour clarifier encore l'utilité et les dépendances entre les notions introduites, la figure 2 les représente.

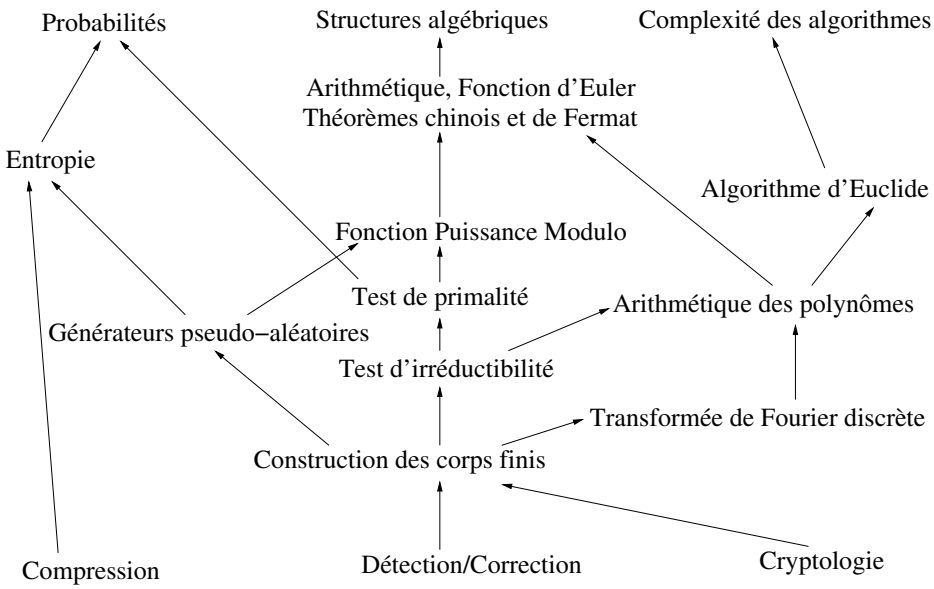

Figure 2 – Schéma des notions introduites dans le premier chapitre, avec leurs dépendances

Cette figure permettra aussi au lecteur pressé ou intéressé par une partie de l'ouvrage seulement de retrouver son chemin, la lecture linéaire n'étant pas obligatoire. Même si ce premier chapitre est conçu pour introduire la théorie des codes, il peut servir de boite à outils de référence lors de la lecture des chapitres suivants. Ceux-ci reprennent séparément la compression, la cryptographie, et les codes détecteurs et correcteurs d'erreur. Ils présentent les résultats théoriques fondamentaux et les algorithmes qui en découlent. Chacun de ces trois chapitres est illustré par des exemples d'utilisation pratique dans le contexte des télécommunications. Nous nous sommes efforcés de présenter à la fois les théories classiques du codage et les développements les plus récents sur le sujet, dans la mesure où un tel livre d'introduction le permettait.
En plus de réunir des théories mathématiques qui partagent le même objet, le credo de cet ouvrage est algorithmique. Ici, les propriétés mathématiques des fonctions servent à rendre efficaces leurs calculs. Les méthodes de calcul sont toujours données de façon détaillée, et immédiatement implémentables en n'importe quel langage de programmation. L'efficacité des méthodes est toujours calculée et discutée, les implémentations existantes comparées. Nos sciences sont à la fois les héritières des mathématiques grecques, qui faisaient

de l'esthétique de leurs énoncés leur principale qualité, et des mathématiques orientales, mues plus souvent par l'utilité et le calcul. C'est encore ce qui peut unir toutes les théories des codes, et c'est un de leurs grands mérites, de faire appel à des mathématiques rigoureuses et appréciées des esthètes pour construire des méthodes efficaces appliquées dans la communication courante. Ce livre est ainsi au confluent de ces fleuves, il attirera les férus de technologie autant que les amateurs de théorie des nombres, sans compter ceux que les histoires de déchiffrement de langages obscurs, de machines qui corrigent leurs propres erreurs et de codes secrets font toujours rêver.

Chapitre 1

Théorie des codes

On appellera un *code* une méthode de transformation qui convertit la représentation d'une information en une autre. Cette définition est suffisamment large pour accueillir plusieurs objets mathématiques (fonctions, algorithmes, transformées), dont nous ferons usage tout au long de ce texte. Le mot *code* servira aussi à désigner le produit de ces transformations, c'est-à-dire l'information codée et sa structure.
Mais pour commencer à se retrouver dans ces définitions et comprendre ce qu'est vraiment le code, voici un exemple simple mélangeant les technologies et les époques.

1.1 De Jules César à la télécopie

Pour introduire toutes les propriétés importantes des codes, en voici un construit à partir d'exemples réellement utilisés, actuellement ou par le passé. Supposons que nous voulions envoyer une information par fax tout en assurant le secret de la transmission. Voici ce que pourrait être un code qui réaliserait cette opération.

1.1.1 La source : de l'image à la suite de pixels

La transmission par fax permet de faire transiter des images par un canal téléphonique. Ce qu'on demandera à ce code sera d'assurer les transformations nécessaires pour obtenir à l'arrivée une image semblable à l'originale, rapidement et secrètement, dans un format qui puisse transiter par le canal.
Le premier procédé de transformation sera assuré par le scanner de l'appareil. Il consiste à lire l'image et à la transformer en une suite de pixels, qu'on peut se représenter comme des petits carrés soit noirs, soit blancs. C'est un code,

selon la définition que nous en avons donnée, et nous l'écrirons comme un algorithme, dans un format que nous adopterons tout au long de cet ouvrage.

Algorithme de codage

Entrées Une image
Sorties Une suite de pixels

L'entrée de l'algorithme est aussi appelée la *source*, et la sortie le *code*∗.
La méthode adoptée sera la suivante : la largeur de l'image source est divisée en 1728 parties égales ; la longueur est ensuite divisée en lignes, de façon à former des carrés de même taille. Ces carrés seront les pixels.
Chaque pixel se voit attribuer la couleur noire si à cet endroit l'image est suffisamment foncée, et la couleur blanche si l'image est claire. C'est la première partie de notre code.

1.1.2 La compression du message

Pour formaliser les messages sources et les codes, on définit le langage dans lequel ils sont exprimés. On appelle *alphabet* un ensemble fini $V = \{v_1, \ldots, v_k\}$ d'éléments appelés *caractères*. Le *cardinal* d'un ensemble fini V est son nombre d'éléments, noté $|V|$.
Une suite de caractères de V est une *chaîne*. On note V^* l'ensemble des chaînes sur V, et V^+ l'ensemble des chaînes de longueur non nulle. Comme l'alphabet du code et l'alphabet de la source peuvent différer, on parle d'*alphabet source* et d'*alphabet du code*. Par exemple, pour la méthode de compression que nous allons voir maintenant, l'alphabet source est le résultat du scanner, à savoir $\{pixel\ blanc, pixel\ noir\}$, et l'alphabet du code sera l'ensemble de bits $\{0, 1\}$. On pourrait envoyer la suite de pixels sous forme de bits numériques, puisqu'ils sont immédiatement traduisibles en 0 (pour un pixel blanc) et 1 (pour un pixel noir).
Mais on peut alors appliquer quelques principes simples à cette suite de 0 et de 1, afin de la compresser. Si on imagine que la page à envoyer ne comporte que quelques lignes de texte, et qu'une bonne partie est blanche, ce qui arrive très souvent dans les fax, alors n'y a-t-il pas de meilleur moyen de transmettre une suite de, mettons, 10000 zéros, que d'employer effectivement les 10000 zéros ? Bien sûr, il y a un meilleur moyen, et nous venons d'ailleurs de l'employer, car nous avons évoqué cette longue suite sans l'écrire explicitement. Il s'agit simplement d'indiquer que le code est composé de 10000 zéros plutôt que de

∗. Le mot *code* est très accueillant. Il désigne aussi bien l'ensemble des transformations que leur résultat, ainsi que, souvent, un programme informatique. Cette apparente confusion permet en fait de saisir les liens entre divers procédés mathématiques et informatiques, et c'est pourquoi nous conservons le terme dans ses multiples acceptions.

les écrire tous. Le code du fax réalise ce principe de compression ligne par ligne, c'est-à-dire par blocs de 1728 pixels. Pour une ligne, on peut décrire précisément l'algorithme de codage par l'algorithme 1.1.

Algorithme 1.1 Codage fax simplifié.

Entrées M, une suite de 1728 pixels (de $M[0]$ à $M[1727]$)
Sorties C, le message M compressé
  $n \leftarrow 1$ *// compteur des pixels consécutifs de même valeur*
  $C \leftarrow$« »*// chaîne vide*
  Pour i de 1 à 1727 **Faire**
    Si $M[i-1] = M[i]$ **Alors**
      $n \leftarrow n+1$ *// incrémente le nombre de pixels consécutifs de valeur* $M[i]$
    Sinon
      Concaténer n et la couleur du dernier pixel $M[i-1]$ à C
      $n \leftarrow 1$ *// ré-initialisation du compteur*
    Fin Si
  Fin Pour
  Concaténer n et la couleur du dernier pixel $M[1727]$ à C
  Renvoyer C

Ainsi, par exemple, une ligne entièrement blanche ne sera plus codée par une suite de 1728 zéros, mais par le code "1728 0", soit, dans un format numérique qui pourra être envoyé par le canal, "11011000000 0", puisque c'est la représentation de 1728 en binaire. On a donc remplacé un message de 1728 bits en un message de 12 bits, ce qui est nettement plus court. C'est donc un principe dit de *compression* des messages, appelé Run-Length Encoding (RLE), et que nous détaillerons dans la section 2.3.1. Nous nous sommes servis d'un caractère d'espacement dans notre représentation (entre 1728 et 0), pour une meilleure visibilité, mais ce caractère ne fait pas partie de l'alphabet du code. En pratique, il faudra donc le supprimer. Nous verrons les contraintes que cela implique un peu plus tard dans ce chapitre.

Exercice 1.1. *Une image pixelisée destinée à être faxée contient beaucoup de paires de pixels consécutifs "01".*
— *Que pensez-vous du code fax présenté plus haut ?*
— *Proposez un code plus efficace.* *Solution page 317.*

1.1.3 La détection d'erreurs

Tous nos lecteurs qui ont déjà utilisé un canal téléphonique ne seront pas étonnés d'apprendre que sa fiabilité n'est pas infinie. Tout message est susceptible d'être altéré, et il n'est pas impossible que si "11011000000 0" est envoyé sur

le canal numérique, “11010000000 0” (altération d'un bit) ou “1101000000 0” (destruction d'un bit) soit reçu. Habituellement, les canaux téléphoniques ont un taux d'erreurs variant de 10^{-4} à 10^{-7}, selon leur nature, ce qui veut dire qu'ils peuvent commettre en moyenne une erreur tous les 10000 bits. C'est loin d'être négligeable, quand on envoie de longs messages, et cela peut évidemment en altérer le sens. Pour une image, si le 1728 envoyé est devenu 1664 à cause de l'altération d'un seul bit, il va se produire un décalage de l'image à l'arrivée qui rendra le résultat inexploitable.
Le code du fax permet de détecter de telles erreurs de transmission. En cas d'erreur détectée sur une ligne, ceci permet de redemander la transmission de la même ligne pour avoir une confirmation, et comme il est très peu probable qu'une erreur se produise deux fois exactement au même endroit, le message pourra être corrigé.
Le principe de la détection d'erreurs dans le code du fax est le suivant : on fait précéder et suivre dans le message chaque ligne par une suite identifiable. Cela permet de synchroniser les bits et les lignes échangés et donc de détecter la perte de bits. Même si ce ne sont pas exactement ces nombres-là qui sont utilisés en pratique, mettons, pour expliquer le principe, que “0” est ajouté au début de chaque ligne, et “1729” est ajouté à la fin de chaque ligne.
On peut alors, pour chaque ligne reçue, vérifier qu'elle est de la forme “0 n_1 b_1 … n_k b_k 1729”, où n_i est un nombre entier qui donne le nombre de bits consécutifs, et b_i la couleur de ces bits. En particulier, on doit avoir $n_1 + \cdots + n_k = 1728$, et les b_i doivent être alternés. Ainsi, on repère facilement l'altération ou la perte d'un bit, dès que ce format n'est pas respecté.
Les principes de détection et correction d'erreurs, étudiés plus en détail au chapitre 4, reposent tous sur ce même principe : ajouter de l'information pour vérifier la cohérence du message reçu.

1.1.4 Le chiffrement

Supposons maintenant, après avoir compressé le message et apporté un format qui permette la détection d'erreurs, que nous voulions garder le message secret pour toute personne excepté son destinataire. Le canal téléphonique, comme la plupart des canaux, ne permet pas le secret en lui-même. Tout message qui y transite peut être facilement lu par un tiers. La mise en œuvre du secret consiste à transformer le message, le mettre sous une forme incompréhensible, et à le retransformer dans sa forme originelle à l'arrivée.
C'est une technique employée par les hommes depuis qu'ils communiquent. Dans les codes secrets employés dans l'antiquité, le secret résidait dans la technique mise en œuvre, qu'on appelle aujourd'hui *algorithme de chiffrement*. Ainsi, les historiens ont retrouvé des messages codés par les services

de Jules César. Les messages étaient des textes, et l'algorithme substituait chaque lettre du message initial M par celle située 3 positions plus loin dans l'alphabet. Pour les deux dernières lettres de l'alphabet, il fallait renvoyer aux deux premières. Par exemple le mot $TROYEN$ devenait $WURBHQ$. Ainsi, le texte n'avait plus de signification immédiate. C'est ce qu'on appelle un principe de *substitution mono-alphabétique*, car chaque lettre est remplacée par une autre, toujours la même, dans le message.
Si César avait voulu envoyer un fax, il aurait adapté son code au format numérique, ce qui aurait donné la fonction $f(x) = x + 3 \mod n$ pour tous les nombres envoyés sur le canal. Le nombre n est la taille de l'alphabet utilisé, ici par exemple 1730, puisqu'aucun nombre supérieur ne peut théoriquement être employé.
Ces fonctions de codage et décodage furent ensuite paramétrées par une simple clef K, un nombre entier choisi secrètement entre les communiquants. Cela revient à construire une fonction $f_K(x) = x + K \mod n$. Les spartiates utilisaient eux un algorithme de chiffrement totalement différent, le chiffrement par transposition : la scytale était un bâton sur lequel était enroulé un parchemin. Le message était écrit sur le parchemin enroulé mais le long du bâton et non le long du parchemin. Ainsi les lettres successives du message apparaissaient sur une circonvolution différente du parchemin. Pour le déchiffrer, le destinataire devait posséder un bâton d'un diamètre identique à celui utilisé pour l'encodage. D'autres systèmes cryptographiques, plus élaborés, virent alors le jour (chiffrements affines $f_{a,b}(x) = a.x + b \mod n$ - étudiés dans les exercices 1.2 et 3.1 ; chiffrements par substitution où chaque lettre est remplacée par un symbole d'un autre alphabet comme le code Morse etc.).

Exercice 1.2. *Marc-Antoine intercepte un message envoyé par César, crypté par chiffrement affine. Sans connaître la clef (a, b), comment décode-t-il le message ?* *Solution page 317.*

1.1.5 Le décodage

Nous avons vu dans ce qui précède toutes les techniques pour coder un message et en assurer l'efficacité de transmission et le secret. Cette description ne serait pas complète sans un mot pour présenter les principes de l'opération complémentaire, le *décodage*.

L'attaque et le déchiffrement

Le message arrive crypté. Il s'agit de le décoder. Pour qui ne possède pas la clef ou la méthode de chiffrement, on parle d'*attaque* sur le code, ou de *cassage* de code. La discipline qui consiste à inventer des méthodes d'attaque

pour briser les codes existants ou pour construire des codes plus résistants est la *cryptanalyse*. Pour le destinataire du message qui connaît la clef, il s'agit de *déchiffrement*. On parlera de l'attaque bientôt. Pour l'instant, la méthode de déchiffrement est très simple et consiste à appliquer la fonction inverse de la fonction de codage, éventuellement paramétrée par une clef, soit $f_K^{-1}(x) = x - K \mod n$.
Le message est alors déchiffré. Il reste à appliquer toutes les autres transformations en sens inverse. Tout d'abord, vérifier le format de chaque ligne pour détecter d'éventuelles erreurs (en cas d'erreur, redemander une émission), puis appliquer un algorithme de décodage, qui, étant donnée une suite de nombres, renverra la suite de pixels initiale. C'est ce qui est formalisé pour une ligne dans l'algorithme 1.2.

Algorithme 1.2 Décodage fax.

Entrées Une suite de nombres, sous la forme "0 n_1 b_1 ... n_k b_k 1729"
Sorties La suite des 1728 pixels correspondants
  Pour un indice i de 1 à k **Faire**
    Dessiner n_i pixels de couleur b_i
  Fin Pour

La décompression et les pertes

En appliquant cet algorithme pour toutes les lignes, on obtient une suite de pixels, qui est la même que celle de départ, en laquelle nous avions transformé l'image. Il ne reste plus qu'à restituer l'image. Ou plutôt une image semblable à l'image de départ. En effet, la seule information dont nous disposons est la valeur des pixels, et la méthode consiste donc à imprimer sur une feuille la suite de carrés noirs ou blancs correspondants. L'image de départ sera dégradée, puisque les carrés d'origine n'étaient pas entièrement noirs ou blancs, ce qui donne toujours aux fax un aspect peu esthétique, mais l'essentiel de l'information est conservé.
Quand, comme ici, l'information initiale n'est pas entièrement restituée, mais arrive sous une forme s'en rapprochant, on parle de codage *avec perte*. On utilise souvent des codes qui permettent un certain niveau de perte, quand on estime que l'information importante n'est pas altérée. C'est souvent le cas pour les informations audiovisuelles (voir la section 2.5).

1.1.6 Les défauts de ce code

Nous avons entièrement décrit un procédé de codage et son pendant, le décodage, en respectant les principales contraintes que nous rencontrerons en théorie

des codes (l'efficacité de la compression, la détection d'erreurs, le secret). Mais les principes utilisés dans ce code-là ne sont guère utilisables en pratique dans les communications numériques courantes, pour plusieurs raisons :

La compression : le principe RLE tel qu'il est appliqué ici a plusieurs défauts. D'abord, si le message est constitué de pixels blancs et noirs alternés, la taille du message « compressé » sera plus grande que la taille du message original. Le principe n'a donc aucune garantie de compression. De plus, inclure les bits b_i est quasiment inutile puisqu'ils sont alternés et que la valeur du premier suffirait à déterminer tous les autres. Le code du fax les élimine. Il devient alors une suite de nombres $n_1 \dots n_k$. Mais cette suite, codée en bits (chaque nombre a sa traduction binaire) est plus difficile à décoder. En effet, en recevant une suite de bits "1001011010010", comment savoir s'il s'agit d'un nombre n_i, ou de plusieurs nombres concaténés ? On peut par exemple coder tous les nombres sur le même nombre de bits pour résoudre ce problème, mais le principe de compression n'est alors pas optimal. Il oblige à coder "2" par "00000000010", ce qui augmente la taille du message. Nous verrons dans la suite de ce chapitre comment s'assurer de la déchiffrabilité, et dans le chapitre suivant comment en déduire de bons principes de compression.

La détection d'erreurs : ce principe oblige à demander un renvoi de l'information à chaque erreur détectée, alors qu'on pourra concevoir des codes qui corrigent automatiquement les erreurs du canal (chapitre 4). De plus, aucune garantie théorique ne vient assurer que ce codage est bien adapté au taux d'erreurs des canaux téléphoniques. On ne sait pas quel est le taux d'erreurs détecté par ce code, ni s'il pourrait accomplir la même performance en ajoutant moins d'information dans les messages. L'efficacité de la transmission peut dépendre de la qualité des principes de détection et de correction d'erreurs.

Le secret : le code de César est cassable par un débutant en cryptographie. Pour tous les principes de *substitution mono-alphabétique*, il est facile de décoder, même sans connaître la clef. Une cryptanalyse simple consiste à étudier la fréquence d'apparition des lettres au sein du texte chiffré et d'en déduire la correspondance avec les lettres du texte clair. La table 1.1 présente par exemple la répartition statistique des lettres dans le livre que vous avez entre les mains (les codes de mise en forme LaTeX inclus). Comme ce livre est assez long, on peut supposer que ces fréquences sont assez représentatives d'un texte scientifique français. Il est bien sûr possible d'avoir des tables de fréquences représentatives de textes littéraires français, scientifiques anglais, etc.

E	14.66%	I	7.83 %	T	7.56 %	N	7.37 %
R	6.78 %	S	6.68 %	A	6.40 %	O	5.88 %
L	5.02 %	C	4.77 %	D	4.70 %	U	4.64 %
M	3.38 %	P	3.20 %	F	1.91 %	B	1.64 %
G	1.64 %	X	1.49 %	H	1.25 %	Q	0.94 %
V	0.87 %	Y	0.52 %	K	0.36 %	J	0.19 %
Z	0.17 %	W	0.16 %				

Table 1.1 – Répartition des lettres dans ce manuscrit LaTeX.

Exercice 1.3. *Scipion trouve un papier sur lequel se trouve le message chiffré suivant :* `HJXFW FZWFNY JYJ HTSYJSY IJ ATZX!`

Aidez Scipion à déchiffrer ce message. *Solution page 317.*

1.1.7 Ordres de grandeur et bornes de complexité des algorithmes

Nous avons décrit un code et exposé ses qualités et ses défauts. Il est une qualité cruciale des codes que nous n'avons pas encore évoquée, celle de la rapidité à coder et décoder. Elle dépend de la vitesse des ordinateurs, et surtout de la complexité des algorithmes.

Taille des nombres

Nous considérons des nombres et leurs tailles, soit en chiffres décimaux, soit en bits. Ainsi, un nombre m possédera $\lceil \log_{10}(m) \rceil$ chiffres et $\lceil \log_2(m) \rceil$ bits. Pour fixer les idées, 128 bits font 39 chiffres décimaux ; 512 bits font 155 chiffres et 1024 bits sont représentables par 309 chiffres.

La vitesse des ordinateurs

Aujourd'hui n'importe quel PC est cadencé à au moins 1 GHz, c'est-à-dire qu'il effectue 1 milliard (10^9) d'opérations par seconde. À titre de comparaison, la vitesse la plus fantastique de l'univers est celle de la lumière : 300000 km/s $= 3.10^8$ m/s. Il ne faut que dix milliardièmes de seconde à la lumière pour traverser une pièce de 3 mètres ! Eh bien, pendant ce temps là, votre ordinateur a donc effectué 10 opérations ! ! ! On peut donc dire que *les ordinateurs actuels calculent à la vitesse de la lumière.*

Taille et âge de l'univers

Cette vitesse de calcul est véritablement astronomique ; pourtant la taille des nombres que l'on manipule reste énorme. En effet, rien que pour compter jusqu'à un nombre de 39 chiffres il faut énumérer 10^{39} nombres. Pour se convaincre à quel point c'est énorme, calculons l'âge de l'univers en secondes :

$$\text{âge Univers} \simeq 15 \text{ milliards années} \times 365,25 \times 24 \times 60 \times 60 \approx 5 \cdot 10^{17} \text{secondes}.$$

Ainsi, un ordinateur cadencé à 1 GHz mettrait plus de deux millions de fois l'âge de l'univers pour simplement compter jusqu'à un nombre de « seulement » 39 chiffres ! Quant à un nombre de 155 chiffres (ce qu'on utilise couramment en cryptographie), c'est tout simplement le carré du nombre d'électrons contenus dans l'univers : notre univers contiendrait environ $3 \cdot 10^{12}$ galaxies chacune renfermant grosso modo 200 milliards d'étoiles. Comme notre soleil pèse à la louche $2 \cdot 10^{30}$kg et qu'un électron ne pèse que $0,83 \cdot 10^{-30}$kg, on obtient :

$$\text{Univers} = (2 \cdot 10^{30} / 0,83 \cdot 10^{-30}) \times 200 \cdot 10^{9} \times 3 \cdot 10^{12} \approx 10^{84} \text{ électrons}.$$

Ce sont pourtant des nombres que nous aurons à manipuler. Leur taille sera l'un des défis algorithmiques que nous aurons à relever, en même temps qu'une assurance de secret de nos messages, si pour les décoder sans en posséder la clef, il faut des millions d'années de calcul mobilisant le milliard d'ordinateurs disponibles sur la Terre. Nous allons voir dans la suite comment construire des algorithmes capables de travailler avec de tels nombres entiers.

Complexité des algorithmes

Malgré la puissance des ordinateurs, un algorithme mal conçu pourra se révéler inexploitable. La *complexité* d'un algorithme est une mesure du temps que mettra un ordinateur, sur lequel on implémentera l'algorithme, pour terminer son calcul. La vitesse des ordinateurs augmentant constamment, une bonne mesure pour estimer ce temps est de compter le nombre d'opérations arithmétiques que nécessite l'algorithme. En général, par souci de simplification, il s'agit donc de compter seulement les quatre opérations classiques (addition, soustraction, multiplication, division) et parfois également des opérations binaires comme les décalages de bits. Bien sûr, ce nombre d'opérations dépend de la taille des nombres qu'on fournit en entrée, c'est pourquoi l'efficacité est toujours une fonction de la taille de l'entrée de l'algorithme. Pour pouvoir calculer cette taille dans tous les cas, on suppose que l'entrée est une suite de bits, et la taille de l'entrée est la longueur de cette suite. Ce qui suppose que pour calculer l'efficacité d'un algorithme, on doit concevoir un code qui transforme son entrée en suite de bits. C'est une opération souvent assez simple.

Fonction	Complexité	Exemple
$O(1)$	Constante	Accès à un élément dans une table
$O(\log n)$	Logarithmique	Recherche dans un tableau trié
$O(n)$	Linéaire	Recherche dans un tableau non trié
$O(n \log n)$	Quasi-linéaire	Tri d'un tableau
$O(n^k), k > 1$	Polynomiale	Multiplication de matrices
$O(k^n), k > 1$	Exponentielle	Énumération des parties d'un ensemble

Table 1.2 – Complexités classiques obtenues par l'analyse d'algorithmes.

Par exemple, pour coder un entier a, il faut de l'ordre de $\log_2(a)$ bits (on les écrit simplement en base 2). La taille d'un entier est donc son logarithme en base 2. La taille d'une suite de pixels noirs et blancs est la longueur de cette suite.
Comme il n'est souvent pas possible de compter exactement toutes les opérations réalisées lors de l'exécution d'un programme, on encadre la complexité d'un algorithme par des majorants et minorants asymptotiques en utilisant la notation dite de Landau. Si $k(n)$ est le nombre d'opérations arithmétiques effectuées par l'ordinateur qui exécute l'algorithme sur une entrée de taille n, et f une fonction quelconque, on dit que $k(n) = O(f(n))$ s'il existe une constante c telle que pour tout n assez grand, $k(n) \leqslant c \times f(n)$. On peut alors dire que la complexité de l'algorithme est au plus de l'ordre de $f(n)$. La table 1.2 fournit les complexités les plus communément rencontrées.

Exercice 1.4. *Quelle est la complexité du code fax, présenté section 1.1.2 ?*
Solution page 317.

Comme la notation de Landau permet de donner la complexité à une constante près, il est par exemple possible d'écrire $O(\log(n))$ sans donner la base du logarithme, puisque la base ne modifiera la fonction que d'une constante (le logarithme de la base).
En pratique tous les algorithmes devront présenter une borne de complexité linéaire $O(n)$, où n est la taille du message source, pour pouvoir prétendre à une efficacité « temps-réel », c'est-à-dire pour pouvoir être utilisé dans une transmission où un temps d'attente long n'est pas acceptable (téléphonie, audiovisuel...).
Une partie de la cryptographie repose sur l'hypothèse qu'il existe des problèmes pour lesquels on ne connaît aucun algorithme d'une telle complexité. Pour ces problèmes, tous les algorithmes connus nécessitent des temps de calcul astronomiques qui rendent leur exécution complète impossible.
Dans ce livre, nous considérons qu'un problème est impossible à résoudre (on emploie simplement parfois l'euphémisme « difficile ») si on ne connaît pas

d'algorithme qui le résolve en un temps humainement raisonnable, soit par exemple si sa résolution nécessiterait plus de 10^{50} opérations.

1.2 Codage par flot et probabilités

Afin d'obtenir des méthodes efficaces, c'est-à-dire linéaires en fonction de la taille du message, on peut considérer le message comme un *flot* de bits, c'est-à-dire une succession potentiellement infinie de caractères, qu'on codera un par un. Ainsi, la technique de transformation peut changer pour chaque symbole et une erreur sur le canal de communication n'est pas propagée par le codage. Cette technique est notamment utilisée en cryptographie (on parle alors de chiffrement par flot), et permet d'obtenir des messages dits *inconditionnellement sûrs*, *i.e.* pour lesquels la connaissance du message chiffré n'apporte aucune information sur le message clair. On parle alors de *chiffrement parfait*. Ainsi, la seule attaque possible est la recherche exhaustive de clef secrète. On utilise également le modèle du codage par flot pour construire des principes de détection et correction d'erreur (voir les codes convolutifs dans le chapitre 4). Le cryptosystème à *clef jetable* – en anglais *one-time-pad* – est un exemple de code par flot à visée cryptographique dont on peut prouver la sécurité inconditionnelle, moyennant l'introduction de quelques bases de probabilités et de théorie de l'information.

1.2.1 Vernam et le chiffrement à clef jetable (one-time-pad)

En 1917, pendant la première guerre mondiale, la compagnie américaine AT&T avait chargé le scientifique Gilbert Vernam d'inventer une méthode de chiffrement que les Allemands ne pourraient pas casser. Il proposa l'idée de combiner les caractères tapés sur une machine à écrire avec ceux d'une clef pré-définie. Dans les années 1920, Joseph O. Mauborgne, chef du service cryptographique de l'armée américaine suggera d'utiliser une clef unique et aléatoire. C'est la combinaison de ces deux idées qui amena le cryptosystème à clef jetable ou One-time-pad (OTP), le seul chiffrement connu à l'heure actuelle qui soit mathématiquement prouvé inconditionnellement sûr. Le système à clef jetable est un système cryptographique dit à *clef secrète*, c'est-à-dire que le secret réside dans un paramètre des fonctions de codage et de décodage connu uniquement de l'émetteur et du destinataire. C'est aussi le cas du chiffrement de César, dans lequel le paramètre est la taille du décalage des lettres ou des nombres.
Le chiffrement de Vernam se décrit formellement de la façon suivante : pour tout message clair M et une clef K de même longueur, le texte chiffré C est donné par :

$$C = M \oplus K,$$

ou $\oplus$ (qu'on note aussi xor) désigne l'opération logique « ou exclusif » bit à bit. Il s'agit donc d'une addition modulo 2 de chaque bit. C'est cette utilisation du ou exclusif qui a été brevetée par Vernam en 1919. Le déchiffrement s'effectue de la même façon que le chiffrement grâce à la propriété suivante :

$$C \oplus K = (M \oplus K) \oplus K = M$$

Si la clef K est totalement aléatoire, utilisée une seule fois et de même taille que M (*i.e.* $|K| = |M|$), on parle d'un chiffrement à clef jetable et un observateur tiers n'obtient aucune information sur le message clair s'il intercepte le message chiffré (hormis la taille de M). Cette propriété de sécurité inconditionnelle a été démontrée par Claude Shannon, nous y reviendrons dans la section 1.2.5.

Exercice 1.5. *Pourquoi faut-il jeter la clef après l'avoir utilisée,* i.e. *changer de clef pour chaque nouveau message ?* *Solution page 318.*

Exercice 1.6. *Construire un protocole, à base de clefs jetables, permettant de se connecter depuis un ordinateur quelconque sur internet à un serveur sécurisé. Le mot de passe doit être crypté pour ne pas circuler de façon visible sur le réseau ni être capturé par la machine utilisée.* *Solution page 318.*

Évidemment, il reste à formaliser ce que signifie générer un nombre aléatoire, donner des bons moyens de le faire et, pour prouver que le système est sûr, préciser ce que veut dire « obtenir de l'information » sur le message clair. Pour cela, nous introduisons maintenant les principes fondamentaux de la théorie de l'information, qui servent également de base aux principes de compression des messages.

1.2.2 Un peu de probabilités

Dans un système cryptographique, si on utilise une clef générée au hasard, toute déviance par rapport au « vrai » hasard sera un angle d'attaque pour la cryptanalyse. Le hasard a aussi sa part dans les méthodes de compression, car dès qu'il y a un ordre visible, une redondance, une organisation dans un message, non seulement les casseurs de codes vont s'en servir, mais les créateurs de codes vont y voir un moyen d'exprimer le message de façon plus dense. Mais qu'appelle-t-on une déviance par rapport au hasard, et plus simplement qu'appelle-t-on le hasard ? Par exemple, si les numéros “1 2 3 4 5 6” tombent au Loto, on doutera beaucoup qu'ils aient été vraiment générés au hasard, bien que *stricto sensu* cette combinaison ait autant de chance d'être générée que n'importe quelle autre. Si nous n'irons pas bien loin dans la philosophie, cette section apportera quelques moyens mathématiques d'aborder le hasard et ses effets, puis de créer quelque chose qui se rapprocherait du hasard.

Évènements et mesure de probabilité

Un *évènement* est le résultat possible d'une expérience aléatoire. Par exemple, si l'expérience est un jet de dé à six faces, l'obtention du nombre 6 est un évènement. Les opérateurs sur les ensembles ($\cup$, $\cap$, $\setminus$) sont utilisés pour les évènements (ils signifient *ou*, *et*, *sauf*).
On note Ω l'ensemble de tous les évènements possibles pour une expérience donnée. Une mesure de probabilité P est une application définie sur Ω, à valeur dans $[0, 1]$, qui vérifie :

1. $P(\Omega) = 1$ et $P(\emptyset) = 0$;
2. quels que soient A, B des évènements disjoints ($A \cap B = \emptyset$), $P(A \cup B) = P(A) + P(B)$.

Si l'ensemble des évènements est un ensemble discret ou fini, on parle de *probabilité discrète*.
Par exemple, si l'expérience aléatoire est le jet d'un dé à six faces non pipé, l'ensemble des évènements est $\{1, 2, 3, 4, 5, 6\}$, et la probabilité d'occurrence de chacun $1/6$.
L'ensemble des valeurs prises par la fonction de probabilité est la *distribution des probabilités*, ou *loi de probabilité*. Une distribution est dite *uniforme* si tous les évènements ont une même probabilité d'occurrence.
Le lemme dit de Gibbs est un résultat sur les distributions discrètes qui nous sera utile plusieurs fois dans cet ouvrage.

Lemme 1 (de Gibbs)**.** *Soient* $(p_1, \ldots, p_n)$, $(q_1, \ldots, q_n)$ *deux lois de probabilité discrètes telles que* $p_i, q_i > 0$ *pour tout* i*. Alors :*

$$\sum_{i=1}^{n} p_i * \log_2 \frac{q_i}{p_i} \leqslant 0 \ .$$

Preuve. Comme $\log_2(x) = \frac{\ln x}{\ln 2}$ et $\ln(2) > 0$, il suffit de prouver le lemme pour le logarithme népérien ln. Or pour tout $x \in \mathbb{R}_*^+$, $\ln(x) \leqslant x - 1$. En particulier, $\sum_{i=1}^{n} p_i * \ln \frac{q_i}{p_i} \leqslant \sum_{i=1}^{n} p_i * (\frac{q_i}{p_i} - 1)$. Le résultat se déduit du fait que $\sum_{i=1}^{n} p_i = \sum_{i=1}^{n} q_i = 1$. □

Probabilités conditionnelles et formule de Bayes

On dit que deux évènements sont *indépendants* si $P(A \cap B) = P(A)P(B)$. On appelle *probabilité conditionnelle* de l'évènement A par rapport à l'évènement B, la probabilité que A se produise, sachant que B s'est déjà produit. Elle est notée $P(A|B)$ et définie par :

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Par récurrence, on montre facilement que pour un ensemble d'évènements $A_1, \ldots, A_n$,

$$P(A_1 \cap \ldots \cap A_n) = P(A_1|A_2 \cap \ldots \cap A_n)P(A_2|A_3 \cap \ldots \cap A_n) \ldots P(A_{n-1}|A_n)P(A_n)$$

La formule dite de Bayes permet de calculer pour un ensemble d'évènements $A_1, \ldots, A_n, B$ les probabilités $P(A_k|B)$ en fonction des $P(B|A_k)$.

$$P(A_k|B) = \frac{P(A_k \cap B)}{P(B)} = \frac{P(B|A_k)P(A_k)}{\sum_i P(B|A_i)P(A_i)}$$

Exercice 1.7. *On propose le code secret suivant, permettant de coder deux caractères a et b avec trois clefs différentes k_1, k_2 et k_3 : si la clef est k_1 alors $a \to 1$ et $b \to 2$, si la clef est k_2 alors $a \to 2$ et $b \to 3$, sinon $a \to 3$ et $b \to 4$. On suppose en outre que l'on a des connaissances* a priori *sur le message M envoyé et la clef K utilisée : $P(M = a) = 1/4$; $P(M = b) = 3/4$ et $P(K = k_1) = 1/2$; $P(K = k_2) = P(K = k_3) = 1/4$.*
Quelles sont les probabilités d'obtenir les chiffres 1, 2 ou 3 ? Quelles sont les probabilités conditionnelles que le message soit a ou b sachant la valeur du chiffre ? Peut-on dire intuitivement si ce code secret est un « chiffrement parfait » ? *Solution page 318.*

Ces brefs rappels étant faits sur la théorie mathématique permettant d'étudier des évènements aléatoires, voyons ce que l'aléatoire signifie pour des ordinateurs.

1.2.3 Entropie

Source d'information

On appelle S l'alphabet du message source. Un message est alors un élément de S^+. Pour tout message, on peut calculer les fréquences d'apparition de chaque élément de l'alphabet, et construire ainsi une distribution de probabilités sur S. Une *source d'information* est constituée du couple $\mathcal{S} = (S, \mathcal{P})$ où $S = (s_1, \ldots, s_n)$ est l'alphabet source et $\mathcal{P} = (p_1, \ldots, p_n)$ est une distribution de probabilités sur S, c'est-à-dire que p_i est la probabilité d'occurrence de s_i dans une émission. On peut construire une source d'information à partir de n'importe quel message, en construisant la distribution de probabilités à partir de la fréquence des caractères dans le message ; on parle alors de *source induite* par le message.
La source d'information $\mathcal{S} = (S, \mathcal{P})$ est dite *sans mémoire* lorsque les évènements (occurrences d'un symbole dans une émission) sont indépendants et que leur probabilité reste stable au cours de l'émission (la source est *stationnaire*).

La source est dite *markovienne* si les probabilités d'occurrence des caractères dépendent des caractères émis précédemment. Dans le cas d'un seul prédécesseur, $\mathcal{P} = \{p_{ij}\}$, où p_{ij} est la probabilité d'occurrence de s_i sachant que s_j vient d'être émis. On a alors $p_i = \sum_j p_{ij}$.
Par exemple, un texte en français est une source, dont l'alphabet est l'ensemble des lettres latines et les probabilités d'occurrence sont les fréquences d'apparition de chaque caractère. Comme les probabilités dépendent dans ce cas fortement des caractères qui viennent d'être émis (un U est beaucoup plus probable après un Q qu'après un autre U), le modèle markovien sera plus adapté.
Une image induit aussi une source, dont les caractères de l'alphabet sont des niveaux de couleurs. Un son est une source dont l'alphabet est un ensemble de fréquences et d'intensités.
Une source $\mathcal{S}$ est dite *sans redondance* si sa distribution de probabilités est uniforme. Ce qui n'est évidemment pas le cas pour les messages courants, où certaines lettres ou certains mots sont beaucoup plus employés que d'autres. Ce sera l'angle d'attaque des méthodes de compression mais aussi des pirates qui chercheraient à lire un message sans y être autorisés.

Entropie d'une source

L'entropie est une notion fondamentale pour la manipulation d'un code. C'est en effet une mesure, à la fois de la quantité d'information qu'on peut attribuer à une source (ce qui sera utile pour la compression des messages), et du degré d'ordre et de redondance d'un message, ce qui est une information cruciale pour la cryptographie.
L'*entropie* d'une source $\mathcal{S} = (S, \mathcal{P})$, $S = (s_1, \ldots, s_n)$, $\mathcal{P} = (p_1, \ldots, p_n)$ est :

$$H(\mathcal{S}) = H(p_1, \ldots, p_n) = -\sum_{i=1}^{n} p_i \log_2(p_i) \ = \sum_{i=1}^{n} p_i \log_2(\frac{1}{p_i}) \ .$$

On désigne par extension l'entropie d'un message comme l'entropie de la source induite par ce message, la distribution de probabilités étant calculée à partir des fréquences d'apparition des caractères dans le message.

Propriété 1. *Soit $\mathcal{S} = (S, \mathcal{P})$ une source.*

$$0 \leqslant H(\mathcal{S}) \leqslant \log_2 n \ .$$

Preuve. En appliquant le lemme de Gibbs, page 19, à la distribution (q_1, …, q_n) = ($\frac{1}{n}$, …, $\frac{1}{n}$), on obtient $H(\mathcal{S}) \leqslant \log_2 n$, quelle que soit la source S. Enfin, la positivité est évidente puisque les probabilités p_i sont inférieures à 1. □

Remarquons que pour une distribution uniforme, l'entropie atteint donc son maximum. Elle baisse quand on s'éloigne de la distribution. C'est pour cette raison qu'on l'appelle « mesure du désordre », en supposant que le plus grand désordre est atteint par la distribution uniforme.

Exercice 1.8. *Quelle est l'entropie d'une source qui émet un caractère* 1 *avec une probabilité* 0.1 *et le caractère* 0 *avec une probabilité* 0.9 *? Pourquoi une faible entropie est-elle un bon augure pour la compression ? Solution page 318.*

Entropies conjointe et conditionnelle

On étend facilement la définition de l'entropie à plusieurs sources. Soient $\mathcal{S}_1 = (\mathcal{S}_1, P_1)$ et $\mathcal{S}_2 = (\mathcal{S}_2, P_2)$ deux sources sans mémoire, dont les évènements ne sont pas forcément indépendants. On note $\mathcal{S}_1 = (s_{11}, \ldots, s_{1n})$, $P_1 = (p_i)_{i=1..n}$, $\mathcal{S}_2 = (s_{21}, \ldots, s_{2m})$ et $P_2 = (p_j)_{j=1..m}$; puis $p_{i,j} = P(\mathcal{S}_1 = s_{1i} \cap \mathcal{S}_2 = s_{2j})$ la probabilité d'occurrence conjointe de s_{1i} et s_{2j} et $p_{i|j} = P(\mathcal{S}_1 = s_{1i} | \mathcal{S}_2 = s_{2j})$ la probabilité d'occurrence conditionnelle de s_{1i} et s_{2j}.
On appelle l'*entropie conjointe* de $\mathcal{S}_1$ et $\mathcal{S}_2$ la quantité

$$H(\mathcal{S}_1, \mathcal{S}_2) = -\sum_{i=1}^{n}\sum_{j=1}^{m} p_{i,j} \log_2(p_{i,j})$$

Par exemple, si les sources $\mathcal{S}_1$ et $\mathcal{S}_2$ sont indépendantes, alors $p_{i,j} = p_i p_j$ pour tous i, j et donc, dans ce cas, on montre facilement que $H(\mathcal{S}_1, \mathcal{S}_2) = H(\mathcal{S}_1) + H(\mathcal{S}_2)$.
Au contraire, il peut arriver, si les évènements de $\mathcal{S}_1$ et $\mathcal{S}_2$ ne sont pas indépendants, que l'on veuille connaître la quantité d'information contenue dans une source, connaissant un évènement de l'autre. On calcule alors l'*entropie conditionnelle* de $\mathcal{S}_1$ relativement à la valeur de $\mathcal{S}_2$, donnée par

$$H(\mathcal{S}_1 | \mathcal{S}_2 = s_{2j}) = -\sum_{i=1}^{n} p_{i|j} \log_2(p_{i|j})$$

Enfin, on étend cette notion à une entropie conditionnelle de $\mathcal{S}_1$ connaissant $\mathcal{S}_2$, qui est la quantité d'information restant dans $\mathcal{S}_1$ si la loi de $\mathcal{S}_2$ est connue :

$$H(\mathcal{S}_1 | \mathcal{S}_2) = \sum_{j=1}^{m} p_j H(\mathcal{S}_1 | \mathcal{S}_2 = s_{2j}) = \sum_{i,j} p_{i,j} \log_2 \left(\frac{p_j}{p_{i,j}} \right)$$

Cette notion est cruciale en cryptographie. En effet, il est très important que tous les messages cryptés aient une entropie forte, pour éviter que des traces d'organisation dans un message donnent des informations sur la manière dont il a été crypté. Mais il est aussi important que l'entropie reste forte si on arrive

à connaître des informations dépendantes, par exemple même la connaissance d'un message et de son cryptage ne devrait pas donner d'information sur la clef utilisée. On a alors les relations simples mais importantes suivantes :

$$H(\mathcal{S}_1) \geqslant H(\mathcal{S}_1|\mathcal{S}_2)$$

avec égalité si et seulement si $\mathcal{S}_1$ et $\mathcal{S}_2$ sont indépendantes ; et encore :

$$H(\mathcal{S}_1, \mathcal{S}_2) = H(\mathcal{S}_2) + H(\mathcal{S}_1|\mathcal{S}_2)$$

Mais l'entropie d'une source sans mémoire à elle seule ne capture pourtant pas tout l'ordre ou le désordre contenu dans un message. Par exemple, les messages "1 2 3 4 5 6 1 2 3 4 5 6" et "3 1 4 6 4 6 2 1 3 5 2 5" ont la même entropie, et pourtant le premier est suffisamment ordonné pour qu'on puisse le décrire par une formule, du type "1…6 1…6", ce qui n'est sans doute pas le cas du second. Pour prendre en compte ce type d'organisation, on fait appel aux extensions de sources.

Extension d'une source

Soit une source $\mathcal{S}$ sans mémoire. La $k^{ième}$ *extension* $\mathcal{S}^k$ de $\mathcal{S}$ est le doublet $(S^k, \mathcal{P}^k)$, où S^k est l'ensemble des mots de longueur k sur S, et $\mathcal{P}^k$ est la distribution de probabilité ainsi définie : pour un mot $s = s_{i_1} \dots s_{i_k} \in S^k$, alors $P^k(s) = P(s_{i_1} \dots s_{i_k}) = p_{i_1} \dots p_{i_k}$.

Exemple : $S = (s_1, s_2)$, $\mathcal{P} = (\frac{1}{4}, \frac{3}{4})$ alors $S^2 = (s_1s_1, s_1s_2, s_2s_1, s_2s_2)$ et $P^2 = (\frac{1}{16}, \frac{3}{16}, \frac{3}{16}, \frac{9}{16})$.
Si $\mathcal{S}$ est une source markovienne, on définit de la même façon S^k, et pour un mot $s = s_{i_1} \dots s_{i_k} \in S^k$, alors $P^k(s) = p_{i_1} p_{i_2 i_1} \dots p_{i_k i_{k-1}}$.

Propriété 2. *Soient $\mathcal{S}$ une source, et $\mathcal{S}^k$ sa $k^{ième}$ extension.*

$$H(\mathcal{S}^k) = kH(\mathcal{S})\,.$$

Autrement dit, cette propriété exprime que la quantité d'information d'une source étendue à k caractères est exactement k fois celle de la source originelle. Cela semble tout à fait naturel. Cependant, il convient de distinguer une source étendue à k caractères d'un message (un fichier par exemple) où les caractères seraient regroupés k par k. Il est en effet possible de dénombrer dans un fichier les occurrences des caractères pour en déduire leur distribution et donc l'entropie d'une source qui aurait les mêmes caractéristiques probabilistes. C'est d'ailleurs ce qui est utilisé pour compresser des fichiers comme nous le verrons dans le chapitre 2. Néanmoins, si le message est « étendu » au sens

où des groupes consécutifs de k caractères sont formés et que la distribution de ces groupes est évaluée, alors l'entropie de la source correspondant à cette « extension de message » est forcément inférieure à l'entropie correspondant au message initial, comme démontré ci-après.

Propriété 3. *Soient $\mathcal{M}$ un message de taille n, $\mathcal{S}_{\mathcal{M}^k}$ la source dont les probabilités correspondent aux occurrences des k-uplets consécutifs de M et $\mathcal{S}^k_{\mathcal{M}}$ la $k^{ième}$extension de la source induite $\mathcal{S}_{\mathcal{M}}$. Alors*

$$H(\mathcal{S}_{\mathcal{M}^k}) \leqslant H(\mathcal{S}^k_{\mathcal{M}})\,.$$

Preuve. Nous donnons les détails pour $k = 2$. Soient (q_i) les probabilités de la source induite $\mathcal{S}_{\mathcal{M}}$, $(q_{i,j} = q_i \cdot q_j)$ celles de la deuxième extension de source induite $\mathcal{S}^2_{\mathcal{M}}$ et $(p_{i,j})$ celles de la source induite par les couples d'éléments successifs de M, $\mathcal{S}_{\mathcal{M}^2}$. Alors le lemme de Gibbs, page 19, appliqué à $p_{i,j}$ et $q_{i,j}$ donne $\sum_{i,j} p_{i,j} \log_2 \left(\frac{q_i q_j}{p_{i,j}}\right) \leqslant 0$. Cela se réécrit

$$H(\mathcal{S}_{\mathcal{M}^2}) \leqslant -\sum_{i,j} p_{i,j} \log_2(q_i q_j) \tag{1.1}$$

Notons n_i le nombre d'apparitions du symbole i dans $\mathcal{M}$, et $n = |\mathcal{M}|$, alors $q_i = n_i/n$. Notons également $n_{i,j}$ le nombre d'occurences du couple (i, j) dans $\mathcal{M}^2$, alors $|\mathcal{M}^2| = n/2$ et $p_{i,j} = 2n_{i,j}/n$. On obtient également $n_i = \sum_j n_{i,j} + \sum_k n_{k,i}$ et donc $2q_i = \sum_j p_{i,j} + \sum_k p_{k,i}$.
Ainsi le second membre de l'inéquation 1.1 se réécrit

$$\begin{aligned}
\sum_{i,j} p_{i,j} \log_2(q_i q_j) &= \sum_{i,j} p_{i,j} \log_2(q_i) + \sum_{i,j} p_{i,j} \log_2(q_j) \\
&= \sum_i \log_2(q_i) \left(\sum_j p_{i,j}\right) + \sum_j \log_2(q_j) \left(\sum_i p_{i,j}\right) \\
&= \sum_i \log_2(q_i) \left(\sum_j p_{i,j} + \sum_k p_{k,i}\right) \\
&= \sum_i \log_2(q_i)\,(2q_i) \\
&= -2H(\mathcal{S}_{\mathcal{M}}) \\
&= -H(\mathcal{S}^2_{\mathcal{M}})
\end{aligned} \tag{1.2}$$

Et donc l'inéquation 1.1 devient

$$H(\mathcal{S}_{\mathcal{M}^2}) \leqslant H(\mathcal{S}^2_{\mathcal{M}})$$

Cette construction se généralise à n'importe quel k en dénombrant les k-uplets de la même manière. □

Les deux situations sont illustrées dans l'exemple suivant.

Exemple : Les messages "1 2 3 4 5 6 1 2 3 4 5 6" et "3 1 4 6 4 6 2 1 3 5 2 5" correspondent à la même entropie en prenant la première extension de la source : 6 caractères de probabilité un sixième chacun, donnent une entropie de $\sum_{i=1}^{6} \frac{1}{6}\log_2(6) = \log_2(6) \approx 2.585$. Avec la deuxième extension de la source $(\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6})$, on obtient 36 groupes possibles chacun de probabilité $\frac{1}{36}$ et d'entropie conforme à la propriété 2 : $\log_2(36) = \log_2(6^2) = 2\log_2(6)$.
Toutefois, en regroupant les messages par exemple par blocs de deux caractères, on obtient :

— (12)(34)(56)(12)(34)(56) : trois couples distincts de probabilité un tiers chacun correspondent à une entropie de $\log_2(3) \approx 1.585$.
— De même (31)(46)(46)(21)(35)(25) donnent 5 couples distincts d'entropie $\frac{1}{6}\log_2(6) + \frac{2}{6}\log_2(\frac{6}{2}) + 3\frac{1}{6}\log_2(6) \approx 2.252$.

Dans les deux cas l'entropie induite obtenue est bien inférieure à deux fois celle induite du message initial. Nous précisons ceci dans la propriété 4.

Propriété 4. *Soient $\mathcal{M}$ un message de taille n et $\mathcal{S}_{\mathcal{M}^k}$ la source dont les probabilités correspondent aux occurrences des k-uplets consécutifs de M. Alors*

$$H(\mathcal{S}_{\mathcal{M}^k}) \leqslant log_2\left(\left\lceil \frac{n}{k} \right\rceil\right).$$

Preuve. Il y a $\lceil \frac{n}{k} \rceil$ k-uplets dans le message M. En outre l'entropie est maximale pour le plus grand nombre de k-uplets distincts possibles d'occurrences égales. Dans ce cas la source équivalente serait d'au plus $\lceil \frac{n}{k} \rceil$ caractères tous distincts et de probabilité d'occurrence $\frac{1}{\lceil \frac{n}{k} \rceil}$. Ce qui donne une entropie de $log_2\left(\lceil \frac{n}{k} \rceil\right)$. □

Ceci nous amène au problème du hasard et de sa génération. Une suite de nombres générée au hasard devra répondre à des critères forts, et en particulier avoir une haute entropie. On ne serait pas satisfait de la suite "1 2 3 4 5 6 1 2 3 4 5 6", qui présente une forme d'organisation aisément détectable. On le serait plus si on rencontrait "3 1 4 6 4 6 2 1 3 5 2 5", qui a une entropie induite supérieure, mais parfois les modes d'organisation et les biais sont plus subtils, et pourtant ils sont de bons angles d'attaque pour les casseurs de codes. Nous détaillerons les générateurs de nombres aléatoires en section 1.3.7.

1.2.4 Stéganographie et tatouage numérique

L'entropie est un outil puissant permettant de modéliser l'information contenue dans un code. Elle peut être utilisée par exemple également pour détecter la stéganographie par une analyse de la quantité d'information présente.
La stéganographie est l'art de dissimuler de l'information. La connaissance même de l'existence d'information cachée peut être suffisante pour la découvrir. Des dispositifs stéganographiques sont par exemple l'encre invisible, les

micropoints dans les images, les Digital Right Management (DRM), le codage d'information dans les espaces blanches d'un texte clair, etc.
De nos jours, la stéganographie est en général combinée à de la cryptographie pour dissimuler l'existence d'information tout en assurant sa confidentialité même dans le cas où cette existence serait révélée. Elle peut également être combinée à de la correction d'erreur. En effet, même si le stégotexte est modifié, et des parties de l'information altérées, le secret peut rester accessible si un nombre suffisant de bits restent inchangés.
Le tatouage numérique est une variante de la stéganographie qui est utilisée pour dissimuler de l'information, résistante aux modifications, dans des images, du son ou de la vidéo numérique.
On distingue deux grands types de tatouage :

- Le tatouage *fragile* est très proche de la stéganographie classique. Il est en général invisible et utilisé pour détecter les modifications du flot. Par exemple les billets de banques sécurisés incluent souvent des tatouages fragiles qui disparaissent à la photocopie.
- Le tatouage numérique *robuste* peut être visible et devrait, au moins partiellement, résister à des modifications simples de la source, comme de la compression sans perte. C'est ce qui est requis par exemple pour les Digital Right Management.

De manière générale, il est difficile de cacher de grandes quantités d'information sans être détecté. En effet, considérons un média M, de l'information X et un stégotexte S où l'information a été cachée à l'intérieur du média. Comme le stégotexte ne doit pas être très différent du média afin de ne pas être détecté, la quantité d'information globale du stégotexte ne peut pas être très différente de la somme des quantités d'informations de M et X : $H(S) \approx H(M)+H(X)$. Ainsi, la stéganalyse classique revient à calculer l'entropie d'un média et à la comparer à des valeurs classiques de média similaires non marqués. Si l'entropie obtenue est sensiblement plus grande que la moyenne, il y a des chances qu'une information supplémentaire soit contenue dans ce média. En d'autres termes, pour être indétectée, la stéganographie doit se limiter à une faible quantité d'information ajoutée.

Exercice 1.9. `Nous avons caché un nombre dans les espaces de ce texte. Pouvez vous le trouver ?` *Solution page 318.*

Nous verrons un exemple de tatouage numérique d'images utilisant le format Joint Photographic Experts Group (JPEG) en section 2.5.3.

1.2.5 Chiffrement parfait

Nous disposons maintenant de l'attirail théorique pour préciser ce que nous entendons par chiffrement inconditionnellement sûr, ou chiffrement parfait. Un

chiffrement est *parfait* si le message chiffré C ne fournit aucune information sur le message initial M. En termes d'entropie, puisque c'est la mesure de la quantité d'information que nous avons adoptée, on aura : $H(M|C) = H(M)$.

Exercice 1.10 (Preuve de la perfection du code de Vernam)**.**

1. *Démontrer que pour un code secret où la clef K est générée aléatoirement pour chaque message M on a $H(M|K) = H(M)$.*
2. *En utilisant les entropies conditionnelles et la définition du code de Vernam, démontrer que dans un code de Vernam ($C = M \oplus K$), on a toujours $H(M, K, C) = H(M, C) = H(M, K) = H(C, K)$; en déduire des relations entre les entropies conditionnelles de M, C et K.*
3. *Démontrer que le chiffrement de Vernam est un chiffrement parfait.*

Solution page 318.

1.2.6 Chiffrement parfait en pratique et principes de Kerckhoffs

On dispose d'une méthode de cryptage dont on possède la preuve de la sécurité. C'est d'ailleurs la seule méthode connue pour laquelle on ait prouvé son caractère parfait.
Mais pour se servir de cette méthode en pratique, il faut savoir générer des clefs aléatoires de même taille que le message à envoyer et ce problème est loin d'être trivial. On utilise un générateur de nombres pseudo-aléatoires pour construire les bits successifs de la clef qui sont alors combinés avec le flot de bits du message clair. On réalise ainsi un chiffrement bit à bit (par flot) comme illustré sur la figure 1.1. Nous aurons l'occasion d'y revenir dans la section 3.2.2.

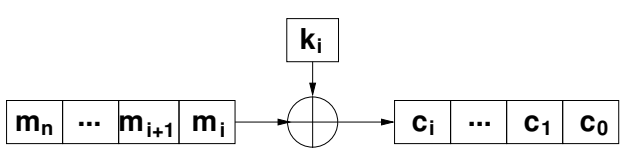

Figure 1.1 – Chiffrement bit à bit (par flot).

Dans tous les cas, comme la clef n'est pas réutilisable, les protocoles d'échange des clefs entre émetteurs et destinataires restent problématiques.
Plus généralement, la sécurité d'un système cryptographique a longtemps reposé sur le secret qui entourait les algorithmes utilisés dans ce système. On citera comme exemple le chiffrement de César (voir la section 1.1) ou, plus récemment, le code ADFVGX utilisé durant la Première Guerre Mondiale par

les forces allemandes. Cette sécurité est illusoire car tôt ou tard, les détails de l'algorithme seront connus et sa faiblesse éventuelle pourra alors être exploitée. Par ailleurs, les systèmes publics sont souvent meilleurs car ils font l'objet d'attaques continuelles, et sont donc soumis à rude sélection. Un système cryptographique dont les mécanismes internes sont librement diffusés et qui résiste aux attaques continues de tous les cryptanalystes pourra être considéré comme sûr.
Le premier à avoir formalisé ces principes est Auguste Kerckhoffs en 1883 dans l'article « La cryptographie militaire » paru dans le « Journal des Sciences Militaires ». Son article comporte en réalité six principes, connus depuis sous le nom de « *principes de Kerckhoffs* ». On en résumera ici que trois, les plus utiles aujourd'hui :

1. La sécurité repose sur le secret de la clef et non sur le secret de l'algorithme.
2. Le déchiffrement sans la clef doit être impossible (en temps raisonnable) ;
3. Trouver la clef à partir du clair et du chiffré est impossible (en temps raisonnable).

Ainsi, toute attaque doit être envisagée en supposant que l'attaquant connaît tous les détails du cryptosystème. Malheureusement ces principes sont souvent encore ignorés. Parmi les exemples récents les plus médiatiques, on citera le cas des algorithmes de chiffrement A5/0 et A5/1 utilisés sur GSM et surtout le logiciel de protection anti-copie de DVD CSS (*Content Scrambling System*) introduit en 1996 qui s'avéra être contourné en quelques semaines en dépit du secret qui entourait l'algorithme de chiffrement utilisé.
Nous allons voir dans la section suivante comment construire des codes qui réutilisent une clef unique et rendent le protocole d'échange moins fastidieux. Sans être parfaits, ces codes tendent vers cette propriété. Le principe général est de découper le message en blocs et de chiffrer chaque bloc séparément.

1.3 Codage par blocs, algèbre et arithmétique

Le chiffrement de Vernam est aujourd'hui le seul algorithme cryptographique à clef secrète prouvé inconditionnellement sûr. Ainsi tous les autres systèmes aujourd'hui connus sont théoriquement cassables.
Pour ces systèmes, on utilise des chiffrements *pratiquement sûrs* : la connaissance du message chiffré (ou de certains couples message clair/message chiffré) ne permet de retrouver ni la clef secrète ni le message clair *en un temps humainement raisonnable* (voir les ordres de grandeur et les limites des machines à la section 1.1.7).

On peut décider par exemple, pour éviter des protocoles d'échange de clef trop fréquents, de choisir une clef unique, et de la réutiliser. Ceci impose le découpage des messages sources en blocs d'une certaine taille, fonction de la taille de la clef. Le chiffrement par blocs est un standard qui sera également largement utilisé pour la détection et la correction d'erreurs.
C'est aussi le principe de l'un des codes les plus célèbres, le code ASCII (pour « *American Standard Code for Information Interchange* »), pour la représentation numérique des lettres et signes de l'alphabet latin.

A	01000001	J	01001010	S	01010011
B	01000010	K	01001011	T	01010100
C	01000011	L	01001100	U	01010101
D	01000100	M	01001101	V	01010110
E	01000101	N	01001110	W	01010111
F	01000110	O	01001111	X	01011000
G	01000111	P	01010000	Y	01011001
H	01001000	Q	01010001	Z	01011010
I	01001001	R	01010010	espace	00100000

Table 1.3 – Un extrait du code ASCII.

Dans le code ASCII, l'alphabet source est l'alphabet latin, et l'alphabet du code est $V = \{0, 1\}$. Les *mots de code* sont tous les mots de longueur 8 sur V :

$$C = \{00000000,\ 00000001,\ \ldots,\ 11111111\}\ . \tag{1.3}$$

Chacun des $2^8 = 256$ caractères (majuscules, minuscules, caractères spéciaux, caractères de contrôle) est représenté par un mot de longueur 8 sur V suivant une fonction de codage dont un extrait est donné par la table 1.3.
Par exemple, le codage ASCII du message : *UNE CLEF*, est la chaîne :
01010101010011100100010100100000010000110100110001000101010001 10.

1.3.1 Blocs et modes de chaînage, du CBC au CTR

On peut coder chaque bloc d'un message par un même algorithme de façon indépendante pour chaque bloc. C'est ce qui est appelé le mode de codage Electronic Code Book (ECB). Plus généralement, cette indépendance de codage entre les blocs n'est pas requise et les différentes façons de combiner les blocs sont appelés *modes de chiffrement*.

Le mode ECB : Electronic Code Book Dans ce mode, le message M est découpé en blocs m_i de taille fixe. Chaque bloc est alors crypté séparément

$$c_i = E_k(m_i) \tag{1.4}$$

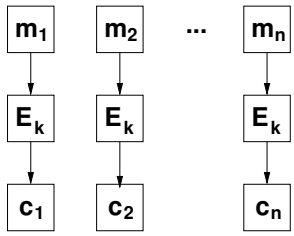

Figure 1.2 – Mode de chiffrement par blocs ECB.

par une fonction E_k, paramétrée par une même clef k, comme sur la figure 1.2. Ainsi un bloc de message m sera toujours codé de la même manière. Ce mode de chiffrement est le plus simple mais ne présente aucune sécurité. Il n'est donc normalement jamais utilisé en cryptographie.

Le mode CBC : Cipher Block Chaining Le mode CBC a été introduit pour qu'un bloc ne soit pas codé de la même manière s'il est rencontré dans deux messages différents.

$$c_i = E_k(m_i \oplus c_{i-1}) \tag{1.5}$$

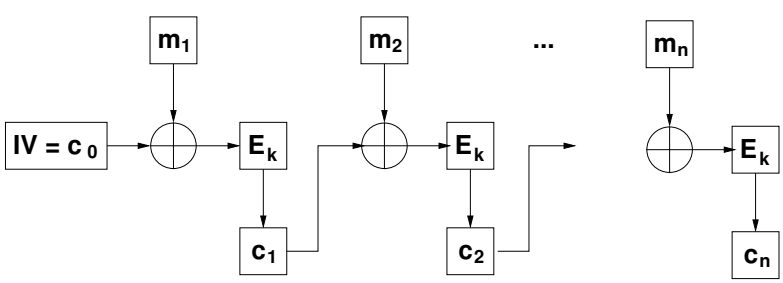

Figure 1.3 – Mode de chiffrement par blocs CBC.

Il faut ajouter une valeur initiale C_0 (ou IV pour « *initial value* »), aléatoire par exemple. Chaque bloc m_i est d'abord composé avec le bloc crypté précédent c_{i-1} par une opération XOR avant d'être lui-même crypté conformément à la figure 1.3. C'est le mode de chiffrement le plus utilisé. Le déchiffrement nécessite l'inverse de la fonction de codage $D_k = E_k^{-1}$ pour déchiffrer : $m_i = c_{i-1} \oplus D_k(c_i)$.

Le mode CFB : Cipher FeedBack Pour ne pas avoir besoin de la fonction inverse pour décrypter, il est possible de faire un XOR après le cryptage, c'est le principe du mode CFB, comme on peut le voir sur la figure 1.4.

$$c_i = m_i \oplus E_k(c_{i-1}) \tag{1.6}$$

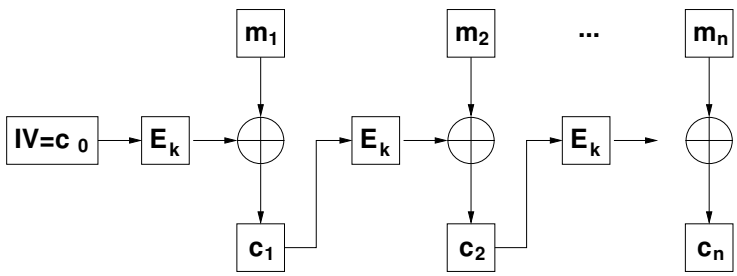

Figure 1.4 – Mode de chiffrement par blocs CFB.

L'intérêt du mode CFB est que le déchiffrement ne nécessite pas l'implémentation de la fonction D_k : $m_i = c_i \oplus E_k(c_{i-1})$. Ce mode est donc moins sûr que le CBC et est utilisé par exemple pour les cryptages réseaux.

Le mode OFB : Output FeedBack Une variante du mode précédent permet d'avoir un codage et un décodage totalement symétrique, c'est le mode OFB suivant la figure 1.5.

$$z_0 = c_0 \ ; \ z_i = E_k(z_{i-1}) \ ; \ c_i = m_i \oplus z_i \tag{1.7}$$

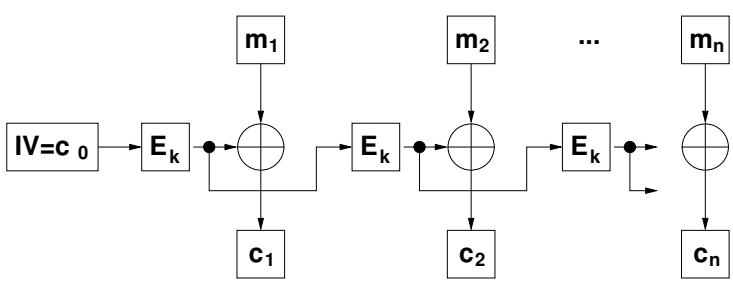

Figure 1.5 – Mode de chiffrement par blocs OFB.

Ce qui se déchiffre par : $z_i = E_k(z_{i-1})$; $m_i = c_i \oplus z_i$. Ce mode est utile dans les satellites pour lesquels minimiser le nombre de circuits embarqués est crucial.

Le mode CTR : CounTeR mode encryption Ce mode est également totalement symétrique, mais en outre facilement parallélisable. Il fait intervenir le chiffrement d'un compteur de valeur initiale T.

$$c_i = m_i \oplus E_k(T+i) \tag{1.8}$$

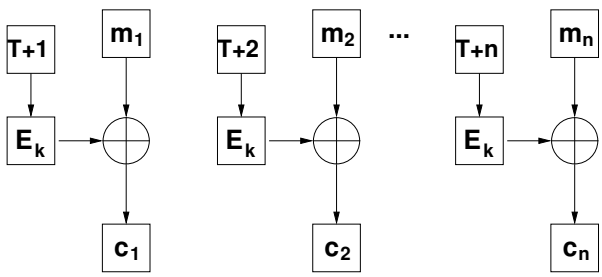

Figure 1.6 – Mode de chiffrement par blocs CTR.

L'intérêt du mode CTR est que les différents calculs sont indépendants, comme pour le mode ECB, mais qu'un même bloc n'est *a priori* jamais codé de la même façon. Enfin, le déchiffrement du mode CTR est identique au chiffrement : $m_i = c_i \oplus E_k(T+i)$.

Exercice 1.11. *Un message M est découpé en n blocs $M = M_1, \ldots, M_n$ qui sont cryptés dans un schéma par blocs en $C = C_1, \ldots C_n$. Bob reçoit les blocs C_i, mais malheureusement, il ne sait pas qu'un et un seul des blocs a été transmis incorrectement (par exemple, des 1 ont été changés en 0 et* vice versa *durant la transmission du bloc C_1).*
Montrer que le nombre de blocs du message mal décryptés par Bob est égal à 1 si l'un des modes ECB, OFB ou CTR a été utilisé et égal à 2 si CBC ou CFB a été utilisé. *Solution page 319.*

1.3.2 Structures algébriques des mots de codes

L'élaboration des codes par blocs nécessite de pouvoir faire des opérations et des calculs sur les blocs. Par exemple, l'opération $\oplus$ sur un bloc de bits est l'addition modulo 2 de deux vecteurs de bits. D'autre part, les fonctions de codage doivent être inversibles, ce qui nécessitera des structures où l'on peut facilement calculer l'inverse d'un bloc. Pour pouvoir faire ces calculs sur des bases algébriques solides, rappelons les structures fondamentales. Tout au long de l'ouvrage, on note $\mathbb{Z}$ l'ensemble des nombres entiers et $\mathbb{N}$ l'ensemble des nombres entiers positifs ou nuls.

Groupes

Un *groupe* $(G, *)$ est un ensemble muni d'un opérateur binaire interne vérifiant les propriétés suivantes :

1. $*$ est associative : pour tous $a, b, c \in G, a * (b * c) = (a * b) * c$.
2. Il existe un élément neutre $e \in G$, tel que pour tout $a \in G$, on trouve $a * e = e * a = a$.
3. Tout élément a un inverse : pour tout $a \in G$, il existe $a^{-1} \in G$ tel que $a * a^{-1} = a^{-1} * a = e$.

De plus, si la loi est commutative ($a * b = b * a$ pour tous $a, b \in G$), alors G est dit *abélien*. On dit qu'un sous-ensemble H de G est un *sous-groupe* de G lorsque les restrictions des opérations de G confèrent à H une structure de groupe. Pour un élément a d'un groupe G, on note a^n la répétition de la loi $*$, $a * \cdots * a$, portant sur n termes égaux à a pour tout $n \in \mathbb{N}^*$.
Si un élément $g \in G$ est tel que pour tout $a \in G$, il existe $i \in \mathbb{Z}$, tel que $a = g^i$, alors g est un *générateur* du groupe $(G, *)$ ou encore une *racine primitive*. Un groupe est dit *cyclique* s'il possède un générateur.
Par exemple, pour un entier n fixé, l'ensemble des entiers modulo n, $\mathbb{Z}/n\mathbb{Z} = \{0, 1, \ldots, n-1\}$, muni de la loi d'addition modulo n est un groupe cyclique généré par 1 ; si $n = 7$, et si on choisit pour loi de composition la multiplication modulo 7, l'ensemble $\{1, \ldots, 6\}$ est un groupe cyclique généré par 3, car $1 = 3^0$, $2 = 3^2 = 9$, $3 = 3^1$, $4 = 3^4$, $5 = 3^5$, $6 = 3^3$.
Soient un groupe $(G, *)$ et $a \in G$. L'ensemble $\{a^i, i \in \mathbb{Z}\}$ est un sous-groupe de G, noté $< a >$ ou G_a. Si ce sous-groupe est fini, son cardinal est l'*ordre* de a. Si G est fini, le cardinal de tout sous-groupe de G divise le cardinal de G.

Propriété 5 (Lagrange)**.** *Soit G un groupe fini. Le cardinal de tout sous-groupe de G divise le cardinal de G. En particulier l'ordre de tout élément divise le cardinal de G.*

Preuve. Soit H un sous-groupe d'un groupe fini G et $|H|$ son cardinal. Pour un élément $a \in G$, on considère le sous-ensemble aH de G : $aH = \{ah, h \in H\}$. Tout d'abord tous les ensembles aH ont le même cardinal : si $ah_1 = ah_2$, comme a est inversible, $h_1 = h_2$, et donc $|aH| = |H|$. Ensuite ces ensembles forment une partition de G : prenons aH et bH avec $a \neq b$ et supposons qu'il existe un élément x dans leur intersection. Alors $x = ah_1 = bh_2$. Ainsi, pour tout élément $u \in aH$, $u = ah_u = b(h_2 h_1^{-1} h_u)$. Mais comme H est un sous-groupe, $h_2 h_1^{-1} h_u \in H$ et donc $u \in bH$. Ce qui prouve que aH est inclus dans bH. Avec l'argument inverse on peut prouver de même que bH est inclus dans aH. Ainsi deux ensembles aH et bH sont soit égaux, soit disjoints. Enfin tout élément x de G est dans xH. En conclusion, les aH forment une partition de