Introduction à la cryptologie

Denis Lapoire

11 septembre 2006

Table des matières

Ces notes de cours sont un support au cours de cryptologie dispensé en deuxième année aux élèves-ingénieurs de l'Enseirb. Le module comporte 9 séances de cours de 1h30 et 9 séances de TD de 2h00.

Cette introduction à la cryptologie présente quelques protocoles de base et étudie la difficulté de ceux-ci. Les protocoles étudiés sont principalement issus de problèmes arithmétiques. Une brève introduction de la "cryptologie quantique" clôt cet enseignement.

Ce document a été librement inspiré d'ouvrages de référence en rayon dans notre chère bibliothèque de l'Enseirb.

- Cryptologie contemporaine. Gilles Brassard, Masson(1993).
  Excellent ouvrage, il présente clairement les enjeux de la cryptologie. Et se lit rapidement !
- Cryptographie appliquée. Bruce Schneier, International Thomson Publishing(1997).
  Cet ouvrage est une référence. Il comporte notamment deux chapitres (I et II) fournissant un grand nombre de protocoles variés. Ces deux chapitres présentent ainsi les enjeux de la cryptologie moderne et s'offrent à une lecture aisée qui ouvrira l'appétit à tout informaticien.
- Cryptographie Théorie et Pratique. Douglas Stinson, International Thomson Publishing(1996).
  Cet ouvrage est une référence. Plus approfondi que le précédent dans l'étude des protocoles, il contient notamment un réquisitoire précis du D.E.S et ses variantes et présente quelques protocoles à divulgation nulle.
- Handbook of Applied Cryptography. Menezes, van Orshoot, Vanstone, CRC Press (1997).
  Cet ouvrage très complet est une petite encyclopédie. À consulter régulièrement. Il est de plus accessible en ligne à l'adresse :
  `http://www.cacr.math.uwaterloo.ca/hac/`.
- Computational complexity. Christos H. Papadimitriou, Addison Wesley(1995).
  Cet ouvrage ne porte pas sur la cryptologie mais sur l'une de ses matières premières : la complexité calculatoire. À recommander absolument à toute personne intriguée par la conjecture $\mathbf{P} \neq \mathbf{NP}$.

Ce document peut contenir des omissions, des contre-sens voire des erreurs. Merci

de me les signaler à : `lapoire@enseirb.fr`. La correction de celles-ci figurera à :
`http://www.enseirb.fr/~lapoire/2emeAnnee/Crypto/cours/`

Ces notes de cours ainsi que d'autres documents pédagogiques sont accessibles à l'adresse : `http://www.enseirb.fr/~lapoire/2emeAnnee/Crypto/`.

Remerciements

Je remercie l'élève-ingénieur Arnaud Ebalard pour l'ensemble des corrections qu'il a su apporter à une précédente version.

Chapitre 1

Introduction

1.1 Survol historique

Avant de tenter de donner une définition de la cryptologie, accordons-lui ses lettres de noblesse.

La civilisation de Sumer (ancienne Babylone, 3^{e}millénaire avant J.C.), nous donne l'exemple de protocole suivant :

Exemple 1 (Sumer) Ce protocole concernait un propriétaire de brebis et son berger également méfiants entre eux : à chaque printemps, le propriétaire confiait au berger des moutons. Celui-ci les lui ramenait à l'automne, les bêtes en plus du fait des naissances étaient alors partagées. Le protocole garantissait au propriétaire une non sous-estimation du troupeau initial et au berger une non surestimation. Il utilisait pour cela de la boue !

Au printemps, les deux compères "comptaient" les animaux en représentant ce nombre par un ensemble de cailloux (calculus en latin) en argile, puis plaçaient ceux-ci dans une boule en argile fermée. Cette boule était scellée (c' est à dire signée à l'aide d'un sceau) par le propriétaire puis confiée au berger. Au retour du troupeau d'automne, le propriétaire brisait cette boule et sous les yeux du berger calculait le nombre de brebis en plus. Le partage pouvait alors avoir lieu.

Cet exemple présentant un protocole d'échange sécurisé d'informations relève de la cryptologie. Notons que de cette représentation d'un nombre par un ensemble de cailloux puis de cailloux et de disques, naîtront l'Écriture ainsi que le Calcul. Ceci permet ainsi de situer la cryptologie comme une science ancienne qui précède l'Écriture : une science préhistorique !

Cette science connut plusieurs révolutions. Nous en considèrerons trois :

l'Écriture l'utilisation de la cryptologie est alors le fait d'états, de diplomates qui souhaitent échanger des courriers en toute confidentialité. On présente alors la cryptologie comme une même pièce composée de deux faces. D'une part, la cryptographie, c'est-à-dire la science de garder secrets des messages.

D'autre part, la cryptanalyse, c'est à dire la science de décrypter des messages chiffrés. Notons la distinction des termes déchiffrer et décrypter : le premier suggère une action légitime, le second une action frauduleuse.

l'informatique et la complexité calculatoire La découverte de l'outil informatique ne révolutionne pas dans un premier temps la cryptologie. Les algorithmes se ressemblent. Au risque de choquer, on peut apparenter le chiffrement de César et le chiffrement D.E.S. Les utilisateurs sont les mêmes : les états, les banques et de grosses entreprises. La méthode est la même : "touiller" les messages pour que cela ait l'air compliqué à décrypter.

La cryptologie est révolutionnée grâce à la découverte de protocoles à clefs publiques qui donnent (enfin !) une assise scientifique à la notion de sécurité. Cette base scientifique a pour nom : la complexité calculatoire. Un autre phénomène bouleverse la cryptologie : l'explosion des protocoles. On ne se contente plus d'échanger des messages en secret, mais on les signe, on leur adjoint un accusé de réception. On vote. On paie anonymement. On joue...

Un bémol toutefois : les protocoles utilisés ne sont pas "impossibles" à casser, ne sont pas non plus "difficiles" à casser, ils sont "supposés" difficiles à casser. Nous reviendrons longuement sur ces notions d'impossibilité, de difficulté ou de difficulté conditionnelle.

la physique quantique Grâce à la physique quantique et l'une de ses découvertes sur l'incertitude du comportement de particules élémentaires, la cryptologie franchit un nouveau pas. Elle offre désormais des protocoles "impossibles" à casser. Du fait du coût, l'implémentation grand public de ceux-ci n'est toutefois pas prévue pour les décennies futures.

1.2 Tentative de définition

Trois mots clefs permettent de définir la cryptologie :

protocole les procédés concernés font intervenir au moins deux participants. On ne s'intéressera donc pas au particulier qui souhaite cacher un secret en utilisant un disque dur ou un trou au fond du jardin.

information l'objet échangé lors du protocole est une information.

sécurité le protocole doit posséder des propriétés garantissant à chaque participant que ses intérêts ne seront pas lésés au cours de ce protocole par un autre participant ou un espion.

Ainsi, peut-on définir la cryptologie comme la science de l'étude des protocoles sécurisés d'échange d'informations.

Notons l'absence de la notion de secret dans cette définition. Il existe en effet des protocoles qui ne requièrent pas, dans leur définition, le secret. Toutefois,

si la notion de secret n'apparaît pas dans la définition, elle apparaît dans son implémentation.

Prenons l'exemple de l'envoi par Alice à Bob d'un document signé mais non chiffré. Le secret ne porte pas sur le document, tout le monde peut en prendre connaissance, mais sur la capacité à imiter la signature de Alice.

Chaque protocole utilisera des informations gardées secrètes par les participants. Ces informations sont appelées des clés privées ou secrètes :

clef secrète information secrète partagée par au moins deux participants d'un protocole.

clef privée information gardée secrète par un des participants.

Citons dès maintenant un autre type de clef :

clef publique information connue de tous (les participants et le monde entier !) et associée à l'un des participants.

1.3 Un premier protocole : l'échange de messages

Afin de situer quelques enjeux de la cryptologie et d'apercevoir quelques difficultés inhérentes à cette science, considérons l'exemple du protocole le plus célèbre : l'échange de messages. Lors de ce protocole, un message m est envoyé par Alice à Bob.

1.3.1 Quelques propriétés

Voici quelques propriétés de sécurité naturelles :

Confidentialité Le message doit rester secret. Tout espion écoutant les conversations de Alice et Bob ne doit pas pouvoir prendre connaissance du message.

Intégrité Bob doit pouvoir vérifier que le message n'a pas été partiellement tronqué ou modifié.

Authentification Bob doit pouvoir s'assurer de l'identité de l'expéditeur.

Signature Bob doit pouvoir s'assurer de l'identité de l'expéditeur et être capable de convaincre un tiers de cette même identité.

Datation

Accusé de réception

Mise en gage Ici, le message m est chiffré et n'est révélé à personne, pas même à Bob. Toutefois, Alice est capable de prouver à une date ultérieure que le document reçu par Bob est bien le chiffrement de m. Une utilisation immédiate est, par exemple, le dépôt de brevet.

1.3.2 Définition générale

Une définition englobant l'immense majorité des protocoles d'échange de message que nous appellerons de façon plus concise *cryptosystème* est fournie par la donnée d'un quintuplet $\Gamma = (\mathbf{M}, \mathbf{C}, \mathbf{K}, \mathbf{E}, \mathbf{D})$ où :

M est l'ensemble des textes *clairs* possibles.

C est l'ensemble des textes *chiffrés* possibles.

K est l'ensemble des *clés* possibles.

E est l'ensemble des fonctions de chiffrement possibles, c'est à dire un ensemble de la forme $(e_k)_{k\in\mathbf{K}}$.

D est l'ensemble des fonctions de déchiffrement possibles, c'est à dire un ensemble de la forme $(d_k)_{k\in\mathbf{K}}$ tel que pour tout texte clair $m \in \mathbf{M}$ et toute clé $k \in \mathbf{K}$ on ait : $d_k(e_k(m)) = m$.

L'utilisation d'un tel système se fait grâce au protocole décrit sur la Figure 1.1. La définition de la Figure 1.1 est très générale. Ce qui explique le caractère un peu

```
ENTRÉE Alice : m
SORTIE Alice :
ENTRÉE Bob   :
SORTIE Bob   : m

Alice et Bob font en sorte de posséder respectivement
      deux fonctions e_k et d_k pour une même clef k ;
Alice calcule c = e_k(m) ;
      envoie c à Bob ;
Bob   calcule d_k(c) ;
```

FIG. 1.1 – Protocole d'envoi de message

flou de l'instruction `Alice et Bob font en sorte de posséder respectivement ...` Plusieurs protocoles réalisent cette instruction, qu'ils utilisent une clef secrète k, un jeu de clefs, clef privée clef publique. Plusieurs chapitres entiers sont consacrés à ces protocoles.

1.3.3 Attaque d'un tel protocole

Étudier la sécurité d'un protocole suppose l'existence de personnes :

1. malintentionnées.
2. douées de capacités d'écoute (des oreilles, voire des écouteurs).
3. douées de réflexion (un cerveau, voire un ou des ordinateurs).

Ceci est bien plus qu'une hypothèse. C'est un fait. On en déduira qu'un protocole est sûr si il résiste à l'attaque de toute personne qui dispose notamment des moyens suivants :

- il connaît le protocole Γ utilisé.

  Cette hypothèse est cruciale. L'idée selon laquelle un protocole serait d'autant plus sûr qu'il est secret est naïve pour plusieurs raisons. Citons en quelques-unes dès à présent :
  - Un protocole secret entre deux participants est un protocole dont la sécurité n'a été étudiée que par ces deux personnes. Autant dire personne. Un protocole est considéré comme sûr si il a été rendu public et si il a fait l'objet de critiques de la communauté cryptologique. L'exemple du chiffrement de César, présenté ci-après, est de ce point de vue édifiant.
  - L'ensemble des protocoles qui seront créés par des hommes, d'ici disons 100 ans, est en nombre infiniment plus petit que tout espace de clefs de tout protocole à sécurité raisonnable ($2^{256} \simeq 10^{77}$). En d'autres termes, une attaque exhaustive sur l'ensemble de ce type de protocoles est plus rapide qu'une attaque exhaustive sur l'ensemble des clefs d'un protocole public considéré comme sûr.
- il a accès à tous les échanges.
- il a accès à des moyens de calcul.

Exemple 2 (Chiffrement de César) Lors de ses campagnes militaires en Gaule, Jules César utilisait un chiffrement qui consistait à remplacer chaque lettre par la troisième qui la suivait dans l'ordre alphabétique. L'histoire ne dit pas si une indiscrétion romaine auprès des gaulois lui porta préjudice.

Différentes sorties possibles d'une attaque

Cette liste n'est pas exhaustive. Bien au contraire ! Obtenir un haut niveau de sécurité revient à protéger un protocole contre un attaquant doué de plus grandes capacités. Si l'on considère par exemple la propriété de confidentialité pour un protocole d'échange de message, nous envisagerons les différentes attaques possibles :

attaque à texte chiffré Oscar connaît $e_k(m)$.

attaque à texte en clair connu Oscar connaît une séquence de textes en clair et leurs chiffrements, c'est à dire une séquence de la forme $(m_1, e_k(m_1)), \ldots, (m_l, e_k(m_l))$.

attaque à texte en clair choisi statique Oscar connaît une séquence de la forme $(m_1, e_k(m_1)), \ldots, (m_l, e_k(m_l))$ où $m_1, \ldots, m_l$ sont choisis par Oscar lui-même.

attaque à texte en clair choisi dynamique Oscar connaît une séquence de la forme $(m_1, e_k(m_1)), \ldots, (m_l, e_k(m_l))$ où $m_1, \ldots, m_l$ sont choisis par Oscar lors du déroulement de l'attaque, et donc non nécessairement au début.

Ces hypothèses ne sont pas la conséquence du seul désir de jeux intellectuels mais sont nécessitées par l'histoire de la cryptanalyse. Un exemple célèbre et douloureux pour certains est la bataille de Midway.

Exemple 3 (Bataille de Midway) Cette bataille aéronavale (juin 1942) opposa les flottes américaines et japonaises dans le Pacifique. L'état-major japonais communiquait avec l'un de ses postes avancés par ondes radios en utilisant constamment une clef secrète k. Les américains réalisèrent une attaque à texte en clair choisi en transmettant en clair un message m entre deux de leurs postes. Le message était suffisamment important pour que le poste avancé japonais envoie immédiatement à son état-major une copie chiffré $e_k(m)$ de m. Il suffit aux américains d'intercepter le message radio pour obtenir ainsi le couple $(m, e_k(m))$ et calculer la clef k.

Les attaques évoquées ci-dessus sont dites passives. Nous distinguons ainsi de façon générale deux types d'attaques :

attaques passives le ou les attaquants ne font que lire les messages échangés ; ils ne modifient pas les émissions.

attaques actives le ou les attaquants modifient le contenu (ajout, remplacement, destruction partielle ou totale), retardent (ou anticipent) les messages, ...

Définir une attaque consiste non seulement, comme nous l'avons vu précédemment, à préciser les connaissances supposées de l'attaquant mais à définir l'objectif de l'attaque. Sur l'exemple d'un échange de messages, les attaques peuvent être hiérarchisées ainsi :

cassage complet Oscar calcule la clef k et connaît ainsi la fonction de déchiffrement d_k.

cassage global Oscar calcule la fonction de déchiffrement d_k.

cassage local Oscar calcule le texte clair $d_k(c)$.

cassage partiel Oscar calcule quelque information sur le message en clair ou la clef.

Exemple 4 (Attaque combinée) Si Oscar souhaite casser un mot de passe à 6 caractères, il peut dans un premier temps essayer simplement de calculer la première lettre. Une attaque exhaustive sur les 5 derniers caractères est alors 256 fois plus rapide que sur les 6 caractères ($O(2^{40})$ au lieu de $O(2^{48})$).

Un cassage partiel équivaut parfois à un cassage complet. Nous verrons en TD pourquoi dans certains protocoles le simple calcul du premier bit d'une clef permet en fait de calculer la clef entière.

Conclusion

Définir un protocole sûr, c'est définir un protocole robuste face à une attaque disposant de moyens importants (Entrée) et ayant un objectif (Sortie) modeste. Ce propos est illustré par le graphique suivant.

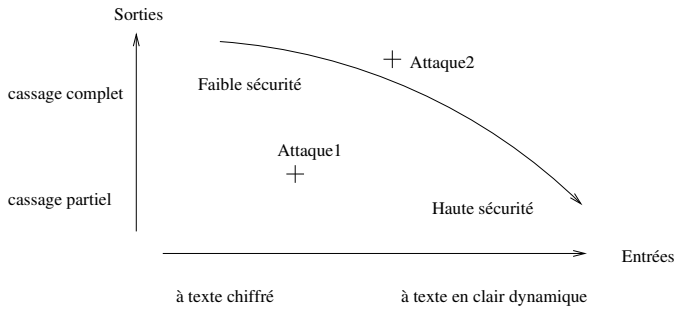

FIG. 1.2 – Critère de robustesse

Ni la vie ni la science ne sont linéaires. Parfois, il n'est pas possible de comparer ni deux entrées, ni deux sorties, ni à fortiori deux couples entrée/sortie. Ainsi, aprfois, les attaques peuvent ne pas pouvoir être comparées (cas des attaques 1 et 2 sur le graphique).

1.3.4 Un système à sécurité inconditionnelle : le masque jetable

Il existe un protocole d'échange de messages possédant un degré de sécurité absolu, dite sécurité inconditionnelle. Ce protocole dit protocole à masque jetable ou chiffrement de Vernam fut utilisé longtemps par la diplomatie soviétique. La sécurité inconditionnelle implique que Oscar ne peut extraire aucune information d'un message chiffré $e_k(m)$, même en supposant qu'il ait une capacité de calcul infinie.

Pour formaliser cette intuition, on traduit l'idée "connaître ou non le message chiffré $e_k(m)$, c'est pareil !". Ainsi, une définition plus formelle tirée de la théorie de l'information indique qu'un cryptosystème $\Gamma = (\mathbf{M}, \mathbf{C}, \mathbf{K}, \mathbf{E}, \mathbf{D})$ est *inconditionnellement sûr* si pour tout message possible en clair $m \in \mathbf{M}$ et tout message possible chiffré $c \in \mathbf{C}$ la probabilité que le texte en clair soit m est la probabilité que le texte en clair soit m sachant que le message chiffré est c (pour plus de détails, voir l'ouvrage de D. Stinson).

Le protocole de Vernam est défini à partir du quintuplet $(\mathbf{M}, \mathbf{C}, \mathbf{K}, \mathbf{E}, \mathbf{D})$ où :

$\mathbf{M}, \mathbf{C}, \mathbf{K}$ sont égaux à $\{0,1\}^n$, c'est à dire l'ensemble des mots binaires de longueur n fixé.

$\mathbf{E} = (e_k)_{k \in \mathbf{K}}$ où pour tout $m = m_1 \dots m_n \in \mathbf{M}$ et tout $k = k_1 \dots k_n \in \mathbf{K}$, on a : $e_k(m) = c_1 \dots c_n$ avec $c_i = (m_i + k_i) mod\ 2$.

$\mathbf{D} = (e_k)_{k \in \mathbf{K}}$

et a pour écriture celle de la figure 1.3.

```
ENTRÉE Alice : m
SORTIE Alice :
ENTRÉE Bob   :
SORTIE Bob   : m

Alice et Bob échangent par un canal sûr une clé aléatoire k ;
Alice calcule c = e_k(m) ;
      envoie c à Bob ;
Bob   calcule d_k(c) ;
```

FIG. 1.3 – Protocole de Vernam

Fait 1 Le chiffrement de Vernam est inconditionnellement sûr.

preuve :
Soient M, C et K les variables associées respectivement au message en clair, au message chiffré et à la clef. Afin d'établir l'égalité $p(M = m \mid C = c) = p(M = m)$ pour tous messages m et c, établissons que C décrit à l'image de K une loi uniforme.

Pour tout message $c \in \mathbf{C}$, la probabilité $p(C = c)$ est successivement égale à $\sum_{k \in \mathbf{K}} p(C = c, K = k)$, $\sum_{k \in \mathbf{K}} p(M = c+k, K = k)$, $\sum_{k \in \mathbf{K}} p(M = c+k) \cdot p(K = k)$ (car M et K sont indépendants), $\frac{1}{2^n} \cdot \sum_{k \in \mathbf{K}} p(M = c + k)$, $\frac{1}{2^n}$ (car $\sum_{k \in \mathbf{K}} p(M = c + k) = \sum_{k \in \mathbf{K}} p(M = k) = 1$). Ainsi, tout message c vérifie $p(C = c) = \frac{1}{2^n}$.

Pour tous $m \in \mathbf{M}$ et $c \in \mathbf{C}$, établissons $p(M = m \mid C = c) = p(M = m)$. La probabilité $p(M = m \mid C = c)$ est égale à $p(M = m, C = c)/p(C = c)$, $p(M = m, K = m + c)/p(C = c)$ (on a : $M + C = K$), $p(M = m) \cdot p(K = m + c)/p(C = c)$ (M et K sont indépendants), $p(M = m) \cdot \frac{2^n}{2^n}$ (car K et C sont uniformes) et donc $p(M = m)$.

Le diable est caché dans les détails !

Le protocole de Vernam est inconditionnellement sûr : tout message chiffré c peut avec la même probabilité, être le chiffrement de n'importe quel texte en clair m.

Le cours de cryptologie ne termine pas pour autant par la donnée de ce protocole ayant une sécurité indépassable ! Une lecture détaillée du protocole indique deux contraintes :

1. la capacité de tirer aléatoirement un entier.

2. l'utilisation d'un canal sûr pour échanger la clef.

La contrainte k aléatoire est cruciale : en effet, si Oscar était capable de prédire, en tout ou partie, la clef k échangée, une attaque serait possible. Conséquence de la nécessité de choisir aléatoirement la clef, celle-ci ne peut pas être utilisée deux fois : on doit la *jeter* après utilisation. D'où le nom de masque jetable.

La contrainte de l'utilisation d'un canal sûr est très forte, et diminue quelque part l'intérêt d'un tel protocole. Un des principes premiers en cryptologie est l'absence de canal sûr, ou du moins son utilisation parcimonieuse. De tels canaux peuvent effectivement être construits, par des moyens non cryptologiques, mais s'avèrent d'un coût prohibitif.

L'exemple du système diplomatique soviétique nous l'indique. Ces clefs circulaient dans des valises diplomatiques fortement surveillées et exigeaient des moyens humains importants.

1.4 Une science amusante et exigeante

Amusante car tout protocole procédant par échange de données requérant quelque sécurité que ce soit relève de la cryptologie. Ces protocoles peuvent déjà exister ou restent même à inventer : en effet que penser d'un protocole qui permettrait à Alice de prouver à Bob une certaine connaissance sans rien révéler à Bob ?

De plus, ces protocoles peuvent même être affinés en ajoutant de nouvelles conditions. En effet que penser d'un protocole de paiement par carte bancaire qui préserve l'anonymat auprès du commerçant, de la banque ou même de toute institution ?

L'une des contreparties est une certaine difficulté. Il ne s'agit pas seulement de concevoir des algorithmes, exercice délicat, ni seulement d'en étudier la correction, exercice encore plus délicat, mais d'en étudier la sécurité. Cet exercice est nouveau car il consiste à résoudre un problème par un algorithme et de prouver l'inexistence d'algorithmes le cassant !

Chapitre 2

Cryptosystèmes à clefs secrètes

Ce chapitre fournit une description de protocoles qui ont été historiquement les plus employés. Ces protocoles permettent à Alice d'envoyer un message à Bob en utilisant une clef secrète à longueur fixe. L'enjeu ici est de pouvoir envoyer un message d'une longueur éventuellement grande, plusieurs milliers de caractères, en utilisant une clef de petite taille, n'excédant pas quelques centaines de bits, et de ne s'exposer à aucune attaque, fût-elle à texte en clair choisi. Nous ne nous intéressons pas ici à l'échange des clefs, elle sera abordée ultérieurement.

Ce chapitre s'inspire librement de l'ouvrage de D. Stinson. Pour plus de détails, le lecteur est invité à s'y reporter.

2.1 Quelques vieux cryptosystèmes alphabétiques

Nous considérons ici, l'alphabet latin $\mathcal{A}$ composé des 26 lettres `'a'`,`'b'`,...`'z'`. Nous considérons cet ensemble en bijection avec l'intervalle d'entiers $[0, 25]$ et nous le munissons des opérations arithmétiques modulaires élémentaires.

Exemple 5 Quelques exemples d'égalités : $'\mathtt{a}' + '\mathtt{d}' = '\mathtt{d}'$ car $(0+3) mod\, 26 = 3$; $'\mathtt{x}' + '\mathtt{e}' = '\mathtt{b}'$ car $(23+4) mod\, 26 = 1$; $'\mathtt{d}' \cdot '\mathtt{e}' = '\mathtt{m}'$ car $(3 \cdot 4) mod\, 26 = 12$.

L'ensembles des messages en clair et des messages chiffrés sont en fait les mots définis sur cet alphabet.

2.1.1 Chiffrement par décalage

Ce chiffrement est en fait une généralisation du chiffrement de César. L'ensemble des clefs $\mathbf{K}$ est l'alphabet $\mathcal{A}$. La fonction de chiffrement associe à tout mot $m = m_1 \dots m_l$ et à toute clef k le mot $e_k(m) = (m_1 + k) \dots (m_l + k)$. La fonction de déchiffrement opère de façon similaire en soustrayant la clef k à chacune des lettres : $d_k(c) = (c_1 - k) \dots (c_l - k)$.

Exemple 6 Sont représentés sur la Figure 2.1 un mot en clair et son chiffrement en utilisant la clef $k = '\mathtt{l}' = 11$:

```
wewillmeetatmidnight
lllllllllllllllllll
hphtwwxppelextoytrse
```

FIG. 2.1 – Chiffrement par décalage

Attaque à texte chiffré

En supposant que le texte en clair soit en anglais (ou tout autre langue) et soit de longueur suffisante (≥ 4), et en admettant (on en conviendra aisément) qu'il existe un algorithme efficace décidant si un mot appartient ou non à cette langue, il est facile de réaliser une attaque exhaustive sur l'espace des clefs. Cet espace a une cardinalité ridiculement faible.

Notons que l'attaque ici fonctionne car il est extrêmement peu probable que deux textes anglais suffisamment longs puissent avec deux clefs différentes se coder identiquement. En d'autres termes, on suppose qu'il existe un unique décalage d pour lequel $c + d^{|c|}$ soit anglais.

Toute attaque à texte chiffré requiert une information sur le texte en clair

L'attaque précédente supposait que le texte en clair m est anglais. Si on n'émet pas ce genre d'hypothèse (c'est à dire le fait que m n'est pas purement aléatoire), il est impossible de calculer avec certitude la clef, la fonction de déchiffrement ou le texte en clair.

La preuve dans le cas du chiffrement par décalage est très simple : supposons, à contrario, que Alice choisisse un texte en clair m aléatoire, chiffre m en utilisant un décalage aléatoire d. Le message chiffré est ainsi $c := m + d^{|m|}$.

Notons M, C, K les variables aléatoires associées aux messages en clair, à la clef et au message chiffré. La probabilité de connaître c sachant m est toujours différente de 1, elle est égale à $\frac{1}{|\mathbf{K}|}$ c'est à dire $\frac{1}{26}$. En effet, la probabilité $p(M = m \mid C = c)$ est égale à :

- $\frac{1}{26}$ si $c - m$ est de la forme $d^{|m|}$ avec $d \in [0, 25]$.
- 0 sinon.

Le cryptosystème n'est pas inconditionnellement sûr : l'attaquant connaît quelques informations sur m (dans ce cas-ci, il en connaît beaucoup), mais hésite entre $26 = |\mathbf{K}|$ solutions possibles, qu'il ne peut pas distinguer par un futur calcul.

2.1.2 Chiffrement affine

Le chiffrement affine est une généralisation du chiffrement par décalage. Il utilise pour clefs l'ensemble des couples de la forme (k_1, k_2) où k_1 est premier avec 26 et k_2 un entier dans $[0, 25]$ quelconque.

Chaque lettre m_i du mot m est chiffrée par la lettre $c_i = k_1 \cdot m_i + k_2$. Chaque lettre c_i du mot chiffré est déchiffré en utilisant k_1^{-1} l'inverse modulo 26 de l'entier k_1. La lettre m'_i obtenue par l'opération $m'_i = k_1^{-1} \cdot (c_i - k_2)$.

Attaque à texte chiffré

L'espace des clefs $\mathbf{K}$, égal à $\{1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25\} \times [0, 25]$, est de cardinalité 364. Pour les mêmes raisons que le chiffrement à décalage, un tel cryptosystème s'offre à une attaque exhaustive.

2.1.3 Chiffrement monoalphabétique par permutation

Afin de faire échec aux attaques exhaustives du chiffrement affine, une idée consiste à étendre l'espace des clefs à l'ensemble des permutations sur $\mathcal{A}$ (rappel : une permutation σ sur $\mathcal{A}$ est une bijection de $\mathcal{A}$ dans lui-même). Ce chiffrement a pour nom, le chiffrement monoalphabétique par substitution.

Ainsi, si σ est la clef, chaque lettre m_i du message en clair est chiffrée en $c_i = \sigma(m_i)$, chaque lettre c_i du message chiffré est déchiffré en $m'_i = \sigma^{-1}(c_i)$.

Attaque à texte clair exhaustive

Une permutation sur $\mathcal{A}$ peut être décrite comme une séquence sans répétition d'éléments de $\mathcal{A}$. Le nombre de permutations sur $\mathcal{A}$ est 26!. D'après la formule de Stirling ($n! = \sqrt{2\pi n}(\frac{n}{e})^n \cdot (1 + \Theta(\frac{1}{n}))$), 26! est proche de $3 \cdot 10^{26}$.

Sachant qu'une année contient $3 \cdot 10^7$ secondes, une attaque exhaustive employant 10 millions d'ordinateurs testant chacun 1 milliard de clefs à la seconde nécessiterait 1000 ans. Ce qui fait quand même beaucoup. Un telle attaque est donc irréalisable aujourd'hui.

Loi de Moore : une exponentielle peut en linéariser une autre !

Il faut se méfier des énoncés du type précédent. Car ils supposent que les ordinateurs ne s'améliorent pas. Or selon la loi de Moore, les ordinateurs voient leur capacité de calcul doublée tous les 18 mois. En supposant que ce soit le cas pour les trente prochaines années, la même attaque en 2031 nécessiterait 10 000 ordinateurs (à capacité 10^{15} clefs par seconde) et 1 an. Ce qui pourrait être réalisée par une grande communauté d'internautes très motivés. En 2061, elle nécessiterait un unique ordinateur (à capacité 10^{21} clefs par seconde) pendant 3 jours. Ce qui pourrait être réalisée par n'importe qui.

Attaque à texte clair connu

Si l'attaquant connaît un couple $(m, e_k(m))$ où m contient les 26 lettres de l'alphabet, il connaît ainsi la fonction de déchiffrement (ainsi que la clef).

Si l'attaquant connaît un couple $(m, e_k(m))$ où m contient seulement 13 lettres de l'alphabet, il peut par une attaque exhaustive sur les 13 autre lettres calculer la fonction de déchiffrement. Sur un ordinateur traitant une clef en 10^{-9} seconde, 13! opérations nécessitent 6 secondes ($13! \simeq 6 \cdot 10^9$).

Ces attaques sont rendues possibles par une faiblesse du cryptosystème : la faible longueur des blocs. En effet, le message en clair est découpé en blocs que sont les lettres, chacun des blocs étant chiffré par la même fonction. Le faible nombre de blocs (26) autorise une attaque à texte clair connu.

Une condition nécessaire mais non suffisante pour la sécurité

Ainsi, la sécurité n'exige pas uniquement d'avoir un grand espace de clefs, elle exige aussi d'avoir un grand espace de blocs.

Quelques propriétés statistiques des langages naturels

Réaliser une attaque à texte chiffré peut utiliser les propriétés du message en clair. Si l'on sait que ce texte est un texte en langue anglaise (pour les autres langues, on a des propriétés similaires) on peut supposer avec une forte probabilité qu'il a les propriétés suivantes.

La fréquence d'apparition des caractères dans la langue anglaise est donnée par le tableau suivant :

`e` 0, 127
`t` 0, 091 `a` 0, 082 `o` 0, 075 `i` 0, 070 `n` 0, 067 `s` 0, 063 `h` 0, 061 `r` 0, 060
`d` 0, 043 `l` 0, 040
`c` 0, 028 `u` 0, 028 `m` 0, 024 `w` 0, 023 `f` 0, 022 `g` 0, 020 `y` 0, 020 `p` 0, 019
`b` 0, 015
`v` 0, 010 `k` 0, 008 `j` 0, 002 `x` 0, 001 `q` 0, 001 `z` 0, 001

De même, les 29 digrammes les plus courants sur 676 en langue anglaise sont :

```
th he in er an re ed on es st en at to nt ha
nd ou ea ng as or ti is et ar te se ho of
```

Pour information, les 12 trigrammes les plus fréquents sur 17576 sont :

```
the ing and her ere ent tha nth was eth for dth
```

Attaque à texte chiffré non exhaustive

On peut utiliser les propriétés statistiques de la langue anglaise (ou toute autre langue) pour écrire un algorithme de décryptage. Cet algorithme est composé de plusieurs parties :

1. une première partie qui analyse la fréquence des caractères, des digrammes et des trigrammes du texte chiffré.

2. une seconde partie contenant :
   - une liste des propriétés vérifiées par la fonction de déchiffrement σ issue des comparaisons des fréquences dans le texte chiffré et dans la langue considérée.
   - une parcours exhaustif des fonctions vérifiant ces propriétés afin d'obtenir celle permettant d'obtenir un texte anglais.

À titre d'illustration, nous reprenons l'exemple fourni par D. Stinson. Le texte chiffré c est le suivant :

```
YIFQFMZRWQFYVECFMDZPCVMRZWNMDZVEJBTXCDDUMJ
NDIFEFMDZCDMQZKCEYFCJMYRNCWJCSZREXCHZUNMXZ
NZUCDNJXYYSMRTMEYIFZWDYVZVYFZUMRZCRWNZDZJJ
XZWGCHSMRNMDHNCMFQCHZJMXJZWIEJYUCFWDJNZDIR
```

Une analyse des fréquences nous donne les résultats suivants.

Fréquence des caractères

```
20 Z          7 E
16 M          6 X
15 C          5 I U V
13 D          4 H Q
11 F J        3 S
10 R Y        2 T
 9 N          1 B G K P
 8 W          0 A L O
```

Fréquence des digrammes

```
4 DZ MD MR ZW
3 CD CH FM IF NM NZ ZU
2 CF DI EJ EY FQ FZ HZ JM JN JX MX NC QF RN RW RZ SM
  UC UM VE WD WN XC XZ YF YI YV ZC ZD ZJ ZR ZV
1 BT CE CJ CM CR CS CV CW DD DH DJ DM DR DU DY EC EF
  EX FC FE FW FY GC HN HS IE IR JB JC JJ JY JZ KC ME
  MF MJ MQ MY MZ ND PC QC QZ RE RJ RT SZ TM TX UN VM
  VY VZ WG WI WJ WQ XJ XY YR YS YU YY ZK ZN ZP
```

Les premières propriétés de la fonction de déchiffrement σ sont :

1. $\sigma(\mathtt{Z}) = \mathtt{e}$ car ce sont les caractères de plus hautes fréquences.
2. $\sigma(\mathtt{D}) \in \{\mathtt{r}, \mathtt{s}, \mathtt{t}\}$ car les seules lettres α présentes dans des digrammes anglais fréquents de la forme $\mathtt{e}\alpha$ et $\alpha\mathtt{e}$ sont `r`, `s` ou `t` et la seule lettre présente dans des digrammes du texte chiffré de la forme $\mathtt{Z}\beta$ et $\beta\mathtt{Z}$ est `D` (`DZ` et `ZD` apparaissent respectivement 4 et 2 fois).
3. $\sigma(\mathtt{W}) \in \{\mathtt{d}, \mathtt{n}, \mathtt{a}\}$ car `RW` et `WR` apparaissent respectivement 3 et 0 fois.
4. $\sigma(\mathtt{R}) = \mathtt{n}$ car `RW` et `WR` apparaissent respectivement 3 et 0 fois.

5. $\sigma(\texttt{N}) = \texttt{h}$ car `NZ` et `ZN` apparaissent respectivement 3 et 1 fois.
6. $\sigma(\texttt{C}) = \texttt{a}$ et $\sigma(\texttt{W}) = \texttt{d}$ suggéré par la présence de la séquence `neCnWhe` dans le texte partiellement déchiffré, l'appartenance $\sigma(\texttt{W}) \in \{\texttt{d}, \texttt{n}, \texttt{a}\}$ et l'égalité $\sigma(\texttt{R}) = \texttt{n}$.

A ce stade, le texte est partiellement déchiffré en :

```
------end-----a---e-a--nedh--e------a-----
h-------ea---e-a---a---nhad-a-en--a-e-h--e
he-a-h------n------ed---e---e--neandhe-e--
-ed-a---nh---ha---a-e----ed-----a-d--he--n
```

Les lettres qu'ils restent à déchiffrer sont par ordre de fréquence décroissante non nulle : `M D F J Y E X I U V H Q S T B G K P`. Deux stratégies s'offrent à nous :

à la main Soit on poursuit en utilisant les techniques déjà vues. Et quitte à émettre quelques hypothèses, on obtient le déchiffrement complet. La suite est présentée dans l'ouvrage de D. Stinson.

exhaustive Chacune des 18 lettres à déchiffrer ne pouvant être associée qu'à au plus 5 lettres en moyenne du fait des fréquences : par exemple, la lettre `M` de fréquence $0,095 = \frac{16}{168}$ a pour déchiffrement un des symboles non encore déchiffrés `t,o,i,s,r` à fréquence comprise entre $0,091$ et $0,060$.

Le nombre d'opérations d'une telle attaque serait proche de 18^5, c'est à dire inférieur à $2 \cdot 10^6$ qui est accessible à tout ordinateur personnel et ce en un temps court.

Afin de ne pas laisser le lecteur sur sa faim, voici le message en clair :

```
ourfriendfromparisexaminedhisemptyglasswit
hsurpriseasifevaporationhadtakenplacewhile
hewasntlookingipouredsomemorewineandhesett
ledbackinhischairfacetilteduptowardsthesun
```

2.1.4 Chiffrement de Vigenère

Afin de faire échec à la cryptanalyse précédente, un savant du nom de Vigenère (16^{e} siècle) eut l'idée non pas de déchiffrer chaque caractère uniformément mais de le déchiffrer selon notamment la place qu'il occupe dans le texte en clair. Pour cela, il choisit une clef k qui est un mot, découpe le message en clair m en blocs de longueur celle de la clef k et additionne à chaque bloc la clef.

Plus formellement, le chiffrement de Vigenère est le cryptosystème défini par :

M=C=K sont les mots définis sur l'alphabet $\mathcal{A} = \{\texttt{a}, \ldots, \texttt{z}\}$.

E est l'ensemble des fonctions de chiffrement de la forme e_k avec $k \in \mathbf{K}$ qui associe à tout mot $m \in \mathbf{M}$ le mot $c \in \mathbf{C}$ ainsi : si l désigne la longueur de la clef k et l' celle de m, pour tout $i \in [1, l']$ la i^{e}lettre c_i de c est la i^{e}lettre de m additionnée à la $(1 + (i-1) mod\, l)^{\text{e}}$lettre de k (voir exemple ci-dessous).

D est l'ensemble des fonctions de déchiffrement définies de façon similaire à ci-dessus en opérant non pas une addition mais une soustraction.

Exemple 7 Le chiffrement du mot `unepetitephrase` selon la clef `code` est réalisée ainsi :

$m =$ `unepetitephrase`
`codecodecodecod`
$c =$ `wbhtghlxgdkvcgh`

FIG. 2.2 – Chiffrement de Vigenère

Autre propriété statistique des langages naturels

Définissons pour tout couple de mots w, w', l'*indice de coïncidence* de w et w', noté $I(w, w')$, égal à la probabilité que deux lettres respectivement de w et w' soient égales.

Le calcul de $I(w, w')$ s'effectue simplement en calculant la somme $\sum_{i\in[1,26]} \frac{f_i f'_i}{|w|\cdot|w'|}$ où f_i (resp. f'_i) désigne le nombre de fois où le i^{e}caractère de l'alphabet apparaît dans w (resp. w') et où $|w|$ (resp. $|w'|$) désigne la longueur de w (resp. w').

Conséquence des propriétés statiques des fréquences d'apparition, il est aisé de démontrer la forte probabilité que deux textes anglais w et w' aient un indice de coïncidence proche de $\sum_{i\in[1,26]} p_i^2$ (où p_i est la fréquence d'apparition du i^{e}caractère en anglais : $p_5 = 0,127$, $p_1 = 0,082$, etc) c'est à dire proche de $0,065$.

Une extension naturelle de l'indice de coïncidence est celle relative à un décalage : l'*indice de coïncidence de deux mots w et w' relativement à un décalage* $d \in \mathcal{A}$ est le réel noté $I(w, w', d)$ égal à $I(w+d^{|w|}, w')$ où $w+d^{|w|}$ est le mot obtenu à partir de w en additionnant à chacune de ses lettres la lettre d. Des études statistiques ont démontré que pour tous mots anglais w et w' et tout décalage non nul, $I(w, w', d)$ a une forte probabilité d'être proche de $\frac{1}{26}$, ou plus précisément d'appartenir à l'intervalle $[0,031; 0,045]$.

En résumé, pour tous mots anglais w et w' et tout décalage $d \in \mathcal{A}$, on a :

$$I(w, w', d) \simeq 0,065 \quad \text{si } d = 0$$
$$I(w, w', d) \in [0,031, 0,045] \quad \text{sinon.}$$

Il est facile d'étendre ce résultat à tout couple de mots aléatoirement extraits de deux mots anglais (suffisamment longs).

Attaque du chiffrement de Vigenère

Une attaque non exhaustive du chiffrement de Vigenère utilisant les propriétés statistiques du texte en clair anglais procède en deux temps :

`ENTRÉE: un mot` c `de la forme` $e_k(m)$
`SORTIE: la clef` k

`Calcul de la longueur` l `de la clef` k ;
`Calcul des` l `décalages` $k_1, \ldots, k_l$;
`Retourner` $(k_1, \ldots, k_l)$;

Considérons un texte chiffré c de la forme $e_k(m)$. Pour tout entier $i \geq 1$ et pour tout entier $j \in [1, i]$, notons $c(i, j)$ le mot obtenu à partir de c en découpant c en plusieurs blocs consécutifs de longueur i et en ne conservant que les j^{es}lettres de chacun de ces blocs (si les blocs sont disposés en lignes, $c(i, j)$ est donc la j^{e}colonne).

Conséquence des propriétés énoncées précédemment, les quelques faits suivants :

Fait 2 Pour tout entier $j \in [1, k]$ on a :

$$I(c(k, j), c(k, j), 0) \simeq 0,065$$

preuve :
L'entier $I(c(k, j), c(k, j), 0)$ est égal à $I(m(k, j) + k_j^l, m(k, j) + k_j^l, 0)$ (k_j^l désigne le mot composé de l fois le caractère k_j), à $I(m(k, j), m(k, j), 0)$ donc à $0,065$ (puisque $m(k, j)$ est extrait d'un mot anglais).

Le fait précédent permet de distinguer par le calcul certaines mauvaises longueurs de clefs. Ce fait peut être étendu afin d'obtenir une caractérisation de la longueur de la clef. Cette caractérisation suppose notamment que la clef ne possède bien-entendu pas de période (de la forme `codecode` ou `catcatcat`). La preuve sera admise.

Fait 3 Pour tout entier k', les deux assertions suivantes sont équivalentes :

1. k' est un multiple de k.
2. pour tout entier $j \in [1, k']$ on a : $I(c(k', j), c(k', j), 0) \simeq 0,065$.

Fait 4 Pour tous entiers $j, j' \in [1, k]$, l'entier $d := k_{j'} - k_j$ est tel que :

$$I(c(k, j), c(k, j'), d) \simeq 0,065$$

preuve :
L'entier $I(c(k, j), c(k, j'), k_{j'} - k_j)$ est égal à $I(m(k, j) + k_j^l + (-k_j)^l, m(k, j') + k_{j'}^l, +(-k_{j'})^l, 0)$, à $I(m(k, j), m(k, j'), 0)$, donc à $0,0065$ (puisque $m(k, j)$ et $m(k, j')$ sont extraits d'un mot anglais).

Le fait précédent exhibe une propriété de la différence des décalages $k_{j'} - k_j$. Cette propriété est en fait une caractérisation comme l'établit le fait suivant. Sa preuve est omise.

Fait 5 Pour tout entier d, les deux assertions suivantes sont équivalentes :

1. $I(c(k, j), c(k, j'), d) \simeq 0,065$.
2. $d = k_{j'} - k_j$.

L'attaque sur un exemple

L'ouvrage de D. Stinson nous donne l'exemple du texte chiffré suivant :

```
CHREEVOAHMAERATBIAXXWTNXBEEOPHBSBQMQEQERBW
RVXUOAKXAOSXXWEAHBWGJMMQMNKGRFVGXWTRZXWIAK
LXFPSKAUTEMNDCMGTSXMXBTUIADNGMGPSRELXNJELX
VRVPRTULHDNQWTWDTYGBPHXTFALJHASVBFXNGLLCHR
ZBWELEKMSJIKNBHWRJGNMGJSGLXFEYPHAGNRBIEQJT
AMRVLCRREMNDGLXRRIMGNSNRWCHRQHAEYEVTQQEBBI
PEEWEVKAKOEWADREMXMTBHHCHRTKDNVRZCHRCLQOHP
WQAIIWXNRMGWOIIFKEE
```

L'algorithme d'attaque calcule au fur et à mesure les mots extraits $c(i,j)$ et teste la condition : $\forall j \in [1,i] I(c(i,j), c(i,j), 0) \simeq 0,065$.

Par exemple, le mot $c(1,1)$ est égal au mot c lui-même, le mot $c(2,1)$ est égal à, avant suppression des espaces blancs :

```
C R E O H A R T I X W N B E P B B M E E B
R X O K A S X E H W J M M K R V X T Z W A
L F S A T M D M T X X T I D G G S E X J L
V V R U H N W W T G P X F L H S B X G L H
Z W L K S I N H R G M J G X E P A N B E J
A R L R E N G X R M N N W H Q A Y V Q E B
P E E K K E A R M M B H H T D V Z H C Q H
W A I X R G O I K E
```

Le mot $c(2,1)$ est égal à, avant suppression des espaces blancs :

```
H E V A M E A B A X T X E O H S Q Q Q R W
V U A X O X W A B G M Q N G F G W R X I K
X P K U E N C G S M B U A N M P R L N E X
R P T L D Q T D Y B H T A J A V F N L C R
B E E M J K B W J N G S L F Y H G R I Q T
M V C R M D L R I G S R C R H E E T Q B I
E W V A O W D E X T H C R K N R C R L O P
Q I W N M W I F E
```

Le calcul des indices $I(c(i,j), c(i,j), 0)$ fournit les données suivantes :

	$j=1$	j=2	j=3	j=4	j=5
i=1	0,045				
i=2	0,046	0,041			
i=3	0,043	0,050	0,047		
i=4	0,042	0,039	0,046	0,040	
i=5	0,063	0,068	0,069	0,061	0,072

Conséquence du Fait 3, l'entier $i = 5$ a forte probabilité d'être la longueur de la clef. Cet entier est donc retourné comme longueur de clef.

Le calcul des indices $I(c(5,i), c(5,j), d)$ fournit les données suivantes :

i	j	d=0,1,...,25
1	2	$\in$[0,025 ;0,052] sauf $0,068$ pour $d = 9$
1	3	$\in$[0,024 ;0,056]
1	4	$\in$[0,027 ;0,055]
1	5	$\in$[0,024 ;0,054] sauf $0,070$ pour $d = 16$
2	3	$\in$[0,028 ;0,056] sauf $0,067$ pour $d = 13$
2	4	$\in$[0,022 ;0,055]
2	5	$\in$[0,029 ;0,054] sauf $0,080$ pour $d = 7$
3	4	$\in$[0,023 ;0,053]
3	5	$\in$[0,025 ;0,056] sauf $0,072$ pour $d = 20$
4	5	$\in$[0,024 ;0,055] sauf $0,061$ pour $d = 11$

Conséquence de ces calculs et du Fait 5, les constituants $(k_1, k_2, k_3, k_4, k_5)$ de la clef k ont de fortes probabilités de vérifier :

$$\begin{array}{ll} k_1 - k_2 = 9 & k_1 - k_5 = 16 \\ k_2 - k_3 = 13 & k_2 - k_5 = 7 \\ k_3 - k_5 = 20 & \\ k_4 - k_5 = 11 & \end{array}$$

On en déduit que k est de la forme $(k_1, k_1 - 9, k_1 - 22, k_1 - 5, k_1 - 16)$ c'est à dire de la forme $(k_1, k_1 + 17, k_1 + 4, k_1 + 21, k_1 + 10)$. La recherche du caractère x de plus grande fréquence dans les textes $c(5,1), c(5,2) + 9^*, c(5,3) + 22^*, c(5,4) + 5^*, c(5,5) + 16^*$ permet de calculer le décalage k_1 comme la différence $x -' e'$ (ici $k_1 = 9$).

La clef k est donc cassée ! Elle est égale à $(9, 0, 13, 4, 19)$ ou sous forme alphabétique à `JANET`. Pour obtenir le message chiffré, il suffit donc d'additionner sur chaque bloc l'opposé de k à savoir le mot `ranwh`. Le message en clair est ainsi, après réintroduction des signes de ponctuation :

> The almond tree was in tentative blossom. The days were longer, often ending with magnificent evenings of corrugated pink skies. The hunting season was over, with hounds and guns put away for six months. The wineyards were busy again as the well -organized farmers treated their wines and the more lackadaisical neighbors hurried to do the pruning they should have done in November.

2.2 Chiffrement par blocs

Dans cette section, nous présentons les cryptosystèmes à clefs secrètes utilisés de nos jours qui manipulent des mots binaires. Ces chiffrements procèdent en découpant le message m en blocs de taille constante. Ces cryptosystèmes se distinguent :

- par la longueur n des blocs.
- par le mode d'utilisation.
- par le système de chiffrement de chacun des blocs.

2.2.1 Mode d'utilisation

Chaque mode de chiffrement consiste :
- à découper le message en clair m en autant de blocs notés $m_1, \ldots, m_l$.
- à calculer l blocs $c_1, \ldots, c_l$.
- à concaténer les blocs $c_1, \ldots, c_l$ pour obtenir le mot chiffré c.

Il existe quatre modes principaux de chiffrement des blocs ainsi définis :

ECB Electronic CodeBook mode $c_i = e_k(m_i)$.

OFB Output Feedback mode $c_i = m_i + y_i$ où $y_i = e_k(y_{i-1})$ avec y_0 un bloc constant fourni.

CFB Cipher Feedback mode $c_i = e_k(c_{i-1}) + m_i$ avec c_0 un bloc constant fourni.

CBC Cipher BlockChaining mode $c_i = e_k(c_{i-1} + m_i)$ avec c_0 un bloc constant fourni.

Exemple 8 La Figure 2.3 présente les deux modes d'utilisation ECB et CFB. En haut de la figure apparaissent les modes de chiffrement, en bas ceux de déchiffrement.

Ces modes ont notamment pour propriété d'éviter la contagion d'une erreur de transmission à plus de deux blocs. Il est facile d'observer sur la Figure 2.3, qu'une altération du bloc c_{i-1} en $c_{i-1}^{\sim}$ altérera :
- sous le mode ECB l'unique bloc m'_{i-1}.
- sous le mode CFB les deux seuls blocs m'_{i-1} et m'_i. Le bloc m'_{i+1} restant égal à m_i.

Le mode de déchiffrement des modes OFB CBC est laissé en exercice.

2.2.2 Produit de deux cryptosystèmes

Une idée naturelle est de vouloir combiner deux cryptosystèmes en un nouveau ayant une sécurité supérieure aux deux premiers. Une solution souvent envisagée est de réaliser le *produit* de deux ou plusieurs cryptosystèmes.

Définition 1 (Produit) Soient deux cryptosystèmes $\Gamma_1 = (\mathbf{M}', \mathbf{C}', \mathbf{K}', \mathbf{E}', \mathbf{D}')$ et $\Gamma_2 = (\mathbf{M}'', \mathbf{C}'', \mathbf{K}'', \mathbf{E}'', \mathbf{D}'')$ tels que $\mathbf{C}' = \mathbf{M}''$, le *produit* de Γ_1 et Γ_2 est le cryptosystème $(\mathbf{M}', \mathbf{C}'', \mathbf{K}, \mathbf{E}, \mathbf{D})$ où :

K est le produit cartésien $\mathbf{K}' \times \mathbf{K}''$.

E est l'ensemble des fonctions de chiffrement de la forme $e_{(k',k'')}$ qui associe à tout message $m \in \mathbf{M}'$ le texte $e''_{k''}(e'_{k'}(m))$.

D est l'ensemble des fonctions de déchiffrement de la forme $d_{(k',k'')}$ qui associe à tout message $c \in \mathbf{C}''$ le texte $d'_{k'}(d''_{k''}(m))$.

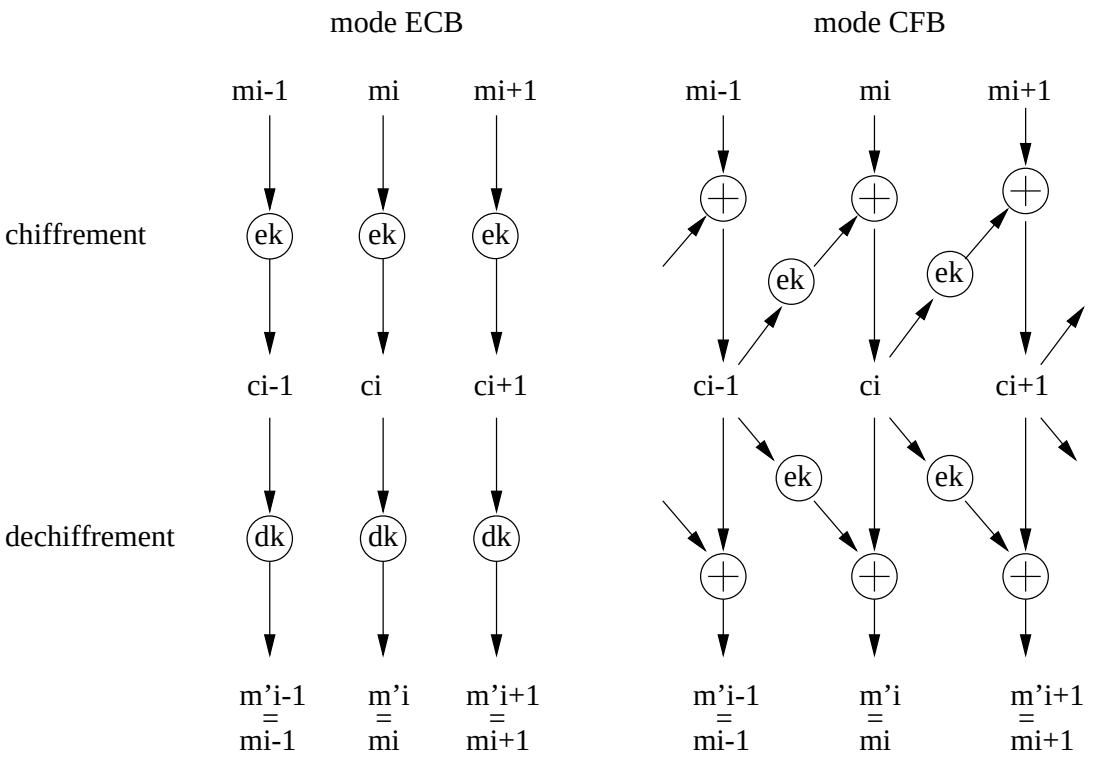

FIG. 2.3 – Les modes ECB et CFB

L'avantage immédiat d'un tel produit est que l'espace des clefs a pour cardinalité le produit des cardinalités des deux espaces de clefs initiaux. Une attaque exhaustive sur l'espace des clés devient alors beaucoup plus difficile.

Excepté cette remarque, on ne peut rien dire en général de la sécurité d'un tel système. Il n'est pas même établi que sa sécurité soit supérieure à celle de chacun des systèmes le composant.

2.3 Réseau de Feistel

Un grand nombre de cryptosystèmes à clefs secrètes ont une structure dite en "réseau de Feistel". L'intérêt de ceux-ci est de pouvoir à partir de n'importe quelle fonction f construire une fonction de chiffrement. Insistons sur le fait que f est quelconque et donc n'est pas nécessairement bijective. Un réseau de Feistel se définit à l'aide :

- d'une longueur de bloc paire notée $2 \cdot l$.
- d'un espace de clefs noté $\mathbf{K}$.
- d'un nombre de "rondes" notée r.
- d'un générateur g de sous-clefs qui associe à chaque clef $k \in \mathbf{K}$ une séquence de r *sous-clefs* $k_1, \ldots, k_r$. Ces clefs ont une longueur commune notée l'.

– d'une fonction f définie de $\{0,1\}^l \times \{0,1\}^{l'}$ dans $\{0,1\}^l$.

La fonction de chiffrement e_k associe à tout message $m \in \{0,1\}^{2 \cdot l}$ le message $e_k(m) := R_r L_r$ défini ainsi :

– L_0 et R_0 désignent les deux blocs de longueur l respectivement bloc gauche (Left en anglais) et bloc droit (Right en anglais).
– pour tout entier $i \in [1, r]$, le bloc L_i est l'ancien bloc droit R_{i-1}, le bloc R_i est obtenu selon le calcul $R_i = L_{i-1} + f(R_{i-1}, k_i)$.

La fonction de déchiffrement d_k est définie de façon identique à la seule exception que les sous clefs sont inversées.

Fait 6 Pour tout message m, on a : $d_k(e_k(m)) = m$.

preuve :
Soit $m \in \{0,1\}^{2 \cdot l}$ un message, $k \in \mathbf{K}$ une clef et $k_1, \ldots, k_r$ ses sous-clefs. Notons $L'_0 R'_0$ la décomposition en blocs de même longueur du message $e_k(m)$ et pour tout entier $i \in [1, r]$, notons $L'_i R'_i$ la décomposition en deux blocs issue de la i^{e}ronde lors du déchiffrement. Par définition, on a : $d_k(e_k(m)) = R'_r L'_r$.

Démontrons $d_k(e_k(m)) = m$ en démontrant $R'_r = L_0$ et $L'_r = R_0$ (on a : $m = L_0 R_0$). Pour cela, il nous suffit de démontrer par récurrence que pour tout entier $i \in [0, r]$ on a : $L'_i = R_{r-i}$ et $R'_i = L_{r-i}$.

L'égalité $e_k(m) = R_r L_r$ entraîne $L'_0 = R_r$ et $R'_0 = L_r$. La propriété est donc établie pour $i = 0$. Supposons la établie pour un entier $i \in [0, r[$:

– L'_{i+1} est par définition R'_i, égal par hypothèse à L_{r-i}, lui même égal par hypothèse à R_{r-i-1} c'est à dire $R_{r-(i+1)}$.
– R'_{i+1} est par définition $L'_i + f(R'_i, k_{r+1-(i+1)})$. Par hypothèse, $L'_i = R_{r-i}$, $R'_i = L_{r-i} = R_{r-i-1}$. Or par définition, $R_{r-i} = L_{r-i+1} + f(R_{r-i-1}, k_{r-i})$. On en déduit $R'_{i+1} = L_{r-i-1} + f(R_{r-i-1}, k_{r-i}) + f(R_{r-i-1}, k_{r-i}) = L_{r-(i+1)}$.

Exemple 9 La figure 2.4 présente une ronde dans un réseau de Feistel. La figure 2.5 présente un réseau de Feistel à deux rondes et le réseau de Feistel de déchiffrement.

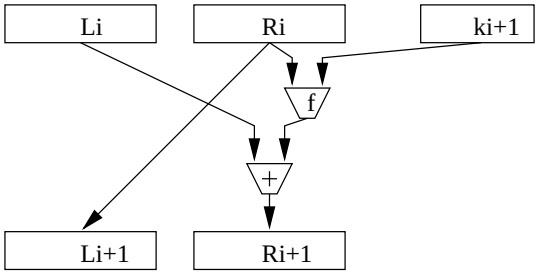

FIG. 2.4 – Une ronde de Feistel

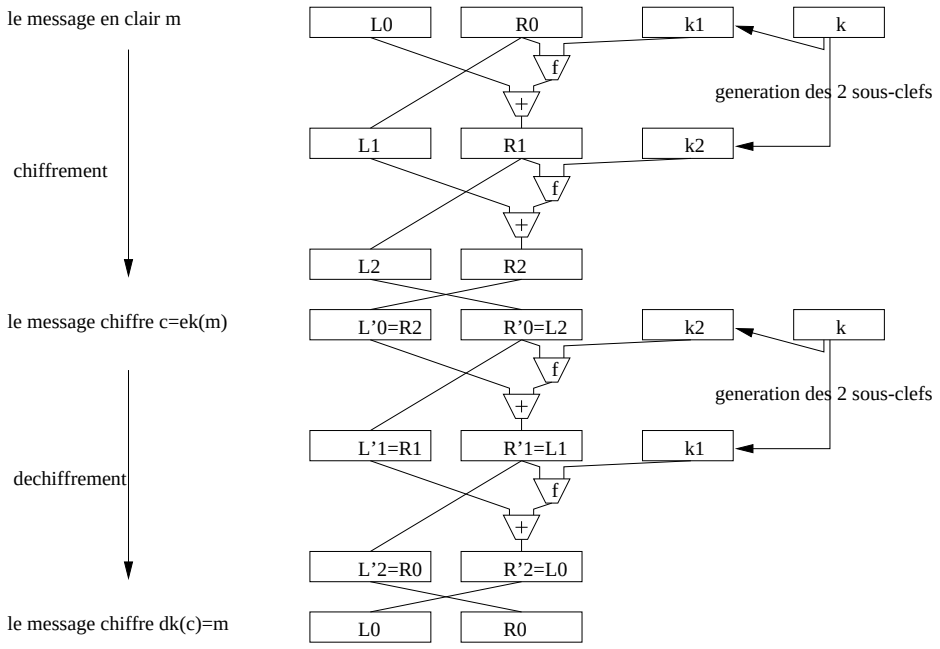

FIG. 2.5 – Un réseau de Feistel à deux rondes et son déchiffrement

Les propriétés des réseaux de Feistel sont multiples. Citons-en quelques unes :

1. la facilité de leur implémentation en "hard". La définition d'un réseau de Feistel est en fait celle d'un circuit électronique. La complexité de celui-ci dépend seulement du nombre de rondes et de la complexité du circuit générant les sous-clefs et implémentant la fonction f. Souvent, la fonction f peut être calculée en un cycle d'horloge, ainsi que chaque sous-clef.
2. la structure en cascade d'un tel réseau permet de chiffrer un nouveau message sans attendre le chiffrement du précédent. Ainsi, on peut chiffrer n messages $m_1, \ldots, m_n$ en $n+r = \Theta(n)$ cycles d'horloge : à chaque date t, les blocs R_0L_0, ..., R_rL_r correspondent aux chiffrements partiels des messages $m_t, m_{t-1}, \ldots, m_{t-r}$.

2.3.1 Un standard : le D.E.S et sa controverse

Le DES ou Data Encryption Standard est un standard de chiffrement établi par le NSA (institution US en charge de la sécurité) en collaboration avec IBM. Ce cryptosystème est un réseau de Feistel où :

1. la longueur des blocs de textes est 64.
2. le nombre de rondes est 16.
3. la clef k est de longueur 56. Ses sous-clefs sont de longueur 48.

La communauté cryptologique est demeurée longtemps sceptique au sujet de la sécurité d'un tel cryptosystème. Certains ont pensé que le NSA a affaibli volontairement le projet d'IBM afin d'y introduire une brèche secrète lui permettant d'attaquer ce système. La décision du NSA de réduire à 56 la longueur de la clef initialement proposée par IBM à 112 est en effet pour le moins surprenante. Les détails d'une telle controverse se trouveront dans tout bon ouvrage de cryptologie.

Ce cryptosystème encore très employé par plusieurs instituts financiers est considéré comme fragile¸ :

1. la longueur de la clef trop petite permet une attaque exhaustive sur l'espace des clefs (voir TD).
2. la littérature est riche d'attaques non exhaustives, donc beaucoup plus rapides, opposées au D.E.S. Citons-en quelques une :

   cryptanalyse différentielle

   cryptanalyse linéaire

   Ces attaques sont exposées notamment dans l'ouvrage D. Stinson.

2.3.2 D'autres cryptosystèmes

Comme nous l'avons déjà dit, il suffit de choisir une fonction f quelconque et un réseau du type de celui de Feistel, pour définir un cryptosystème. Seulement le définir. Prouver sa sécurité est alors bien plus compliqué ! Les cryptologues en herbe ou plus chevronnés n'ont pas manqué à cet exercice. Aussi, trouvera t-on dans la littérature pléthore de cryptosystèmes à clefs secrètes (à ce sujet lire Schneier ou Menezes). Comme alternative au DES, nous citerons le système IDEA qui a pour principales caractéristiques :

- la clef principale et chaque sous-clef est de longueur 128.
- la longueur des blocs est 64.
- le nombre de rondes est 8.
- sa structure est une variante du réseau de Feistel.

La longueur de la clef rend hors de portée toute attaque exhaustive : $2^{128} \simeq 10^{39}$ opérations exigerait à mille milliards d'ordinateurs opérant à un gigahertz de calculer pendant 30 milliard d'années (1 an $=3 \cdot 10^7$ secondes).

D'autres cryptanalyses ont été tentées : elles sont mises en échec dès que le nombre de rondes excède 5. Le lecteur est invité à se reporter à l'ouvrage de Schneier pour plus de détails.

Chapitre 3

Fonction à sens unique à brèche secrète ou la naissance de la cryptologie moderne

Si une révolution scientifique consiste à établir la caducité d'anciennes croyances, à apporter d'heureuses solutions à d'anciennes interrogations et à poser beaucoup plus de problèmes qu'elle n'en résoud, la cryptologie à clefs publiques est bien une révolution !

Cette découverte brise en effet l'enfermement dans lequel se trouvait la cryptologie et ses cryptosystèmes à clefs secrètes : dorénavant, Alice peut envoyer à Bob un message confidentiel sans partager avec lui un secret mais uniquement une information à son sujet que tout le monde connaît : sa clef publique.

Un autre dogme est mis à mal. Désormais la sécurité ne repose pas sur d'obscures tables élaborées dans des bureaux protégés d'institutions fédérales américaines mais reposera sur des problèmes connus de tous. Et il s'agit bien de tous : l'un des problème le plus fameux est la seule décomposition en facteurs d'un entier.

Dans ce chapitre, nous présenterons un objet mathématique au coeur de la cryptologie moderne : la fonction à sens unique à brèche secrète. Nous tenterons d'en donner une définition et de présenter quelques fonctions candidates.

Une révolution ne résoud pas seulement des problèmes, elle en pose de nouveaux. Nous verrons pourquoi cette définition bien que cohérente présente certaines limites.

3.1 Fonction à sens unique à brèche secrète

Au cours de ce chapitre, les objets manipulés par les fonctions seront des mots binaires de longueur quelconque. Parfois, ils représenteront des entiers sous la forme de leur codage binaire avec bit de poids faible (bit de parité) à gauche.

Définir ce qu'est une fonction à sens unique se fait ainsi :

Définition 2 (Fonction à sens unique) Une fonction *à sens unique* est une fonction $f : A \to B$ telle que :

1. le calcul de f, c'est à dire calculer $f(x)$ sachant x, est facile.
2. le calcul de f^{-1} est difficile.

Pour poser une telle définition, il nous faut définir les notions de "calcul" et les notions de "calcul facile". Celle-ci seront abordées dans la section suivante.

Définition 3 (Fonction à sens unique à brèche secrète) Une fonction à sens unique f admet pour *brèche secrète* une information k si

3. le calcul de f^{-1} sachant k est facile.

3.1.1 Utilisation d'une fonction à sens unique à brèche secrète

Toute fonction à sens unique à brèche secrète permet de construire très simplement un cryptosystème à clef publique. Ce protocole contient deux parties, une qui permet à Bob de fabriquer sa clef privée (le couple (f, k)), de diffuser sa clef publique (la fonction f), une seconde l'envoi d'un message de Alice à Bob.

```
Diffusion publique de la clef de Bob
ENTRÉE Bob :
SORTIE Bob : une clef privée (f, k)
SORTIE Pub.: une clef publique f

Bob     choisit une fonction à sens unique f et
            sa brèche secrète k;
        rend publique la fonction f ;

Envoi par Alice d'un message m à Bob
ENTRÉE Alice : la clef publique de Bob f et un message m
SORTIE Alice :
ENTRÉE Bob :   la clef privée de Bob (f, k)
SORTIE Bob :   m

Alice   calcule c := f(m) ;
        envoie c à Bob ;
Bob     calcule m' := f^{-1}(c) en utilisant k ;
        retourne m' ; % on a m' = m
```

3.1.2 Propriétés souhaitées

Dans cette section, nous formalisons davantage la définition d'une fonction à sens unique et présentons d'autres propriétés.

À sens unique

Définir formellement ce qu'est le calcul de f est évident, il s'agit de résoudre le problème suivant :

f
Entrée : x
Sortie : $f(x)$

Définir le calcul de f^{-1} peut-être ambigu. Du fait que f n'est pas nécessairement injectif, f^{-1} peut ne pas être une fonction mais une relation. Une définition du calcul de f^{-1} pourrait-être :

Entrée : y
Sortie : l'ensemble $\{x \mid f(x) = y\}$

Comme nous le verrons au travers de nombreux exemples, supposer difficile ce problème est insuffisant. Le problème que l'on souhaite rendre difficile est le calcul non de toutes les images inverses mais de simplement l'une d'entre elles. Il est ainsi formalisé :

f^{-1}
Entrée : y
Sortie : x tel que $f(x) = y$

À brèche secrète

Reprenant ce formalisme, définir ce qu'est le calcul de f^{-1} sachant k devient :

Entrée : y, k
Sortie : x tel que $f_k(x) = y$

À collisions difficiles

Une grande variété de fonctions à sens unique apparaissent en cryptologie. Elles se distinguent par les propriétés qu'elles sont (supposées ou non) vérifier. À titre d'exemple, citons-en deux :

à collision faible difficile si pour tout élément il est difficile de trouver un élément ayant même image c.a.d plus formellement de rendre le problème suivant difficile quel que soit x :

Entrée : x
Sortie : z tel que $z \neq x$ et $f(x) = f(z)$

à collision forte difficile il est difficile de calculer deux éléments distincts ayant même image c.a.d de rendre difficile le problème suivant :

Entrée :
Sortie : x, z tel que $z \neq x$ et $f(x) = f(z)$

3.2 Quelques rudiments en Théorie de la complexité

Nous avons vu que la définition d'une fonction à a sens unique nécessitait la notion de calcul, de calcul facile et de calcul difficile. Nous tenterons ici de répondre aux questions :

Qu'est-ce qu'un calcul ?

Qu'est-ce qu'un calcul facile ?

Qu'est-ce qu'un calcul difficile ?

3.2.1 Calculabilité

La réponse à la première question fait l'unanimité et a pour nom la "Thèse de Church". Cette réponse qui est le fondement de l'Informatique présente une définition de ce qui est calculable et donc de ce qui ne l'est pas. La croyance en cette thèse se fonde sur l'observation (démontrée) que tous les modèles de calculs les plus puissants jusqu'à aujourd'hui proposés sont équivalents entre eux. Ainsi, tout ce qui est calculable par ces modèles l'est, et inversement, par un algorithme écrit dans votre langage préféré que ce soit en `C`, en `Lisp` ou en `Prolog`.

Citons l'un de ces modèles de calcul remarquable pour la simplicité de sa définition : la machine de Turing. Une présentation formelle de ce modèle sort du cadre de ce cours ; pour plus de détails, consulter tout ouvrage traitant de la complexité ou de la calculabilité (Tel l'excellent ouvrage de Papadimitriou). Indiquons simplement que dans un tel modèle, toute entrée et toute sortie est codée à l'aide d'un mot binaire écrite sur un ruban de taille infinie, mot qui est délimité à l'aide d'un caractère spécial marquant la fin du mot. La taille d'une instance est ainsi la longueur de ce mot.

L'inverse de toute fonction calculable est elle-même calculable

Observons que dans la définition d'une fonction à sens unique la fonction f^{-1} est souhaitée "difficile à calculer" et non pas "impossible à calculer". La raison est simple : il n'existe pas de fonction f calculable dont l'inverse est incalculable. L'algorithme suivant en est la preuve :

```
ENTRÉE : y
SORTIE : x tel que f(x)=y

x := motvide() ;
tant que f(x)≠y faire
    x := motSuivant(x) ;

retourner(x) ;
```

Implémenter une fonction `motSuivant`, c'est à dire qui produit selon un ordre total l'ensemble des mots, ne présente aucune difficulté : il suffit de s'inspirer de la fonction d'incrémentation des entiers (un mot représente un entier et inversement).

3.2.2 Un peu de vocabulaire

En théorie de la complexité, en lieu et place du terme fonction, on préfère le terme plus général de "Problème" qui n'est autre qu'une relation. Voici un exemple de problème important en cryptologie :

```
décomposition
ENTRÉE : un entier x
SORTIE : un facteur propre de x
```

L'entrée est appelée une *instance* du problème.

Une *solution* à un problème est un algorithme prenant en entrée une instance x et fournissant en sortie une solution au problème ainsi instancié. Une solution algorithmique au problème précédent est fournie ci-dessous :

```
décomposition
ENTRÉE : un entier x
SORTIE : un facteur propre de x
i := 2 ;
tantque i < x faire
    si i est diviseur de x alors retourner i ;
    i := i + 1 ;
```

Problèmes décisionnels et fonctionnels

Certains problèmes sont remarquables. Ils ont pour noms *problèmes décisionnels* et n'admettent en sortie que des booléens. Les autres problèmes sont dits *fonctionnels*. Un exemple de problème décisionnel est le suivant :

```
ENTRÉE : un entier x
SORTIE : OUI si il existe y un facteur propre de x
```

Aussi, les problèmes décisionnels admettent deux sortes d'instances : celles dites *positives*, qui admettent OUI comme sortie et celles *négatives*. L'ensemble des instances positives est parfois appelé le *langage associé* au problème.

Souvent, un problème décisionnel P a pour langage l'ensemble des instances d'un problème fonctionnel F admettant une sortie. Sur l'exemple ci-dessus, l'ensemble des nombres non premiers L est l'ensemble des instances x associées par `décomposition` à une sortie y. Nous dirons que y est un *certificat* de x relativement au problème P.

En outre, nous dirons qu'un problème (resp. problème décisionnel) est *calculable* (resp. *décidable*) si il admet une solution algorithmique.

3.2.3 Calcul facile, algorithme efficace, problème facile

Nous allons lier ici la notion de complexité aux différents objets que sont :

1. les instances.
2. les algorithmes.
3. les problèmes.

Notons que si les expressions "calcul facile" et "problème facile" sont souvent employées, on rencontre rarement l'expression "algorithme facile". On emploie plutôt l'expression algorithme *efficace*. Ainsi, un problème est *facile* si il admet une solution algorithmique efficace.

La notion de facilité est liée aux deux ressources de tout modèle : le temps et l'espace nécessaires au calcul.

Apprécier la complexité du calcul d'une fonction f est réalisé en observant l'évolution des ressources consommées par le calcul des valeurs $f(x)$ en fonction de la complexité des entrées x. La complexité d'une entrée x est définie comme étant la longueur même du mot x et est notée $|x|$.

Un choix cohérent pour définir la notion de facilité est celui des algorithmes dits *en temps polynomial*, c'est à dire admettant une fonction polynomiale p telle que pour toute entrée x le nombre d'instructions élémentaires exécutées par l'algorithme sur l'entrée x est majorée par $p(|x|)$.

Une classe importante de problèmes est celle des problèmes polynomiaux, notée $\mathbf{P}$, définie comme l'ensemble des problèmes de décision à solution en temps polynomial. Un premier candidat de problèmes faciles est donc $\mathbf{P}$.

Cette candidature a pour grand mérite la simplicité de la définition de $\mathbf{P}$. Mais confrontée au monde réel, elle peut présenter certaines faiblesses :

- certains objectent que cette classe est trop large. Un temps de calcul en x^{10} étant pratiquement irréalisable. Ils ont raison.
- certains objectent que cette classe est trop restreinte. Il existe en effet des algorithmes efficaces en temps linéaire mais fournissant une réponse avec une très très faible marge d'erreur. Ils ont raison.

3.2.4 Problème difficile, "toujours" difficile

Après avoir défini (ou avoir tenté de définir) ce qu'est un problème facile, il semblerait immédiat de définir ce qu'est un problème difficile en le définissant comme un problème qui n'est pas facile. Cette tentation peut amener quelques déconvenues.

Car un problème qui n'est pas facile est un problème pour lequel il n'existe pas d'algorithme calculant efficacement pour toute entrée la sortie désirée. Un tel problème pourrait très bien admettre un calcul facile sur disons $99,99\%$ des entrées et ne pas admettre un tel calcul sur le $0,01\%$ restant. Or, en cryptologie on souhaiterait plutôt assurer une sécurité non pas dans $0,01\%$ des cas mais dans

99, 99% des cas (à défaut de 100% des cas). Les cryptologues recherchent des problèmes non pas seulement difficiles mais "toujours" difficiles. Toutefois, nous nous abstiendrons de définir formellement cette notion et adopterons la définition par défaut : est difficile ce qui n'est pas facile.

3.3 Quelques classes de problèmes

Nous avons précédemment défini la classe **P**, classe des problèmes décisionnels à solution en temps polynomial, comme étant un (premier) candidat à la classe des problèmes faciles.

Ici, nous présentons la classe **NP**. Cette classe est importante en cryptologie car elle contient tous les problèmes dont sont issues les fonctions à sens unique. L'intérêt de cette classe déborde la cryptologie et concerne toute l'informatique. En effet, un grand nombre de problèmes naturels sont **NP**.

Nous présenterons une sous-classe de problèmes de **NP**, la classe **NPC**, qui sera notre premier candidat pour la notion de difficulté.

3.3.1 La classe NP

Que ceci soit dit une bonne fois, la classe **NP** N'EST PAS la classe des problèmes à solution en temps non polynomial, bien au contraire. Une définition, très formelle, de laquelle est tiré le nom est la suivante : ensemble des problèmes décisionnels admettant une solution en temps polynomial sur une machine non-déterministe. Ainsi, le "**N**" signifie non-déterministe. Le "**P**" signifiant polynomiale. La définition, à ce moment là, est simple. Elle se corse fortement quand il s'agit de définir ce qu'est une machine non déterministe. Aussi nous ne le ferons pas. Nous dirons simplement que c'est une machine bien plus rapide que tout modèle de calcul naturel, que l'on pourrait imaginer comme un ordinateur pouvant se mettre dans une infinité d'états à la fois, c'est à dire qui serait capable de distribuer les calculs sur un nombre infini de processeurs et synchroniser les calculs instantanément !

La définition utilisée ici est, bien que formelle, assez simple :

Définition 4 (la classe NP) Un problème appartient à **NP** si il existe un problème R tel que :

1. toute instance du problème P est positive si et seulement si elle admet un certificat dans R.
2. le langage R appartient à **P**, c.a.d on vérifie en temps polynomial l'appartenance ou non de tout couple (x, y) à R.
3. R est *polynomialement équilibrée* c'est à dire qu'il existe un entier k tel que pour tout $(x, y) \in R$ on ait : $|y| \leq |x|^k$.

Ainsi, un problème est dans **NP** si il est facile de vérifier pour toute instance x et pour tout y, si y est un certificat de x. Un exemple célèbre de problème **NP** est le problème `3-coloriabilité` suivant (rappelons qu'un graphe (V, E) admet comme 3-coloriage une fonction $f : V \to [1, 3]$ si aucune arête de E n'a deux extrémités de même couleur) :

`3-coloriabilité`
`ENTRÉE : un graphe` G
`SORTIE : OUI si` G `admet un 3-coloriage.`

Fait 7 Le problème décisionnel `3-coloriabilité` appartient à **NP**.

preuve :
Ce problème P est dans **NP** car la relation R contenant tout couple de la forme (x, y) où x est une représentation d'un graphe 3-coloriable et y la représentation d'un 3-coloriage vérifie les 3 conditions souhaitées. La première condition est vérifiée. L'appartenance R à **P** est immédiate : il est facile d'écrire un algorithme prenant en entrée un graphe $x = (V, E)$ et une fonction $y : V \to [1, 3]$ et de vérifier en temps polynomial si aucune arête de E n'a deux extrémités de même couleur. La troisième condition (équilibre polynomial) est aussi évidente : car il ne pose aucune difficulté de coder un 3-coloriage y de x selon un mot dont la longueur soit une fonction polynomiale de celle du mot codant le graphe G.

LA conjecture P≠NP ?

Ayant observé qu'un grand nombre de problèmes étaient dans **NP**, il est naturel de se demander si ils possèdent une solution efficace, c'est à dire un algorithme en temps polynomial.

Malgré la mobilisation d'un très grand nombre de chercheurs, cette question reste sans réponse et demeure la conjecture la plus célèbre dans la communauté informatique. À défaut de réponse scientifique, des croyances s'établissent. Certains conjecturent l'inégalité :

Conjecture 8

$$\mathbf{P} \neq \mathbf{NP}$$

Cette absence de preuve de l'inégalité **P≠NP** est une mauvaise nouvelle pour les cryptologues à la recherche de problèmes difficiles (voir section 3.3.4). Ceux-ci se consolent en constatant l'absence de solutions efficaces connues à de tels problèmes. Ils parlent alors de sécurité conditionnelle (conditionnée à l'absence de tels algorithmes).

3.3.2 Réduction polynomiale

Garantir la sécurité d'un protocole ne se fait pas en établissant la difficulté de l'algorithme le cassant (souvent, le problème est dans **NP** et n'est qu'hy-

pothétiquement difficile), mais en comparant sa difficulté avec un autre problème que la communauté suppose difficile. Pour se doter d'un tel outil de comparaison, on définit la relation "aussi facile" en introduisant les "réductions polynomiales".

Si la notion de "facilité" est caractérisée par la classe **P**, il est naturel de définir la relation de comparaison "aussi facile" en utilisant la "réduction polynomiale". L'intuition consiste à indiquer qu'un problème P est aussi facile qu'un second problème Q si l'existence d'une solution efficace à Q entraîne l'existence d'une solution efficace à P. On dit alors que Q se réduit en P. Une définition naturelle consiste à définir la notion d'"oracle".

Définition 5 (oracle) Soit Q un problème. Un algorithme *ayant pour oracle* Q est un algorithme utilisant une instruction résolvant le problème Q et dont les complexités en temps et en espace sont supposées appartenir à $O(1)$.

Remarque 9 Dans la définition précédente, on suppose que l'instruction résolvant le problème Q termine toujours. Ainsi, si par exemple le problème Q est le problème décisionnel `3-coloriabilité`, l'instruction retournera `OUI` si l'instance est `3-coloriable` et `NON` sinon.

Définition 6 (Réduction polynomiale) Un problème P *se réduit polynomialement* en un problème Q si P admet une solution algorithmique en temps polynomial ayant pour oracle Q. Ceci est noté $P <_{\mathbf{P}} Q$ ou si le contexte ne présente aucun ambiguïté $P < Q$.
Les expressions "P est aussi facile que Q", "Q est aussi difficile que P" seront admises.

Exemple de réduction polynomiale

Il parfois aisé de comparer certains problèmes. L'exemple suivant illustre l'importance des problèmes décisionnels où l'on montre que certains problèmes fonctionnels sont aussi faciles que des problèmes décisionnels. Considérons par exemple le problème a priori plus difficile que `3-coloriabilité` qui consiste non pas à décider si un graphe est 3-coloriable mais à calculer un éventuel 3-coloriage :

`3-coloriage`
`ENTRÉE : un graphe` G
`SORTIE : un 3-coloriage de` G `si il en existe et NON sinon.`

Ces deux problèmes sont aussi faciles l'un que l'autre comme l'indique le fait suivant :

Fait 10 Chacun des deux problèmes `3-coloriage` et `3-coloriabilité` se réduit polynomialement en l'autre. C'est à dire :

$$\texttt{3-coloriabilité} <_{\mathbf{P}} \texttt{3-coloriage}$$

$$\texttt{3-coloriage} <_{\mathbf{P}} \texttt{3-coloriabilité}$$

preuve :
Clairement, `3-coloriabilité` se réduit polynomialement en `3-coloriage`. Inversement, supposons l'existence d'une instruction `est3coloriable` décidant si un graphe G fourni en entrée est 3-coloriable ou non. Supposons que cette instruction ait une complexité en temps et en espace $O(1)$.

La réduction polynomiale consiste ici à construire sous la forme d'un graphe un 3-coloriage. Observons qu'un 3-coloriage d'un graphe (V, E) n'est autre chose qu'une partition des sommets (A, B, C) en parties tel qu'aucun de ces parties ne contienne les deux extrémités d'une même arête. Associons à tout graphe G et à tout triplet d'ensemble de sommets deux à deux disjoints (A, B, C), le graphe $t(G, A, B, C)$ obtenu à partir de G en ajoutant trois nouveaux sommets 1, 2 et 3 deux à deux adjacents et en connectant le sommet 1 (resp. 2, 3) à chaque sommet de $B \cup C$ (resp. $A \cup C$, $A \cup B$). Appelons de tels graphes des graphes *transformés*.

Il est aisé d'observer qu'un graphe G est 3-coloriable si et seulement si il admet une partition de ces sommets (A, B, C) telle que $t(G, A, B, C)$ soit 3-coloriable.

Soit l'instruction `colorier(H,t,1)` qui transforme un graphe transformé H en connectant un de ses sommets $t \notin \{1, 2, 3\}$ aux sommets 2 et 3. Définissons similairement les instructions `colorier(H,t,2)` et `colorier(H,t,3)`. Clairement une telle instruction a une complexité en temps et en espace polynomiale en la taille de H. Soit l'algorithme suivant :

```
calcul3coloriage
ENTRÉE : un graphe G
SORTIE : un 3-coloriage de G si il en existe, NON sinon

si est3coloriable(G)=NON
    retourner NON ;

H := le graphe obtenu à partir de G en ajoutant la clique {1,2,3} ;

pour tout sommet t de G faire
    si est3coloriable(colorier(H,t,1))
        H := colorier(H,t,1) ;
    sinon si est3coloriable(colorier(H,t,2))
        H := colorier(H,t,2) ;
    sinon
        H := colorier(H,t,3) ;

A := ensemble des sommets de G adjacents à 1 ;
B := ensemble des sommets de G adjacents à 2 ;
C := ensemble des sommets de G adjacents à 3 ;

retourner(A,B,C) ;
```

Il est aisé de démontrer qu'un tel algorithme est correct, c'est à dire qu'il est une solution au problème du 3-coloriage. Conséquence des considérations précédentes, du fait que le graphe transformé final H a une taille polynomialement bornée par celle de G (on ajoute 3 sommets et $2 \cdot n$ arêtes) et du fait que chaque instruction est de complexité en temps polynomialement bornée par la taille de H, on en déduit que la complexité en temps de `calcul3coloriage` est polynomialement bornée par la taille de l'entrée G. Insistons que cette complexité polynomiale repose sous l'hypothèse (folle ?) d'une complexité en temps $O(1)$ de l'instruction `est3coloriable`.

En conclusion, nous avons bien `3-coloriage`$<_{\mathbf{P}}$`3-coloriabilité`.

Faux amis

Voici trois problèmes importants en cryptologie. Les deux premiers sont décisionnels, le troisième fonctionnel.

`primalité`
`ENTRÉE : un entier` n
`SORTIE : OUI si` n `est premier`

`coprimalité`
`ENTRÉE : un entier` n
`SORTIE : OUI si` n `n'est pas premier`

`factorisation`
`ENTRÉE : un entier de la forme` $p \cdot q$ `avec` p `et` q `premiers.`
`SORTIE :` p, q

Malgré la proximité de ces problèmes, la communauté scientifique s'interroge sur l'existence d'une réduction polynomiale de `factorisation` en `primalité`. En d'autres termes, il n'est pas établi que `factorisation` soit aussi facile que `primalité` ou aussi facile que `coprimalité`.

En effet, les problèmes `primalité` et `coprimalité` admettent des solutions en temps polynomiale avec marge d'erreur. Ces solutions algorithmiques qui admettent une très très faible marge d'erreur fournissent des programmes effectifs efficaces. De plus, des chercheurs indiens viennent de trouver (mai 2002) un algorithme en temps polynomial selon une puissance 12 résolvant `primalité`. Ce joli résultat théorique risque (restons prudent !) d'avoir peu d'impact logiciel : la puissance 12 étant trop élevée.

À l'opposé les seuls algorithmes connus solutionnant `factorisation` sont *superpolynomiaux* (ayant une complexité en temps supérieure à toute fonction polynomiale). On ne sait pas s'il en existe de polynomiaux en temps (avec marge d'erreur ou pas). Cette ignorance a un mérite : elle est le fondement de la sécurité de nombreux protocoles cryptologiques.

3.3.3 La classe NPC ou la classe des problèmes aussi difficiles que tout problème de NP

Résultat important en théorie de la complexité, il existe dans la classe **NP** des problèmes aussi difficiles que chacun des problèmes de **NP**. La preuve de ce fait est assez courte mais sort du cadre du cours. Elle pourra être lue dans l'ouvrage de Papadimitriou (p 171).

Fait 11 Il existe dans **NP** des problèmes Q aussi difficile que tout problème de **NP** (en lesquels tout problème de **NP** se réduit). Cet ensemble de problèmes dits NPComplets est noté **NPC**. Il contient notamment `3-coloriabilité`.

L'existence de cette classe **NPC** est une bonne nouvelle pour les cryptologues à la recherche d'une "bonne" fonction à sens unique, c'est à dire facile à calculer mais difficile à inverser. Cependant, des interrogations et des réserves subsistent :

1. est-ce que **NP** contient des problèmes difficiles. A t-on : **P**$\neq$**NP** ?
2. en supposant **P**$\neq$**NP**, existe t-il des problèmes "toujours difficiles" (voir section 3.2.4). Quels sont-ils ? En existe t-il dans **NPC** ?

L'histoire de la cryptologie est de ce point de vue édifiante. L'un des premiers problèmes à avoir fourni des protocoles est un problème NPComplet dit du "Sac à dos". Malgré sa NPComplétude, la plupart des protocoles ont été cassés par des attaques en temps polynomial. Le premier problème à avoir fourni des protocoles non cassés est le problème de factorisation dont on ne sait pas si il est NPComplet.

3.3.4 Une tentative de définition formelle d'une fonction à sens unique

Nous fournissons une définition formelle de la fonction à sens unique. Cette définition présente l'avantage d'une certaine simplicité. Cette définition est la conséquence de deux choix :

1. caractérisation de la notion de la facilité par **P**.
2. caractérisation de la notion de la difficulté par le complémentaire de **P**.

Nous avons déjà longuement évoqué les avantages et inconvénients de ces caractérisations. Ces mêmes observations s'étendant à une telle définition des fonctions à sens unique.

Il est facile d'associer à tout problème décisionnel $P \in$ **NP** un problème fonctionnel F. Soit R une relation de **P** polynomialement équilibrée pour laquelle toute instance x de P est positive si et seulement si elle admet un certificat y relativement à R. Le problème F associé à P et R est le problème suivant :

`ENTRÉE :` x `une instance de` P
`SORTIE : un certificat` y `de` x `si il existe`

Ceci permet de définir la classe **FNP** des problèmes fonctionnels associés aux problèmes de **NP**, ainsi que la sous-classe **FP** qui contient les problèmes fonctionnels à solution algorithmique en temps polynomial. Ainsi, une fonction à sens unique peut se définir comme une fonction f telle que :

1. f appartient à **FP**
2. f^{-1} n'appartient pas à **FP**
3. f et f^{-1} sont polynomialement équilibrés.

Il est facile de démontrer que l'inverse d'une fonction à sens unique est "coincé" entre les classes **FNP** et **FP**.

Fait 12 Si f est une fonction à sens unique, alors son inverse f^{-1} appartient à **FNP-FP**.

preuve :
Soit f une fonction à sens unique associé à un problème P et à une relation R. La relation $\{(x, y) \mid y \text{ certificat de } x\}$ est par définition dans **P** (on vérifie en temps polynomial si pour un couple (x, y) donné, y est un certificat de x). La relation inverse $\{(y, x) \mid y \text{ certificat de } x\}$ appartient à **P**. Du fait que f^{-1} est polynomialement bornée (un polynôme en la longueur de $|y|$ borne la longueur $|x|$), f^{-1} appartient à **NP**.

L'intérêt du résultat précédent est que l'existence d'une fonction nécessite que $\mathbf{FNP} - \mathbf{FP} \neq \emptyset$, ce qui équivaut à démontrer que $\mathbf{NP} - \mathbf{P} \neq \emptyset$. Or la cryptologie à clefs publiques requiert davantage en nécessitant une fonction à sens unique à brèche secrète qui est une hypothèse plus forte que **P**≠**NP**. En d'autres termes, sa démonstration ou son infirmation sera au moins aussi difficile. Ce fait est exprimé ici :

Fait 13 *Si il existe une fonction à sens unique à brèche secrète alors* $\mathbf{P} \neq \mathbf{NP}$.

3.4 Quelques problèmes fournissant des fonctions à sens uniques à brèche secrète

Nous présentons ici quelques problèmes de **NP** supposés ne pas appartenir à **P**. Ces problèmes ou d'autres qui en sont dérivés fournissent des fonctions à sens unique à brèche secrète. Certaines de ses fonctions, associées aux problèmes `factorisation` et `logarithmeDiscret`, seront définies en TD.

3.4.1 Issus du problème de factorisation

La célébrité de ces problèmes est étroitement lié au succès du protocole de chiffrement issu de l'un deux et qui a pour nom RSA, du nom de ses trois inventeurs

Rivest, Shamir, Adleman. Sa célébrité est liée à la simplicité d'un des problèmes sous-jacents : la multiplication. Rappelons le problème de factorisation :

factorisation
ENTRÉE : $n = p \cdot q$ produit de deux nombres premiers
SORTIE : p, q

Ce problème fournit une fonction à sens unique mais ne fournit pas directement de fonction à sens unique et à brèche secrète. Pour obtenir de telles fonctions, il suffit de considérer les problèmes associés :

résiduositéQuadratique
ENTRÉE : un entier $n = p \cdot q$ où p et q sont premiers et distincts et un entier x.
SORTIE : OUI si x est un résidu quadratique modulaire selon n

racineCarréeModulaire
ENTRÉE : un entier $n = p \cdot q$ où p et q sont premiers et distincts et un résidu quadratique modulaire x selon n.
SORTIE : b tel que $x = b^2 \, mod \, n$

RSA
ENTRÉE : un entier $n = p \cdot q$ où p et q sont premiers et distincts, un entier $e > 0$ premier avec $(p-1) \cdot (q-1)$ et un entier c.
SORTIE : m tel que $c = m^e \, mod \, n$

3.4.2 Issus du problème de Logarithme discret

Nous présentons ici deux problèmes supposés difficiles liés à la notion de logarithme discret. Ces notions seront davantage développés dans le cours suivant. Disons simplement :

- Z_p^* désigne ici le groupe multiplicatif $(E, \cdot)$ formé des nombres premiers avec p. Dans le cas où p est premier E peut être considéré comme égal à $[1, p-1]$.
- un générateur g de Z_p^* est ainsi un entier qui permet de générer tous les éléments c'est à dire plus formellement un entier de Z_p^* tel que $\{g^i (mod \, p) \mid i > 0\} = Z_p^*$.

Voici deux problèmes parmi les plus célèbres en cryptologie :

logarithmeDiscret
ENTRÉE : un entier premier p, un générateur g de Z_p^* et $y \in Z_p^*$
SORTIE : un entier e tel que $g^e \, mod \;\; p \;\; = \;\; y$

DiffieHellman
ENTRÉE : un entier premier p, un générateur g de Z_p^* et deux entiers $g^a \, mod \;\; p$ et $g^b \, mod \;\; p$.
SORTIE : l'entier $g^{a \cdot b} \, mod \;\; p$

3.5 Conclusion

Sécurité calculatoire

Nous avions commencé ce chapitre en fondant la sécurité des protocoles sur des considérations calculatoires : cette sécurité repose non sur l'impossibilité mais sur la difficulté de réaliser certains calculs (ici inverser une fonction à sens unique).

Sécurité calculatoire hypothétique

Nous avons conclu ce chapitre par une incertitude visant l'existence ou non de telles fonctions à sens unique. Les fonctions présentées dans la dernière section sont supposées être à sens unique. Cette hypothèse n'est qu'une hypothèse : aucun début de commencement de preuve ne peut l'établir.

La communauté scientifique est à la recherche permanente de nouvelles fonctions supposées à sens uniques. Ainsi la fonction RSA, découverte en 1978, est cassable si l'on considère des entiers de 512 bits (155 chiffres) ; les normes de sécurité préconisent 768, 1024 voire 2048 bits pour une sécurité à plus long terme. La communauté s'intéresse ainsi davantage au problème du Logarithme Discret (1985) relativement à un groupe un peu plus complexe que Z_p^*, celui des courbes elliptiques (1987). La taille des objets assurant une bonne sécurité n'est alors pas quelques centaines de bits mais seulement quelques dizaines. Mais les chercheurs sont sur la brèche ! Aussi, faut-il s'attendre à régulièrement augmenter la taille des clefs voire abandonner certains chiffrements, comme RSA qui semble aujourd'hui trop fragile.

Chapitre 4

Quelques problèmes arithmétiques faciles

Le chapitre précédent nous a présenté le besoin des cryptologues en fonctions à sens unique à brèche secrète. La science dans laquelle ils ont puisé abondamment est celle de la Théorie des nombres. Nous présentons dans ce chapitre quelques uns de ses rudiments.

4.1 Quelques rudiments en Théorie des Nombres

4.1.1 Quelques généralités sur les groupes

Un groupe est un couple $(E, \cdot)$ composé d'un ensemble et d'une opération associative, possédant un élément neutre 1_E et pour laquelle tout élément $a \in E$ admet un élément *inverse*, c'est à dire un élément b tel que $a \cdot b = e$. Un *anneau* est un groupe muni d'une seconde opération associative qui admet un élément neutre et qui est distributive.

Nous considérerons désormais des groupes finis. Ce qui nous permet de définir l'*ordre* d'un groupe fini comme sa cardinalité.

Il est facile d'observer que tout élément e engendre un sous-groupe. Pour tout entier n et tout élément $a \in E$, nous noterons a^n l'élément défini récursivement par $a^n = 1_E$ si $n = 0$ et $a^n = a \cdot a^{n-1}$ sinon.

Fait 14 Pour tout groupe fini $(E, \cdot)$ et tout élément e, il existe un entier $n > 0$ tel que $e^n = 1_E$.

preuve :
Du fait que E est fini, il existe deux entiers non nuls p et q tels que $e^p = e^{p+q}$, c'est à dire tels que : $e^p = e^p \cdot e^q$. L'unicité de l'élément neutre entraine $e^q = 1_E$.

Conséquence quasi immédiate de ce fait :

Fait 15 Pour tout groupe fini $(E, \cdot)$ et tout élément e, le plus petit entier $n > 0$ vérifiant $e^n = 1_E$ est tel que $\{1_E, \ldots, e^{n-1}\}$ forme un sous groupe de E ayant n éléments deux à deux distincts.

preuve :
La preuve est laissée au lecteur.

La propriété précédente nous permet de définir pour tout groupe fini $(E, \cdot)$ et tout élément $e \in E$:

- le *sous-groupe engendré de e* comme étant le groupe ayant pour éléments $1_E, e, e^2, \ldots$.
- l'*ordre* de e comme étant l'ordre du sous-groupe qu'il engendre.

Nous dirons en outre qu'un groupe est *cyclique* si il possède un générateur. Une propriété que nous admettrons lie ces différentes notions avec la notion de diviseur.

Fait 16 (Théorème de Lagrange) L'ordre de tout élément d'un groupe fini divise l'ordre du groupe.

4.1.2 Primalité

Un entier p est un *diviseur* (on dit aussi *facteur*) d'un entier q, noté $p \mid q$, si il existe un entier r tel que $q = p \cdot r$. Il s'agit d'un diviseur *propre* si p est différent de 1 et de q. Ceci nous permet de définir pour tous nombres a et b non nuls tous les deux leur plus grand diviseur commun, noté $pgcd(a, b)$.

Présentons deux caractérisations fort utiles du *pgcd* :

Fait 17 La fonction *pgcd* est égale à la fonction f définie récursivement par :

$$\forall (a,b) \neq (0,0) f(a,b) := \begin{cases} a & \text{si } a = b \\ f(a-b,b) & \text{si } a > b \\ f(b,a) & \text{sinon} \end{cases}$$

preuve :
Il est aisé de démontrer que la fonction *pgcd* vérifie les trois propriétés :

1. $\forall a \neq 0\ pgcd(a,a) = a$
2. $\forall (a,b) \neq (0,0)\ pgcd(a,b) = pgcd(b,a)$
3. $\forall b < a\ pgcd(a,b) = pgcd(a-b,b)$

La fonction f définie plus haut a pour domaine de définition $\mathbb{N}^2 \backslash (0,0)$: la démonstration se fait par simple récurrence selon la taille $|(a,b)| = 2 \cdot a + b$. De ces deux remarques, on déduit que les fonctions f et *pgcd* sont égales.

Fait 18 (Identité de Bézout) Pour tout couple d'entiers a et b, on a :

$$pgcd(a,b) = min\{\alpha \cdot a + \beta \cdot b > 0 \mid \alpha \in \mathbb{Z}, \beta \in \mathbb{Z}\}$$

preuve :
Démontrons cette égalité en effectuant une récurrence selon la taille $|(a,b)| = ||a|-|b||$. La propriété est trivialement vérifiée pour les couples d'entiers de taille -1. Supposons l'existence d'un entier n tel que tout couple d'entiers (a,b) de taille $< n$ vérifie cette égalité. Soit (a,b) un couple d'entiers de taille n. Deux cas apparaissent :

1. $|(a,b)| = 0$.
   Il vient $a = b$. L'égalité $pgcd(a,a) = min\{\alpha \cdot a + \beta \cdot a > 0 \mid \alpha \in \mathbb{Z}, \beta \in \mathbb{Z}\}$ est immédiate et suffit à conclure.
2. $|(a,b)| \neq 0$.
   On en déduit $a \neq b$. Supposons $a > b$. D'après le Fait 17, $pgcd(a,b)$ est égal à $pgcd(a-b,b)$. Or la taille $|(a-b,b)|$ est strictement inférieure à celle de (a,b), c'est à dire n. Par récurrence, on en déduit $pgcd(a,b) = min\{\alpha \cdot (a-b) + \beta \cdot b > 0 \mid \alpha \in \mathbb{Z}, \beta \in \mathbb{Z}\}$ qui n'est autre que $min\{\alpha \cdot a + \beta \cdot b > 0 \mid \alpha \in \mathbb{Z}, \beta \in \mathbb{Z}\}$ (les ensembles $\{\alpha \cdot (a-b) + \beta \cdot b \mid \alpha \in \mathbb{Z}, \beta \in \mathbb{Z}\}$ et $\{\alpha \cdot a + \beta \cdot b \mid \alpha \in \mathbb{Z}, \beta \in \mathbb{Z}\}$ sont clairement égaux ainsi donc que les ensembles $\{\alpha \cdot (a-b) + \beta \cdot b > 0 \mid \alpha \in \mathbb{Z}, \beta \in \mathbb{Z}\}$ et $\{\alpha \cdot a + \beta \cdot b > 0 \mid \alpha \in \mathbb{Z}, \beta \in \mathbb{Z}\}$). Ce qui suffit à conclure. Conséquence de l'égalité $pgcd(a,b) = pgcd(b,a)$, le cas $b > a$ est traité de façon analogue.

Deux entiers a et b sont *premiers entre eux*, noté $a \wedge b$, si leur pgcd vaut 1. L'identité de Bézout permet de caractériser de telles paires d'entiers :

Fait 19 Pour tout couple d'entiers a et b, les deux assertions suivantes sont équivalentes :

1. a et b sont premiers.
2. il existe deux entiers relatifs α et β tels que $\alpha \cdot a + \beta \cdot b = 1$.

preuve :
Conséquence immédiate du Fait 18.
Un entier > 1 est *premier* si il ne possède pas de facteur propre. Un entier est *composite* si il n'est pas premier.

4.1.3 L'anneau $(\mathbb{Z}_n, +_n, \cdot_n)$ et le groupe $(\mathbb{Z}_n^*, \cdot_n)$

La première structure considérée est l'anneau $(\mathbb{Z}_n, +_n, \cdot_n)$ où :
- pour tout entier $n > 0$, le terme $\mathbb{Z}_n$ désigne l'intervalle $[0, n-1]$.
- $+_n$ désigne l'addition modulaire $a +_n b = (a+b) \, mod \, n$.
- $\cdot_n$ désigne la multiplication modulaire $a \cdot_n b = (a \cdot b) \, mod \, n$.

L'*inverse* d'un nombre a modulo un nombre n est, si il existe, l'unique entier α de $[0, n-1]$ à satisfaire l'équation : $1 = a \cdot_n \alpha$. Cet entier est noté $a^{-1} \, mod \, n$. Certains des éléments de $\mathbb{Z}_n$ ne sont pas inversibles. L'identité de Bézout fournit une caractérisation de ceux inversibles :

Fait 20 Pour tout entier n et tout entier a, les deux assertions suivantes sont équivalentes :

1. a est inversible modulo n.
2. a et n sont premiers entre eux.

preuve :
Conséquence immédiate du Fait 18.

Ce qui nous permet de définir la deuxième structure considérée. Pour tout entier $n > 0$, nous notons $(\mathbb{Z}_n^*, \cdot_n)$ le groupe ayant pour domaine $\mathbb{Z}_n^*$ l'ensemble des entiers de $[1, n-1]$ premiers avec n. Observons que si n est lui-même premier, $\mathbb{Z}_n^* = [1, n-1]$.

notation

Si le contexte le permet, les opérations $+_n$ et $\cdot_n$ seront notées plus simplement $+$ et $\cdot$.

Un théorème fondamental permet de décrire la structure de tels ensembles :

Fait 21 (Théorème des restes chinois) Soit un entier m égal au produit de nombres m_1,...,m_l premiers deux à deux entre eux. Pour tous entiers $x_1 < m_1, \ldots, x_l < m_l$, le système d'équations :

$$\begin{aligned} x_1 &= x \bmod m_1 \\ \ldots & \\ x_l &= x \bmod m_l \end{aligned}$$

a pour unique solution

$$x := \left(x_1 \frac{m}{m_1} y_1 + \ldots + x_l \frac{m}{m_l} y_l \right) \bmod m \text{ où } y_i = \left(\frac{m}{m_i} \right)^{-1} \bmod m_i \text{ pour tout } i \in [1, l].$$

preuve :
Il est facile d'observer que la solution proposée vérifie $x \bmod m_1 = x_1$ car :

- $x_1 \frac{m}{m_1} y_1 \bmod m_1$ vaut $\left((x_1 \cdot \left(\frac{m}{m_1} \right) \bmod m_1 \cdot y_1) \right) \bmod m_1$ c'est à dire x_1 puisque $x_1 < m_1$ et $\left(\frac{m}{m_1} \cdot y_1 \right) \bmod m_1 = 1$.
- si $i \neq 1$, $x_1 \frac{m}{m_i} y_i \bmod m_1$ vaut 0 car $\frac{m}{m_i}$ est un multiple de m_1.

De façon similaire, on établit pour tout $i \in [1, l]$ l'égalité : $x \bmod m_i = x_i$.

Pour démontrer l'unicité, il suffit d'observer que tout élément de $[0, m-1]$ est solution d'au plus un système d'équations (l'entier $x \bmod m_i$ est unique !) et donc d'exactement un système d'équations, puisque la cardinalité du nombre d'équations, égal au produit $m_1 \cdot \ldots \cdot m_l$, est égal à la cardinalité de $[0, m-1]$,

c'est à dire m.

Corollaire de ce théorème :

Fait 22 Pour tout produit m de nombres $m_1, \ldots, m_l$ deux à deux premiers, nous avons :

$$(\mathbb{Z}_m, +, \cdot) \text{ est isomorphe à } (\mathbb{Z}_{m_1}, +, \cdot) \times \ldots \times (\mathbb{Z}_{m_l}, +, \cdot)$$

$$(\mathbb{Z}_m^*, \cdot) \text{ est isomorphe à } (\mathbb{Z}_{m_1}^*, \cdot) \times \ldots \times (\mathbb{Z}_{m_l}^*, \cdot)$$

preuve :
Soit f la fonction qui à tout entier x associe $(x \; mod \; m_1, \ldots, x \; mod \; m_l)$ noté $(x_1, \ldots, x_l)$. Démontrons que :

- f est un morphisme de $(\mathbb{Z}_m, +_m)$ dans $(\mathbb{Z}_{m_1}, +_{m_1}) \times \ldots \times (\mathbb{Z}_{m_l}, +_{m_l})$.
  Simple conséquence de l'égalité $(a \; mod \; (b \cdot c)) \; mod \; c = a \; mod \; c$ pour tous entiers a, b, c et du fait que pour tous entiers x, y et m_i, on a : $(x +_m y) \; mod \; m_i = x \; mod \; m_i +_{m_i} y \; mod \; m_i$.
- f est un morphisme de $(\mathbb{Z}_m, \cdot_m)$ dans $(\mathbb{Z}_{m_1}, \cdot_{m_1}) \times \ldots \times (\mathbb{Z}_{m_l}, \cdot_{m_l})$.
  Simple conséquence du fait que pour tous entiers x et y et tout entier m_i, on a : $(x \cdot_m y) \; mod \; m_i = x \; mod \; m_i \cdot_{m_i} y \; mod \; m_i$.
- f est un isomorphisme de $(\mathbb{Z}_m, +, \cdot)$ vers $(\mathbb{Z}_{m_1}, +, \cdot) \times \ldots \cdot (\mathbb{Z}_{m_l}, +, \cdot)$.
  Les deux points précédents montrent que f est un morphisme. Conséquence du Théorème des Restes Chinois et de l'unicité de la solution du système d'équations, f est injectif. Conséquence de l'égalité des cardinalités des ensembles départ et image, égales à $m = m_1 \cdot \ldots \cdot m_l$, f est bijectif.
- f est un isomorphisme de $(\mathbb{Z}_m^*, \cdot)$ vers $(\mathbb{Z}_{m_1}^*, \cdot) \times \ldots \cdot (\mathbb{Z}_{m_l}^*, \cdot)$.
  Démontrons que f est à valeurs dans $(\mathbb{Z}_{m_1}^*, \cdot) \times \ldots \cdot (\mathbb{Z}_{m_l}^*, \cdot)$. Si x est premier avec m, il existe deux entiers relatifs α et β tels que $\alpha \cdot x + \beta \cdot m = 1$. Le reste $x_1 := x \; mod \; m_1$ admet un entier c_1 tel que $x = c_1 \cdot m_1 + x_1$. Il en découle $\alpha \cdot x_1 + (c_1 + \beta \cdot \frac{m}{m_1}) \cdot m_1 = 1$. D'après le Théorème de Bézout, $x \; mod \; m_1$ est premier avec m_1. Des résultats similaires pour tous les restes $x \; mod \; m_i$ avec $i \in [1, l]$ suffisent à conclure.

  Pour les mêmes raisons que dans le cas traité précédemment, on démontre que f est un morphisme et est injectif.
  Pour conclure, démontrons démontrer que f est surjectif de $(\mathbb{Z}_m^*, \cdot)$ vers $(\mathbb{Z}_{m_1}^*, \cdot) \times \ldots \cdot (\mathbb{Z}_{m_l}^*, \cdot)$. Pour cela, il suffit de démontrer que si pour tout $i \in [1, l]$, le reste x_i est premier avec m_i, alors x est premier avec m. Supposons $pgcd(x, m) \neq 1$. L'entier $pgcd(x, m)$ est un diviseur de x et m. Du fait que f est un morphisme multiplicatif, $f(pgcd(x, m)) = (k_1, \ldots, k_l)$ est un diviseur de $f(x)$ et de $f(m)$ (pour tout $i \in [1, l]$, k_i est un diviseur de x_i et m_i). L'égalité $pgcd(x_i, m_i) = 1$ entraine $f(k) = (1, \ldots, 1)$ et donc $pgcd(x, m) = 1$. Ainsi x et m sont premiers.

4.1.4 La fonction d'Euler

La fonction d'Euler est la fonction φ qui associe à tout entier $n > 0$ le nombre d'entiers $< n$ premiers avec n (c'est à dire la cardinalité de $\mathbb{Z}_n^*$). Conséquence du Fait 22, pour tout entier $n > 0$ nous avons :

- $\varphi(n) = n - 1$ si n est premier.
- $\varphi(n) = (n_1 - 1) \cdot \ldots \cdot (n_l - 1)$ si n est le produit de nombres premiers et distincts $n_1, \ldots, n_l$.

On en déduit :

Fait 23 (Théorème d'Euler) Pour tout entier n et tout élément entier $x \in \mathbb{Z}_n^*$, on a :

$$1 = x^{\varphi(n)} \; mod \, n$$

preuve :
Conséquence directe de la définition de φ et du Théorème de Lagrange (Fait 16).

Pour justifier la correction du fonctionnement du cryptosystème RSA, nous avons besoin d'une extension du fait précédent que nous admettrons :

Fait 24 Pour tout entier n sans facteur carré, pour tous entiers x et k on a :

$$x = x^{1+k \cdot \varphi(n)} \; mod \, n$$

4.2 Quelques problèmes faciles de P

L'objet de cette section est de recenser quelques problèmes fréquemment rencontrés dans la manipulation des grands entiers et pour lesquels on connaît des solutions algorithmiques efficaces.

Efficace signifie des algorithmes ayant une complexité en temps polynomiale en la taille des entrées $|x|$. On souhaite même davantage : le plus haut degré doit être petit : 0, 1, 2 ou 3.

Les entiers sont codés selon leur représentation binaire. La taille d'un entier x est ainsi le nombre de bits de son codage binaire, c'est à dire $\lceil \log_2(x+1) \rceil$, que nous noterons plus simplement $\log(x)$.

Exemple 10 L'entier $129 = 2^7 + 1$ a pour codage binaire 10000001 qui est de longueur $8 = \lceil \log_2(2^7 + 2) \rceil$. L'entier 2^1 a pour codage binaire 10 de longueur 2.

4.2.1 Mise en garde

L'algorithme :

```
ENTRÉE : un entier x ;
SORTIE :
```

```
i := 0 ;
tantque (i < x) faire
    i := i + 1 ;
```

a une complexité monstrueuse ! Cette complexité est exponentielle. Le nombre d'instructions est égale à x c'est à dire $2^{\log(n)}$. Prenez par exemple l'entier 2^{200}. Sa taille 200 est raisonnable (une taille conseillée des entiers manipulés par RSA est 1024 bits, c'est à dire 5 fois plus longue). Je peux écrire sa représentation en base binaire sur quelques lignes :

```
1 0000000000 0000000000 0000000000 0000000000 0000000000
  0000000000 0000000000 0000000000 0000000000 0000000000
  0000000000 0000000000 0000000000 0000000000 0000000000
  0000000000 0000000000 0000000000 0000000000 0000000000
```

Une machine effectuant le calcul sur cette entrée nécessiterait en énergie celle produite par le soleil en un millier d'année ! Et ce en supposant que la modification d'un bit ne nécessite que 10^{-16} Joule (quantité minimale théorique admise).

4.2.2 Addition

Le problème :

```
Addition
ENTRÉE : deux entiers x et y ;
SORTIE : la somme x + y ;
```

admet une solution algorithmique linéaire en temps bien connue depuis l'école primaire !

4.2.3 Multiplication

Le problème :

```
Multiplication
ENTRÉE : deux entiers x et y ;
SORTIE : le produit x · y ;
```

admet une solution algorithmique quadratique en temps bien connue depuis l'école primaire. D'autres solutions sont plus efficaces.

Diviser pour régner

Il est très facile d'écrire un algorithme multipliant deux nombres écrits en binaire avec une complexité en temps en $O(n^{\log_2 3})$. Pour cela, il suffit d'écrire un algorithme récursifs qui :

- prend en entrée deux entiers x et y de même taille paire.
- calcule la taille $2 \cdot n$ commune à x et y.

- décompose x en deux entiers x_1 et x_0 tels que $x = x_1 \cdot 2^n + x_0$.
- décompose y en deux entiers y_1 et y_0 tels que $y = y_1 \cdot 2^n + y_0$.
- calcule les produits $p_1 := x_1 \cdot y_1$, $p_2 := (x_1 - x_0) \cdot (y_0 - y_1)$ et $p_3 := x_0 \cdot y_0$.
- calcule et retourne la somme $2^{2n} \cdot p_1 + 2^n \cdot p_1 + 2^n \cdot p_2 + 2^n \cdot p_3 + p_3$. Ici toutes les opérations sont en temps linéaires, que ce soit les 5 additions ou les 4 multiplications par des puissances de 2 qui sont en fait des décalages de bits.

Cet algorithme dont la correction est basée sur l'égalité $x \cdot y = 2^{2n} p_1 + 2^n \cdot p_2 + p_3$ (on laissera le lecteur la vérifier) a une fonction de complexité en temps $f(2 \cdot n)$ vérifiant :

$$f(2 \cdot n) \leq 3 \cdot f(n) + c \cdot n$$

Sa complexité appartient donc à $O(n^{\log_2(3)})$.

Notons qu'une approche similaire, qui découpe chaque entier en plus de 2 blocs, permet pour tout réel $r > 0$, d'obtenir une solution algorithmique en $O(n^{1+r})$.

Transformée de Fourier

D'autres solutions plus efficaces existent. La plus efficace, connu à ce jour, utilise les transformées de Fourier. Sa complexité en temps est $O(n \cdot \log(n) \cdot \log(\log(n)))$. Cette solution est cependant d'une implémentation coûteuse. Pour plus de détails voir l'excellent ouvrage de Knuth section 4.3.3 (The Art of Computer Programming, volume 2. Addison Wesley, 1973)

Conclusion

Nous considérerons que la multiplication admet une solution en temps linéaire.

4.2.4 Division euclidienne

Le problème a pour définition :

Division euclidienne
ENTRÉE : deux entiers positifs a et b ;
SORTIE : les entiers q et r définis par $a = b \cdot q + r$ et $0 \leq r < b$;

Une solution exponentielle en temps

Calculer la division euclidienne de a par b peut être réalisé par l'appel de fonction $\mathtt{F}(b, a, 0)$ ainsi définie :

F
ENTRÉE : trois entiers b, r et q ;
SORTIE : deux entiers r' et q' ;
    si $r \geq b$ alors

```
        retourner F(b, r − b, q + 1) ;
    sinon
        retourner (q, r) ;
```

Cela n'a échappé à personne. L'exécution de cet algorithme sur l'entrée $(0, 2^n, 1)$ nécessite 2^n appels récursifs. La complexité est donc au moins exponentielle.

Une solution en temps quadratique

Calculer la division euclidienne de a par b peut être réalisé par l'appel de fonction $\mathtt{F}(b, a, 0, n)$ où n est un entier vérifiant $0 < a < 2^n \cdot b$ et F est la fonction ainsi définie :

```
F
ENTRÉE : quatre entiers b, r, q, k ;
SORTIE : deux entiers r' et q' ;
    si k = 0
        retourner (r, q) ;
    sinon si r < 2^(k−1) · b
        retourner F(b, r, 2 · q, k − 1);
    sinon
        retourner F(b, r − 2^(k−1) · b, 2 · q + 1, k − 1);
```

Correction

Cette fonction a pour invariants b, $0 \leq r < 2^k b$ et $2^k bq + r$. Le troisième invariant vaut initialement a et à la sortie de programme (lorsque $k = 0$), il vaut $bq + r$. Ainsi, les valeurs retournées q et r vérifient-elles $0 \leq r < b$ et $a = bq + r$.

Complexité en temps

Le troisième argument q vaut initialement 0, est multiplié au moins par 2 à chaque appel de la fonction et vérifie finalement $bq + r = a$ avec $r < b$. On en déduit qu'il y a plus $\log(b) - \log(a)$ appels de fonctions.

A chaque appel de fonction, chacune des opérations est de complexité constante ($2 \cdot q + 1$ nécessite par exemple un décalage et une modification de bit) excepté le calcul $r - 2^{k-1} \cdot b$. Ce calcul est linéaire en la taille des opérandes, nécessite $\log(a)$ opérations élémentaires au premier appel et $\log(b)$ au dernier. Il est facile d'observer que ses opérandes voient leur taille diminuer de 1 à chaque nouveau calcul. La complexité en temps de F est donc $O(log^2(a) - \log^2(b))$.

4.2.5 Exponentiation modulaire

Le problème posé est le suivant :

```
Exponentiation modulaire
ENTRÉE : trois entiers a, e et m ;
SORTIE : a^e mod m
```

Résoudre efficacement un tel problème ne se fait pas en calculant l'exponentielle a^e puis en extrayant son modulo. Car calculer l'exponentielle d'un nombre nécessite un espace et donc un temps exponentiel : la taille de 2^e est égal $||2^e|| = \log(2^e) = n = 2^{\log(e)} = 2^{||e||}$ et est donc une fonction exponentielle de la taille de l'entrée.

Une solution efficace consiste à éviter de manipuler de trop grands nombres en calculant le reste modulaire systématiquement ou de façon régulière. Une première solution qui se caractérise par sa simplicité effectue cette réduction systématiquement.

Une solution en temps cubique

Une solution élégante est la suivante :

```
F
ENTRÉE : trois entiers a, e et m
SORTIE : l'entier a^e mod m
    si e = 1
        retourner ((a · y) mod m) ;
    sinon si e est pair
        retourner F(a^2, e/2, m);
    sinon
        retourner (a · F(a^2, (e−1)/2, m)) mod m ;
```

Correction

Simple conséquence des égalités $a^{2\cdot e} = (a^2)^e$ et $a^{2\cdot e+1} = a \cdot (a^2)^e$.

Complexité en temps

A chaque appel récursif de `F`, la taille de e est diminuée de 1. Le nombre d'appels est donc $\log(e)$.

À chaque appel de fonction, l'opération la plus complexe en temps est le calcul du reste modulaire. On a vu précédemment que le calcul de $s \bmod t$ nécessitait $\log^2(s) - \log^2(t)$ opérations. Lors du premier appel de `F`, ce calcul nécessite $\log^2(a) - \log^2(m)$ opérations. Ensuite, tous les arguments passés en paramètres sont de taille au plus celle du modulo m, à une constante multiplicative 2 près. La complexité de ce calcul est donc au plus $O(\log^2(m))$. En conclusion, la complexité en temps est $O(\log^2(a) + \log(e) \cdot \log^2(m))$.

4.2.6 pgcd

Le problème du calcul du pgcd, c'est à dire :

pgcd
ENTRÉE : deux entiers a et b
SORTIE : $pgcd(a, b)$

admet plusieurs solutions. L'une d'entre elles fut trouvée par un savant grec et a pour nom l'algorithme d'Euclide.

Algorithme d'Euclide

L'algorithme utilise le calcul du reste modulaire et s'écrit ainsi :

```
F
ENTRÉE : deux entiers a et b
SORTIE : pgcd(a,b)
    si a = 0
        retourner b ;
    sinon a > b
        retourner F(a-b,b) ;
    sinon
        retourner F(b,a) ;
```

Cet algorithme a pour invariant l'expression $pgcd(a, b)$. Sa complexité est malheureusement exponentielle. Le calcul sur l'entrée $(a, 1)$ nécessite $a = 2^{||a||}$ appels récursifs !

Plusieurs algorithmes du calcul du pgcd existent. Nous en présentons un qui n'utilise pas de de division euclidienne et qui a une complexité en temps optimale parmi les solutions connues.

Algorithme d'Euclide en version binaire

```
F
ENTRÉE : deux entiers a et b
SORTIE : pgcd(a,b)
    si a = b
        retourner a ;
    sinon si a et b sont pairs
        retourner 2 · F(a/2, b/2) ;
    sinon si a est pair
        retourner  F(a/2, b) ;
    sinon si b est pair
        retourner  F(a, b/2) ;
    sinon si a > b
        retourner  F((a-b)/2, b) ;
```

```
    sinon
        retourner  F(a, (b−a)/2) ;
```

Cet algorithme nécessite $n := max\{\log(a), \log(b)\}$ appels récursifs. Chaque opération est linéaire en la taille des opérandes. La complexité en temps de F est donc $O(n^2)$.

Algorithme d'Euclide étendu

Il est facile d'étendre les algorithmes précédents afin de calculer non seulement le pgcd de deux nombres a et b mais aussi deux entiers relatifs α et β vérifiant : $\alpha \cdot a + \beta \cdot b = pgcd(a, b)$ (leur existence est établie par le Fait 18). Afin de simplifier l'écriture de cet algorithme, on étendra non l'algorithme précédent mais l'algorithme suivant dont le comportement est très proche :

```
ENTRÉE : deux entiers a et b
SORTIE : pgcd(a, b)
    si a = b
        retourner a ;
    sinon si a et b sont pairs
        retourner 2 · F(a/2, b/2) ;
    sinon si a < b
        retourner  F(b, a) ;
    sinon si a est pair
        retourner  F(a/2, b) ;
    sinon       % on a : a et b impairs avec a > b
        retourner  F(a − b, b) ;
```

L'algorithme étendu calcule non seulement le *pgcd* d des deux entiers a et b, mais aussi les entiers $k_a := \frac{a}{pgcd(a,b)}$ et $k_b := \frac{b}{pgcd(a,b)}$ ainsi que deux entiers relatifs α et β satisfaisant : $\alpha \cdot a + \beta \cdot b = pgcd(a, b)$. Il a pour écriture celle de la figure 4.1.

Le nombre d'appels récursifs est, à une constante multiplicative près, au plus la taille maximale des entrées a et b. Tous les entiers manipulés lors de ces appels ont une taille au plus égale au maximum de celles des entrées a et b. Chaque opération de chaque appel (affectation, addition, multiplication par 2) a une complexité en temps au pire linéaire. Ainsi, la complexité en temps est en $O(n^2)$ avec $n := max\{log(a), log(b)\}$.

Autres algorithmes

La littérature est riche d'algorithme calculant le pgcd de deux nombres a et b. Certains d'entre eux procèdent récursivement ainsi :

1. calcul de q, le quotient par la division euclidienne de a par b.
2. calcul du pgcd de $a - q \cdot b$ et de b.

```
Fetnd
ENTRÉE : deux entiers a et b
SORTIE : 5 entiers d, k_a, k_b, α, β vérifiant :
         d = pgcd(a,b), k_a = a/d, k_b = b/d, α·a + β·b = d
    si a = b
        retourner (a, 1, 1, 1, 0) ;
    sinon si a et b sont pairs
        (d, k_a, k_b, α, β) := Fetnd(a/2, b/2) ;
        retourner (2·d, k_a, k_b, α, β) ;
    sinon si a < b
        (d, k_b, k_a, β, α) := Fetnd(b, a) ;
        retourner (d, k_a, k_b, α, β) ;
    sinon si a est pair    % on a b impair et a > b
        (d, k_a, k_b, α, β) := Fetnd(a/2, b) ;
        si α est pair
            retourner (d, 2·k_a, k_b, α/2, β)
        sinon                % α est impair ainsi que k_b
            retourner (d, 2·k_a, k_b, (α+k_b)/2, β − k_a);
    sinon si b est pair    % on a b pair, a pair et a > b
        (d, k_a, k_b, α, β) := Fetnd(a, b/2) ;
        si β est pair
            retourner (d, k_a, 2·k_b, α, β/2)
        sinon                % α est impair ainsi que k_b
            retourner (d, k_a, 2·k_b, α − k_b, (β+k_a)/2);
    sinon                    % on a : a et b impairs avec a > b
        (d, k_a, k_b, α, β) := Fetnd(a − b, b) ;
        retourner (d, k_a + k_b, k_b, α, β − α) ;
```

FIG. 4.1 – Algorithme d'Euclide étendu

L'ensemble de ces algorithmes conserve la même complexité en temps à savoir $O(n^2)$ avec $n := max\{log(a), log(b)\}$.

4.2.7 Inversion modulaire

Le calcul de l'inverse se fait simplement en utilisant le calcul du pgcd étendu traité dans la section précédente. Soit `pgcd` une fonction qui calcule sur l'entrée de deux entiers a et b leur pgcd ainsi que deux entiers relatifs α et β vérifiant $\alpha \cdot a + \beta \cdot b = pgcd(a, b)$. L'algorithme est le suivant :

```
F
ENTRÉE : deux entiers a et n premiers entre eux
SORTIE : a^{-1} mod n
```

```
(d, α, β) := pgcd(a, n) ; % on a d = 1
(q, r) := divisionEuclidienne(α, n) ;
retourner r ;
```

La complexité en temps est à l'image de celle des routines utilisées (`pgcd`, `divisionEuclidienne`) et est donc en $O(n^2)$ avec $n := \max\{\log(a), \log(n)\}$.

4.2.8 Quelques problèmes faciles dans $\mathbb{Z}p$

Les problèmes `résiduositéQuadratique` ou `racineCarréeModulaire` (voir chapitre précédent) sont supposés difficiles si l'on choisit un ensemble $\mathbb{Z}_n$ avec n quelconque (c'est à dire dans l'immense majorité des cas avec n composite). Par contre si l'on considère un entier p premier, ces deux problèmes admettent des solutions rapides. Les faits suivants traduisent ce propos et seront admis.

Fait 25 Le problème :

`ENTRÉE : un entier` p `premier, un entier` $a \in \mathbb{Z}_p$
`SORTIE :` *oui* `si` a `est un résidu quadratique modulo` p

admet une solution algorithmique de complexité en temps $O(\log(p)^2)$.

preuve :
Une solution est détaillée p 72 dans l'ouvrage de Menezes.

Fait 26 Le problème :

`ENTRÉE : un entier` p `premier, un résidu quadratique` a `modulo` p
`SORTIE : toutes les racines carrées modulaires de` a

admet une solution algorithmique de complexité en temps $O(\log(p)^2)$.

preuve :
Une solution est détaillée p 111 dans l'ouvrage de Demazure(cours d'algèbre ed. Cassini).

Chapitre 5

Protocole de Mise en Gage

Imaginons la cambiste Alice souhaitant prouver à Bob qu'elle peut prédire les évolutions de la bourse. Supposons en outre qu'elle ne souhaite pas l'en informer de peur qu'il ne s'enrichisse aussi. Une solution qui s'offre à eux est de se munir d'un coffre-fort avec deux clefs et tel que son ouverture nécessite ces deux clefs. Ainsi,

1. Alice peut déposer l'information dans ce coffre, fermer en utilisant les deux clefs et en remettre une à Bob.
2. le jour venu, Bob peut ouvrir avec l'aide de Alice ce coffre et lire son contenu.

Un tel protocole est appelé protocole de *mise en gage*. Il contient deux sous protocoles :

1. le protocole de mise en gage où Alice s'engage sur le contenu d'un message.
2. un protocole de révélation où Bob vérifie le contenu du message mis en gage.

Ce protocole doit vérifier deux propriétés dites d'*engagement* et de *dissimulation*. Il y a engagement si Alice ne peut modifier le contenu du message sans que Bob ne s'en aperçoive. Il y a dissimulation si Bob ne peut deviner le message de Alice. Dans le cas de l'utilisation d'un coffre-fort à deux clefs, ces deux propriétés sont vérifiées car, d'une part, Alice ne peut ouvrir le coffre en l'absence de Bob et ainsi modifier le message, et d'autre part, Bob ne peut ouvrir le coffre en l'absence de Alice et ainsi lire à son insu le message contenu.

5.1 Définition

Un protocole `P` *de mise en gage* d'un bit $b \in \{0,1\}$ est un protocole défini à partir d'un couple d'ensembles (Y_0, Y_1) et qui contient au moins les deux sous instructions suivantes :

Mise en Gage `Alice` envoie un message $y \in Y_0 \cup Y_1$ à Bob.

Révélation `Alice` envoie à Bob une preuve (y, b, x) de l'appartenance $y \in Y_b$.

Un tel protocole vérifie la propriété :

- d'*engagement* si il est impossible à `Alice` de trouver deux preuves de la forme $(y, 0, x_0)$ et $(y, 1, x_1)$ des appartenances respectives $y \in Y_0$ et $y \in Y_1$.
- de *dissimulation* si il est impossible à `Bob` de prouver la non-appartenance $y \notin Y_{\neg b}$.

Ces deux propriétés sont malheureusement incompatibles. La preuve d'une telle incompatibilité est assez simple : si il est impossible à Alice d'obtenir deux preuves de la forme $(y, 0, x_0)$ et $(y, 1, x_1)$, il suffit à Bob d'énumérer tous les triplets de la forme (y', b', x') et de retourner b' dès que $y = y'$, et ainsi de calculer b. En conséquence, on s'intéresse à fournir non pas une sécurité inconditionnelle mais une sécurité :

calculatoire le terme "impossible" est remplacé par difficile à calculer.

calculatoire conditionnelle le terme "impossible" est remplacé par aussi difficile qu'un problème supposé difficile, comme par exemple le problème `résiduositéQuadratique`.

Remarque 27 Ces définitions sont en fait trop simples. En réalité, dans le cas de la propriété de dissimulation, par exemple, on impose de ne pas pouvoir facilement deviner le bit b mis en gage autrement qu'avec une probabilité $\frac{1}{2}$. En vue de simplification, on conservera les définitions précédentes.

5.1.1 Un exemple de construction

Il est facile de construire à l'aide d'une fonction à sens unique f un tel protocole. Soit f une telle fonction ayant pour ensemble de départ un ensemble de la forme $\{0, 1\} \times X$ et pour ensemble image un ensemble Y. Dans un tel cas, le protocole de mise en gage d'un bit b devient :

`Alice choisit` $x \in X$
`envoie` $y := f((b, x))$ `à Bob ;`

Le protocole de révélation est :

`Alice envoie` (b, x) `à Bob ;`
`Bob vérifie` $y = f((b, x))$ `;`

Pour qu'un tel protocole vérifie :

- la propriété d'engagement selon une sécurité inconditionnelle, il suffit que $f(\{0\} \times X)$ et $f(\{1\} \times X)$ soient disjoints.
- la propriété d'engagement selon une sécurité calculatoire, il suffit qu'il soit difficile de calculer deux couples de la forme $(0, x)$ et $(1, x')$ ayant même image.
- la propriété de dissimulation selon une sécurité calculatoire, il suffit qu'il soit impossible de décider pour tout y et tout b, si il existe ou non un x vérifiant $f(b, x) = y$.

5.1.2 Mise en gage d'un mot

Si l'on souhaite mettre en gage non un bit mais un mot, une solution immédiate est de mettre en gage chacun des bits !

5.2 Exemple d'utilisation : le jeu à pile ou face

Un grand nombre de protocoles simulant des jeux utilisent la mise en gage. À titre d'exemple, citons le plus simple parmi eux : le jeu à pile ou face.

Pile ou face
Alice choisit un bit $b \in \{0,1\}$;
      met en gage auprès de Bob b ;
Bob   choisit un bit $c \in \{0,1\}$;
      envoie c à Alice ;
Alice révèle b à Bob ;
      gagne si $b = c$ et perd sinon.

5.3 Une mise en gage issue de résiduositéQuadratique

L'exemple présenté ici est obtenu à partir du problème résiduositéQuadratique. Ce protocole possède :
- une propriété d'engagement à sécurité inconditionnelle.
- une propriété de dissimulation à sécurité calculatoire conditionnelle.

L'ensemble $QR(n)$ désigne l'ensemble des résidus quadratiques dans $\mathbb{Z}_n$ et $\tilde{QR}(n)$ son ensemble complémentaire.

P1
(Préalable)
Alice choisit aléatoirement deux nombres premiers p et q ;
       calcule $n := p \cdot q$;
       choisit aléatoirement $m \in \tilde{QR}(n)$;
       publie (n, m) ;

(Mise en gage du bit b par Alice auprès de Bob)
Alice choisit aléatoirement $x \in \mathbb{Z}^*_{p \cdot q}$;
       calcule $y := m^b x^2 \, mod \, n$;
       envoie y à Bob ;

(Révélation du bit par Alice auprès de Bob)
Alice envoie b, p, q et x à Bob ;
Bob   vérifie $\bigwedge\{m \in \tilde{QR}(n), \{p,q\} \subseteq Prime, n = p \cdot q,$
                $y = m^b \cdot x^2 \, mod \, n, x \in \mathbb{Z}^*_n\}$;

5.3.1 Propriété d'engagement

Fait 28 `P1` vérifie la propriété d'engagement.

preuve :
Rompre la propriété d'engagement revient à fournir deux triplets de la forme $(y, 0, x), (y, 1, x') \in \mathbb{Z}_n \times \{0,1\} \times \mathbb{Z}_n^*$ tels que :

$$m^0 \cdot x^2 \, mod \, n = y = m^1 \cdot x'^2 \, mod \, n$$

Ce qui nécessite que m soit égal à $(x \cdot x'^{-1})^2 \, mod \, n$ et donc soit un résidu quadratique. Ce qu'il n'est pas. Ainsi, l'engagement est inconditionnel.

5.3.2 Propriété de dissimulation

Le calcul pour Bob du bit b se ramène à l'étude du problème suivant :

`AttaqueBobP1`
`ENTRÉE` : $n = p \cdot q$ `avec` p `et` q `premiers,`
$y = m^b \cdot x^2 \, mod \, n$ `avec` $b \in \{0,1\}$, $m \in \tilde{QR}(n)$, $x \in \mathbb{Z}_n^*$
`SORTIE` : b

Fait 29 `AttaqueBobP1` est aussi facile et aussi difficile que `résiduositéQuadratique`.

preuve :
Clairement $QR(n) \cup \tilde{QR}(n)$ et $QR(n)$ sont stables par le produit modulaire. De plus, pour tous entiers $x, y \in \mathbb{Z}_n$, on a $x \cdot y^2 \, mod \, n \in QR(n)$ si et seulement si $x \in QR(n)$. Ainsi, `AttaqueBobP1` est équivalent au problème suivant :

`AttaqueBobP1`
`ENTRÉE` : $n = p \cdot q$ `avec` p `et` q `premiers,` $y \in \mathbb{Z}_n$
`SORTIE` : *oui* `si` $y \notin QR(n)$

lui même équivalent à `résiduositéQuadratique`.

Remarque 30 Le protocole présenté ici est une légère simplification du protocole habituellement utilisé. La seule modification porte sur l'ensemble $\tilde{QR}(n)$ qui est non pas l'ensemble complémentaire de $QR(n)$ dans $\mathbb{Z}_n$ mais un sous-ensemble formés des *pseudo-carrés*. En effet $\mathbb{Z}_n$ se partitionne en trois ensembles : l'ensemble des résidus carrés, l'ensemble des pseudo-carrés et un ensemble pour lequel on sait déterminer en temps polynomial l'appartenance à celui-ci. Le problème de résiduosité quadratique restreint à $QR(n) \cup \tilde{QR}(n)$ admet pour seule solution algorithmique connue en temps polynomiale une solution qui devine la résiduosité avec une probabilité de succès $\simeq \frac{1}{2}$. Ainsi, Bob ne peut deviner le bit mis en gage qu'avec une probabilité $\simeq \frac{1}{2}$. Le protocole vérifie ainsi la définition de dissimulation renforcée évoquée dans la remarque 27.

Chapitre 6

Signatures

Définition 7 Un *procédé de signature* est un quintuplet (X, Y, K, S, V) où :

X est un ensemble de messages.

Y est un ensemble de signatures.

K est un ensemble de clefs de la forme $k = (k_{\text{privée}}, k_{\text{publique}})$.

S est un ensemble de fonctions $(s_k)_{k \in K}$ définies $X \to Y$.

V est un ensemble de fonctions $(v_k)_{k \in K}$ définies $X \times Y \to \{0, 1\}$.

tel que pour toute clef $k \in K$, pour tout message $x \in X$ et toute signature $y \in Y$ on ait :

$$v_k(x, y) = 1 \text{ si et seulement si } y = s_k(x)$$

Tout procédé de signature doit contenir une fonction de signature s_k et une fonction de vérification v_k facilement calculable. Un tel procédé doit empêcher de contrefaire une signature, c'est à dire d'empêcher Oscar de résoudre facilement le problème :

`Contrefaçon Universelle`
`ENTRÉE : clef publique` k_{publique}`, un message` $x \in X$
`SORTIE : un couple` (x, y) `tel que` $v_{k_{\text{publique}}}(x, y) = 1$

Cette condition est clairement insuffisante comme l'illustre l'exemple suivant :

Exemple 11 Considérons une machine, par exemple un satellite, recevant des programmes signés et les exécutant. L'utilisateur d'une telle machine souhaite que l'on ne lui porte pas préjudice en faisant exécuter des programmes mêmes aléatoires qui pourraient provoquer l'arrêt de ces machines.

En conséquence, on souhaite rendre difficile le problème suivant, dont la résolution permet non pas de signer un document donné mais de fournir un document signé :

`Contrefaçon (Existentielle)`
`ENTRÉE : clef publique` k_{publique}
`SORTIE : un couple` (x, y) `tel que` $v_{k_{\text{publique}}}(x, y) = 1$

Cette condition peut et doit être renforcée en tenant compte des informations que possède Oscar grâce à l'écoute des messages échangés par Alice et Bob. Une approche plus exigeante consiste ainsi à supposer difficile le problème suivant :

Contrefaçon'
ENTRÉE : clef publique k_{pub} et
j couples $(x_i, y_i)_{i\in[1,j]}$ tels que $v_k(x_i, y_i) = 1 \ \forall i \in [1, j]$
SORTIE : un couple (x, y) autre que ceux entrés avec $v_{k_{\text{pub}}}(x, y) = 1$

6.1 Signature jetable : le procédé de Lamport

Le procédé de signature suivant vérifie une très belle propriété : sa sécurité repose sur l'hypothèse de l'existence de fonctions à sens uniques (contrairement à beaucoup d'autres qui nécessitent des fonctions à sens unique à brèche secrète). Soit $h : Z \to Y$ une fonction à sens unique. Le procédé de Lamport est défini par le 5-uplet $(\{0, 1\}, Y, K, S, V)$ où :

- chaque clef $k \in K$, de la forme $(k_{\text{privée}}, k_{\text{publique}})$, est de la forme $((t_0, t_1, u_0, u_1), (u_0, u_1))$ avec $u_0 = h(t_0)$ et $u_1 = h(t_1)$.
- chaque fonction de signature s_k est définie par $s(0) = u_0$ et $s(1) = u_1$.
- chaque fonction de vérification v_k est définie par $v_k(x, y) := (h(t_x) = y)$.

L'utilisation par Alice d'un tel procédé de signature se fait de la façon suivante.

(Diffusion préalable de la clef publique)
Alice choisit une fonction à sens unique h ;
tire aléatoirement deux éléments t_0 et t_1 ;
calcule $u_0 := h(t_0)$ et $u_1 := h(t_1)$;
publie (h, u_0, u_1) ;

(Envoi du bit x signé)
Alice envoie (x, t_x) ;

(Vérification de la signature)
Bob vérifie $h(t_x) = u_x$;

Le défaut majeur d'un tel procédé est que la vérification de la signature exige la connaissance de la clef privée k et donc fournit l'algorithme de signature s_k. Il est donc nécessaire de "jeter" la clef après chaque signature.

6.2 Signature avec récupération des messages

Il existe deux classes de procédés de signature. La première présentée ici manipule des signatures contenant le message signé :

Définition 8 Un procédé de signature est dit *avec récupération de message* si il est facile de calculer x en fonction de $sig_k(x)$.

Avant de montrer comment construire de tels procédés de signatures, présentons les fonctions "redondantes". Une fonction R est *redondante* si :

- R est injective.
- le codomaine est de cardinalité bien plus grande que celle du domaine.
- les calculs de R et de R^{-1} sont faciles.

Exemple 12 Voici quelques exemples de fonctions redondantes opérant sur un ensemble de la forme $[0, 2^s[$:

- r fixé, la fonction qui à tout $x \in [0, 2^s[$ associe $R(x) := x \cdot 2^r$.
- la fonction R qui associe à tout $x \in [0, 2^s[$ le message $R(x) := x \,||\, x$ obtenu en concaténant x à lui-même (on a : $R(x) = 2^s \cdot x + x$).

Muni d'une telle fonction R et d'une famille de fonctions à sens unique à brèche secrète bijectives $(f_k)_{k \in K}$, on construit un procédé de signature avec récupération de message en prenant :

1. $s_k(x) := f_k^{-1}(R(x))$.
2. $v_k(x, y) :\Leftrightarrow (x = R^{-1}(f_k(y)))$.

Il est nécessaire de choisir une fonction R à codomaine bien plus grand que le domaine comme le suggère le fait suivant :

Fait 31 Si R est l'identité, `Contrefaçon` est facile.

preuve :
`Contrefaçon` admet pour solution algorithmique efficace :

`ENTRÉE : la clef publique contenu dans` k
`SORTIE : un couple` (x, y) `tels que` $v_k(x, y) = 1$
`tirer aléatoirement` $y \in Y$;
`retourner` $(f_k(y), y)$;

Exemple 13 Un exemple est la signature RSA utilisant la fonction à sens unique RSA et une fonction redondante R du type de celles présentées à l'Exemple 12 (pour plus de détails voir section 11.3.5 de l'ouvrage de Menezes).

`(Publication de la fonction de la clef publique)`
`Alice tire aléatoirement deux entiers premiers distincts` p `et` q ;
`calcule` $n := p \cdot q$ `et` $\varphi(n) := (p-1) \cdot (q-1)$;
`tire aléatoirement` $e \in [1, n-2]$ `premier avec` $\varphi(n)$;
`calcule` e^{-1} `l'inverse de` e `modulo` $\varphi(n)$;
`publie` n, e ;

`(Signature du message` $x \in R$`)`
`Alice calcule` $y := R(x)^{e^{-1}} \, mod \, n$;
`envoie à Bob` (x, y) ;

(Vérification de la signature)
Bob vérifie $R(x) = (y^e) \; mod \; n$;

La sécurité du protocole de signature repose non seulement sur les paramètres de la fonction à sens unique RSA (c'est à dire n et e) mais aussi sur la fonction R. Celle-ci doit notamment ne pas être un morphisme multiplicatif comme le prouve le fait suivant :

Fait 32 Si pour tous $x, y \in X$, on a $R(x \cdot y) = R(x) \cdot_n R(y)$, alors contrefaçon' est facile.

preuve :
En effet, il suffit à Oscar de récupérer deux messages (x_1, y_1) et (x_2, y_2) signés par Alice en utilisant sa même clef privée k et de fournir comme nouveau message signé (x, y) avec $x := x_1 \cdot x_2$ et $y := y_1 \cdot_n y_2$. En effet, on a : $v_k(x, y) = (R(x_1 \cdot x_2) = (y_1 \cdot y_2)^e \; mod \; n) = (R(x_1) \cdot_n R(x_2) = y_1^e \; mod \; n \cdot_n {y_2}^e \; mod \; n)$. Ainsi, $v_k(x, y)$ est vrai si $v_k(x_1, y_1)$ et $v_k(x_2, y_2)$ le sont.

6.3 Signature avec appendice

Un procédé de signature est *avec appendice* si il est impossible de calculer le message x à partir de la signature $s_k(x)$. Une définition générale d'un tel procédé utilise :

- une fonction de *hachage* h qui transforme tout message X en un mot de longueur fixe inférieur à celle du message x. Cette fonction est souvent supposée à sens unique (voir exemple suivant).
- pour toute clef k une fonction $v_k : X \times Y \rightarrow \{0, 1\}$ et une famille de fonctions $s_k = (s_{k,i})_{i \in [1,m]}$ telles que pour toute clef $k \in K$, tout message $x \in X$, toute signature $y \in Y$ et tout indice $i \in [1, m]$ on ait :

$$v_k(x, y) = 1 \Leftrightarrow \exists i \in [1, m] s_{k,i}(x) = y$$

6.4 Un exemple : la signature El Gamal

Nous présentons ici un procédé de signature avec appendice important en cryptologie. Ce procédé basé sur la difficulté du calcul du logarithme discret a inspiré un grand nombre de standart de signature (notamment DSS). Sa sécurité sera étudiée en TD.

Soient p un entier premier et α un générateur de $\mathbb{Z}_p^*$ pour lequel le problème DiffieHellman est supposé difficile. Soit h une fonction de hachage à images dans $\mathbb{Z}_{p-1}$. Le protocole de signature est le suivant qui permet à Alice de signer un document vérifié par Bob.

(Génération clef publique)
Alice tire aléatoirement un entier $a \in [1, p-2]$;
calcule $\beta := \alpha^a \, mod \, p$;
retourne comme clef privée (p, α, a) et clef publique (p, α, β) ;

(Signature d'un message m)
Alice tire aléatoirement $k \in [1, p-2]$ premier avec $p-1$;
calcule $r := \alpha^k \, mod \, p$;
calcule $s := (k^{-1} \, mod \, (p-1)) \cdot (h(m) - a \cdot r) \, mod \, (p-1)$;
retourne (r, s) ;

(Vérification de la signature)
Bob vérifie $r \in [1, p-1]$;
vérifie $\alpha^{h(m)} \, mod \, p = \beta^r r^s \, mod \, p$;

Ce procédé de signature sera étudié en TD.

6.5 Signature à l'aveugle

Nous étudions ici un protocole de signature à l'aveugle qui permet à Alice de faire signer par Bob un document sans qu'il en prenne connaissance. Une solution non électronique à un tel problème serait de disposer d'enveloppes carbone qui permettent en signant l'enveloppe de signer le document qui y est caché :

Alice place le message dans une enveloppe carbone ;
donne le message enveloppé à Bob ;
Bob signe l'enveloppe ;
retourne l'enveloppe à Alice ;
Alice ouvre l'enveloppe;

Avant de présenter des solutions électroniques à un tel problème, présentons une utilisation de ces signatures à l'aveugle : le paiement anonyme.

6.5.1 Paiement anonyme

Une utilisation du protocole "Signature à l'aveugle" est de pouvoir régler des achats de façon parfaitement anonyme. C'est à dire qui empêche l'institut émetteur de titres de créance de connaître l'achat fait au moyen de ces titres. Une solution courante est l'utilisation de billets. L'anonymat peut toutefois être cassé : il suffit, lors de la délivrance de ces billets, de marquer les billets (par un procédé "invisible" ou plus simplement de se rappeler des numéros des billets). Un titre de créance contient pour informations :

1. une valeur.
2. une signature de l'institut émetteur.

3. un numéro identifiant, c'est à dire une information qui singularise ce document.

Ce protocole comporte au moins trois participants, que nous nommerons `acheteur`, `banque` et `vendeur`. Le protocole doit préserver les intérêts des trois participants et doit ainsi :

1. permettre à `banque` et `acheteur` d'échanger le titre de paiement et une somme d'argent correspondante.
2. permettre à `vendeur` et `banque` d'échanger le titre de paiement et une somme d'argent correspondante.
3. éviter à `banque` d'associer à `acheteur` le numéro de cette créance, qui lui permettrait lors du retour du titre de créance par `vendeur` d'effectuer le rapprochement entre `acheteur` et `vendeur`.

Présentons un protocole fournissant une solution à ce problème et qui utilise des moyens non électroniques. Le protocole possède une sécurité aléatoire dépendant d'un facteur de sécurité k.

```
(Émission préalable des titres non signés de valeur x)
banque   produit des titres non signés de valeur x à
                  numéros deux à deux distincts ;
         les diffuse gratuitement et anonymement à tous ;

(Échange titre de valeur x contre argent)
acheteur place k titres non signés dans k enveloppes carbone ;
         envoie à banque les k enveloppes ;
banque   fait k − 1 fois
             choisit aléatoirement une enveloppe non déjà ouverte ;
             ouvre l'enveloppe ;
             vérifie la valeur x ;
         signe le dernier titre camouflé ;
         retourne ce titre signé à acheteur ;
```

Observons que le protocole préalable de diffusion anonymes de titres à numéros distincts peut être réalisé par l'acheteur lui-même. Ce protocole est probabiliste et à une sécurité dépendant d'un facteur s supposé grand (par exemple, $s = 1000$) :

```
acheteur tire aléatoirement k numéros i_1, ..., i_k de taille s ;
         fabrique k titres de valeur x et de numéros i_1, ..., i_k ;
         place k titres non signés dans k enveloppes carbone ;
         envoie à banque les k enveloppes ;
banque   fait k − 1 fois
             choisit aléatoirement une enveloppe non déjà ouverte ;
             ouvre l'enveloppe ;
             vérifie la valeur x ;
             vérifie la taille du numéro ;
```

```
            vérifie que le numéro n'a pas déjà été produit ;
        signe le dernier titre camouflé ;
        retourne ce titre signé à acheteur ;
```

Pour implémenter ce protocole sur des documents électroniques, il nous faut utiliser le protocole de mise en gage, qui permet à `acheteur` de cacher une information (ici "mettre sous enveloppe") et le protocole associé de révélation qui permet à `banque` d'ouvrir, avec le consentement de `acheteur`, de prendre connaissance de cette information (ici "ouvrir l'enveloppe").

Il nous faut de plus signer à l'aveugle un document. Ceci est l'objet de cette section.

6.6 Une solution issue de RSA

Supposons qu'un protocole de signature (X, Y, K, S, V) soit tel que pour toute clef $k \in K$, il existe un ensemble C, composé des *facteurs de camouflage*, et deux fonctions $\varphi : X \times C \to X$ et $\psi : Y \times C \to Y$ telles que :

1. $\forall c \in C, \forall x \in X, \forall y' \in Y, v(\varphi(c,x), y') = 1 \Leftrightarrow v(x, \psi(c,y')) = 1$
2. φ et ψ sont facilement calculables si l'on connaît $k_{publique}$.
3. φ est difficile à inverser.

Il est alors aisé de construire un protocole de signature à l'aveugle :

```
Bob signe à l'aveugle un document x envoyé par Alice
ENTRÉE : un document x ∈ X
SORTIE : un document signé par Bob (x, y)
Alice tire aléatoirement c ∈ C ;
      envoie x' := φ(x, c) à Bob ;
Bob   calcule y' := s_k(x') ;
      envoie (x', y') à Alice ;
Alice calcule y := ψ(c, y') ;
      retourne (x, y) ;
```

Si l'on choisit un protocole de signature RSA (on a notamment $s_k(x) = y := R(x)^{e^{-1}} \bmod n$ et $v_k(x,y) :\Leftrightarrow y^e \bmod n = R(x)$), il suffit de choisir pour domaine de facteurs de camouflage $C = \mathbb{Z}_n^*$ et pour fonctions $\varphi(c,x)$ et $\psi(c,y)$ les simples opérations modulaires $c^e \cdot R(x) \bmod n$ et $c^{-1} \cdot y \bmod n$. En effet, on a : $v_k(c \cdot_n R(x), y') = 1$ si et seulement si :

$$y'^e \bmod n = c^e \cdot_n R(x) \Leftrightarrow (c^{-1} \cdot y')^e \bmod n = R(x) \Leftrightarrow v_k(R(x), c^{-1} \cdot y') = 1$$

Dit autrement, le protocole de signature à l'aveugle issu de RSA est le suivant :

```
ENTRÉE : un document x ∈ X
SORTIE : un document signé par Bob (x, y)
Bob   choisit une clef privée (n, p, e, e^{-1}) ; %voir section préc.
```

```
      publie sa clef publique (n, e) ;
Alice tire aléatoirement c ∈ Z*_n ;
      envoie x' := (c^e · R(x)) mod n à Bob ;
Bob   calcule y' := (x'^{e^{-1}}) mod n ;
      envoie (x', y') à Alice ;
Alice calcule y := c^{-1} · y' mod n ;
      retourne (x, y) ;
```

6.7 Signature simultanée de contrat sans arbitre

Supposons que deux participants `Alice` et `Bob` veuillent signer respectivement deux documents x^A et x^B et ce simultanément. On peut imaginer que ces deux documents sont deux créances de l'un vis à vis de l'autre ; aussi, Alice ne veut pas signer une créance avant que Bob n'ait signé la sienne. Et inversement. Une solution simple consiste à choisir un arbitre, c'est à dire un tiers de confiance (un notaire, par exemple) qui fait signer successivement chacun des documents par Alice et Bob puis qui envoie les documents signés. Une question naturelle se pose : peut-on se passer d'un tel arbitre ?

Une solution existe et a pour nom la signature simultanée sans arbitre. La solution que nous présentons ici utilise un sous protocole appelé le transfert inconscient.

6.7.1 Transfert inconscient

Un transfert inconscient permet à Alice d'envoyer à Bob deux messages x et y et permet à Bob d'en recevoir un des deux sans que Alice sache lequel. Plus précisément, le protocole que nous présentons ici consiste à :

1. envoyer par Alice les deux messages chiffrés $f(x)$ et $g(y)$.

et est tel que :

1. il est facile à Bob de calculer x ou y.
2. il est difficile à Bob de calculer x et y.
3. il est impossible à Alice de calculer le message facilement calculable par Bob.

Pour réaliser ce transfert, il suffit à Alice de transférer inconsciemment l'une des deux fonctions de déchiffrement f^{-1}, g^{-1}, puis d'envoyer à Bob $f(x)$ et $g(y)$. Le protocole présenté ci-dessous utilise :

- un protocole d'échange de messages à clefs publiques (style RSA) notée $(M, C, K, (e_k)_{k\in K}, (d_k)_{k\in K})$.
- un protocole d'échange de messages à clefs secrètes (style D.E.S) notée $(M', C', K', (e'_k)_{k\in K'}, (d'_k)_{k\in K'})$.
- on suppose en outre $M = K'$.

Le protocole a pour définition :

(Transfert inconscient d'une fonction de déchiffrement)
Alice choisit deux clefs $k_1, k_2 \in K$;
envoie les clefs publiques $k_{1,pub}, k_{2,pub}$ à Bob ;
Bob tire aléatoirement une clef $ks \in K'$;
tire aléatoirement $i \in \{1, 2\}$;
envoie à Alice $e_{k_i}(ks)$;
Alice calcule $ks_1 := d_{k_1}(e_{k_i}(k))$ et $ks_2 := d_{k_2}(e_{k_i}(k))$;

% À cet instant, Alice et Bob possèdent en commun la clef de
% session k_i, Bob ne peut que difficilement calculer ks_j
% avec $j \neq i$, Alice ne peut pas distinguer ks_i de ks_j

(Transfert inconscient de l'un des messages x ou y)
Alice envoie $e'_{ks_1}(x)$ et $e'_{ks_2}(y)$ à Bob ;
Bob retourne $x' := d'_{ks_1}(e'_{ks_1}(x))$ si $i = 1$; % on a $x' = x$
retourne $y' := d'_{ks_2}(e'_{ks_2}(y))$ si $i = 2$; % on a $y' = y$
%Le message transféré inconsciemment est x si $i = 1$ et y sinon

6.7.2 Un protocole de signature simultanée sans arbitre

Pour cette implémentation, on suppose que la signature du message x^A par Alice revient à posséder l'un quelconque de n couples de la forme $(g_i^A, d_i^A)_{i\in[n]}$. On peut supposer par exemple que chaque contrat numéroté de 1 à n possède deux feuillets gauche et droit et qu'ainsi :

1. Bob peut vérifier facilement pour tout entier $i \in [1, n]$ si un mot quelconque est le feuillet gauche (resp. droit) numéroté i signé.
2. Bob obtient un contrat dés qu'il obtient deux feuillets gauche et droit de même numéro signés.

Hypothèse similaire en ce qui concerne la signature du message x^B par Bob. À un détail que l'on peut considérer comme insignifiant (voir Remarque 33), le protocole est parfaitement symétrique. Il consiste pour Alice à :

1. envoyer des chiffrements des couples (g_i^A, d_i^A) à Bob,
2. à transférer inconsciemment pour chaque couple l'un des deux membres en clair.
3. puis à révéler "petit à petit" chacun des membres de chacun des couples.

Le protocole est défini par la Figure 6.1.

Remarque 33 Le protocole n'est pas parfaitement symétrique. Ainsi, Alice envoie de l'information avant Bob. Ce décalage permet à Bob de recevoir tous les bits de toutes les clefs au dernier tour de boucle avant Alice. Ainsi, Bob pourrait

(Transfert inconscient des moitiés de contrats)
Alice choisit $2 \cdot n$ clefs $k_{g,1}, k_{d,1}, \ldots, k_{d,n}$ ayant l bits ;
Bob idem
Alice crée les $2 \cdot n$ messages $(g_i^{A,A})_{i \in [1,n]}$;
chiffre chaque message avec une clef ;
envoie chaque message chiffré à Bob ;
Bob idem
Alice transfert inconsciemment chaque paire $\{k_{g,1}, k_{d,1}\}$ avec $i \in [1,n]$;
Bob idem
Bob pour tout entier $i \in [1,n]$
déchiffre chaque message avec la clef transférée correspondante ;
vérifie qu'il est un message de la forme g_i^A ou d_i^A ;
Alice idem
% Bob connaît donc n clefs. Ainsi pour tout $i \in [1,n]$,
% Bob peut déchiffrer soit le message g_i^A soit
% le message d_i^A. Idem pour Alice.

(Envoi ''petit à petit'' des contrats)
pour $i := 1$ à l faire
Alice envoie les i^{e} bits de chacune des $2 \cdot n$ clefs ;
Bob idem
Bob vérifie sur les clefs transférées la concordance des i^{e} bits reçus ;
Alice idem

FIG. 6.1 – signature simultanée

sortir du protocole à ce moment et posséder un document signé, contrairement à Alice. Cette différence peut être considérée comme insignifiante. Il ne manque à Alice qu'un bit pour obtenir le document signé. Elle peut donc le deviner avec une attaque exhaustive de complexité en temps $\theta(2)$!.

Cette remarque peut être étendue à tout moment du protocole. Cependant, si les deux adversaires ont des moyens de calcul inégaux. Si par exemple Alice calcule beaucoup plus rapidement que Bob, elle pourrait être tentée de choisir judicieusement un entier $l' < l$ afin de de sortir du protocole au l'^{e}passage de la boucle pour mener une attaque exhaustive réaliste sur le reste des clefs et laisser Bob devoir mener une attaque exhaustive beaucoup plus coûteuse et donc impossible.

6.8 D'autres propriétés et d'autres procédés de signature

D'autres procédés de signature existent. Nous en présentons brièvement quelques uns.

6.8.1 Signature "Fail then stop"

Les procédés de signatures présentés jusqu'ici garantissent à Alice la non contrefaçon de sa signature pourvu que personne ne sache résoudre tel ou tel problème supposé difficile (sécurité calculatoire conditionnelle). L'intérêt des procédés "Fail and stop" est de prouver la bonne foi de Alice dans le cas où sa signature aurait été contrefaite.

Un tel procédé est basé sur un problème P supposé difficile. Sur la difficulté d'un tel problème, repose non seulement la sécurité de la signature de Alice mais de toute une communauté. Contrefaire la signature de Alice nécessite de résoudre ce problème. Ainsi, dans le cas d'une contrefaçon, :

1. la bonne foi de Alice n'est pas mise en cause pourvu que la communauté soit convaincue du caractère ordinaire des moyens de calculs de Alice.
2. l'ensemble des membres de la communauté doit abandonner ce procédé de signature car le problème P est résolu. D'où le terme "Fail and stop".

Pour plus de détails voir Menezes p 478 et Stinson p 202.

6.8.2 Signature incontestable

Ce protocole de signature a deux objets :

1. d'une part, permettre la vérification de la signature qu'avec l'accord du signataire Alice. En d'autres termes, le signataire participe au protocole de vérification.
2. d'autre part protéger Bob contre une tentative frauduleuse de répudiation de la signature par Alice. Un tel protocole protège Bob même si Alice a une capacité de calcul infini.

Pour plus de détails, voir Stinson p 197.

Chapitre 7

Preuve à divulgation nulle

Un protocole important en cryptologie est celui d'identification. Il permet à Peggy de prouver son identité à Vic. Cette identification est une tâche courante de la vie quotidienne, que ce soit lors d'une connexion à un réseau en utilisant un mot de passe ou que ce soit lors d'un retrait d'argent liquide dans un guichet automatique en utilisant un code PIN. Ces deux procédés d'identification sont peu surs, il suffit à un espion d'écouter les messages échangés pour ensuite se faire passer pour Peggy. Cette attaque peut même être réalisée par Vic. Les protocoles présentés dans ce chapitre font échec à ce type d'attaque et empêchent ainsi Vic (ou Oscar) d'usurper l'identité de Peggy même après avoir enregistré la totalité des messages échangés lors d'un premier protocole d'identification.

Les procédés d'identification considérés dans ce chapitre reposent sur des protocoles de preuve et utilisent des problèmes difficiles à résoudre (ou supposés tels). Chaque personne étant identifié à l'un de ces problèmes, prouver son identité consiste alors à prouver sa capacité à résoudre ce problème. Puisque le problème est difficile, personne d'autre n'a cette capacité. Les entités Peggy et Vic se nomment lors du protocole `prouveur` et `vérificateur`.

Les problèmes que nous considérerons sont des problèmes décisionnels, c'est à dire, rappelons le, des problèmes qui associent respectivement à des instances positives ($\in L$) et négatives ($\notin L$) le booléen `oui` et le booléen `non`. Un tel protocole permet à `prouveur` d'établir sa capacité à décider de l'appartenance $x \in L$. Le protocole noté $< P, V >$ prend en entrée une instance x, répartit le calcul sur les deux machines `prouveur` et `vérificateur` et retourne un booléen b noté $< P, V > (x)$ où P désigne l'algorithme utilisé par `prouveur` et V celui utilisé par `vérificateur`. Le booléen $b :=< P, V > (x)$ est retourné en fait par `vérificateur` et a pour sémantique :

$$b = 1 \Leftrightarrow \texttt{vérificateur} \text{ est convaincu de l'appartenance } x \in L$$

Peggy peut établir auprès de Vic sa capacité à prouver $x \in L$ en fournissant une preuve de l'appartenance $x \in L$. Il y a alors divulgation d'information : rien n'empêche alors Vic ou Oscar de divulguer cette preuve auprès d'une tierce

personne et d'usurper l'identité de Peggy. Les protocoles que nous étudions ici sont à divulgation nulle et permettent à Peggy de prouver à Vic cette capacité à prouver $x \in L$ sans rien dévoiler de cette preuve.

Pour utiliser un tel protocole à des fins d'identification, il existe plusieurs possibilités. Dans chacune de celle-ci, le booléen b retourné par le protocole vérifie :

$$b = 1 \Leftrightarrow \texttt{vérificateur} \text{ identifie } \texttt{prouveur} \text{ comme étant Peggy}$$

La première possibilité consiste à associer l'identité du prouveur à sa capacité à prouver l'appartenance $x \in L$ pour toute instance x. En résumé, l'identité de quelqu'un est un langage L. Ceci requiert un sous protocole SP produisant des instances positives $x \in L$. Pour réaliser un tel protocole, on peut par exemple demander à un arbitre de confiance (André) de produire de telles instances. Ce protocole est décrit sur la Figure 7.1.

ENTRÉE :
SORTIE : oui si Vic reconnaît l'identité de Peggy, non sinon.
(diffusion préalable par Peggy de son identité L_{peg}
Peggy choisit et publie un langage L_{Peg} ;
%rappel : décider l'appartenance $x \in L_{Peg}$ doit être difficile

(production d'une instance positive $x \in L$)
André tire aléatoire $x \in L_{Peg}$;

(utilisation du protocole <P,V> associé à L_{Peg})
Peggy et Vic exécutent le protocole $< P, V >$ sur l'entrée x
et calculent $b :=< P, V > (x)$;

Vic retourne b ;

FIG. 7.1 – Un premier protocole générique

La seconde consiste à associer l'identité du prouveur à sa capacité à prouver l'appartenance $x_0 \in L$ où x_0 est fixé. L'identité de quelqu'un est ainsi un élément x_0. Ainsi, les membres d'une communauté {A,B,C,...} peuvent partager un même langage L et se distinguer par des instances positives deux à deux distinctes : $x_A, x_B, x_C, \ldots$. Ce protocole est décrit sur la Figure 7.2. Un protocole de preuve à divulgation nulle doit vérifier trois propriétés :

consistance cette première propriété préserve les intérêts des deux participants qui souhaitent qu'il y ait identification si les conditions sont réunies, c'est à dire lorsque $x \in L$. Ceci se traduit simplement en :

$$(cons) \forall x \in L < P, V > (x) = 1$$

ENTRÉE :
SORTIE : oui si Vic reconnaît l'identité de Peggy, non sinon.
(diffusion préalable par la communauté de L)
André choisit L ;
%rappel : décider l'appartenance $x \in L$ doit être difficile

(diffusion préalable par Peggy de son identité x_{Peg}
Peggy publie une instance positive x_{Peg}

(utilisation du protocole <P,V> associé à L)
Peggy et Vic exécutent le protocole $< P, V >$ sur l'entrée x_{Peg}
et calculent $b :=< P, V > (x_{Peg})$;

Vic retourne b ;

FIG. 7.2 – Un second protocole générique

significativité cette seconde propriété préserve les intérêts de vérificateur et lui garantit de retourner oui seulement si x est une instance positive et ce même en face d'un usurpateur d'identité. C'est à dire de quelqu'un éventuellement malhonnête prêt à dévier du protocole $< P, V >$ en utilisant un algorithme P' différent de P. Cette propriété pourrait s'écrire simplement en :

$$\forall x \forall P' \, (< P', V > (x) = 1 \Rightarrow x \in L)$$

Cette propriété, s'écrirait de façon équivalente (par utilisation de la contraposée) en :

$$\forall x \notin L \forall P' < P', V > (x) = 0$$

Cette définition est malheureusement trop contraignante. Elle n'est donc pas définitive. Une définition définitive sera fournie dans la prochaine section.

divulgation nulle cette troisième propriété préserve les intérêts de prouveur en lui assurant qu'un vérificateur même malhonnête ne puisse extraire de cette échange "aucune information". Nous présenterons plus loin une définition plus précise de cette notion. Pour fixer les idées, disons simplement que l'on souhaite qu'un vérificateur éventuellement malhonnête ne puisse extraire de cette échange aucune information qu'il ne pourrait calculer lui même (sans interagir avec prouveur).

Définition définitive de la significativité

Les deux définitions de significativité et de divulgation nulle sont malheureusement incompatibles. Il nous faut donc relâcher la notion de significativité et accepter que `vérificateur` puisse être trompé avec une probabilité non nulle. À cette fin, on considère non un protocole $< P, V >$ mais une famille protocoles $(< P, V >_k)_{k>0}$ où k est un entier appelé facteur de sécurité. La significativité préserve les intérêts de `vérificateur` en lui garantissant une probabilité d'être trompé aussi faible que souhaitée. Ainsi, un tel protocole est *significatif* si :

$$(sign)\forall k > 0 \forall x \notin L \forall P' prob(< P', V >_k (x) = 1) \leq \frac{1}{2^k}$$

Notons que cette définition est équivalente à la définition où $\frac{1}{2}$ est remplacée par tout réel $0 < c < 1$. D'autres variantes de la significativité existent selon que l'on considère une sécurité :

inconditionnelle Dans les équations précédentes, on obtient une propriété pour tous les algorithmes P'. Ainsi, les intérêts du `vérificateur` sont préservés face à un attaquant disposant éventuellement de moyens calcul infinis. La sécurité est alors inconditionnelle.

calculatoire On peut relâcher la propriété de significativité en ne considérant que des algorithmes P' efficaces (où algorithme efficace signifie par exemple : algorithme probabiliste de complexité en temps polynomial). Il s'agit alors de sécurité calculatoire.

calculatoire conditionnelle On peut de nouveau relâcher cette propriété en ne considérant que des algorithmes P' efficaces et en supposant certains problèmes difficiles (comme le problème du logarithme discret). Il s'agit alors de sécurité calculatoire conditionnelle.

7.1 Premier protocole résolvant `nonIsomorphisme`

Ce protocole est basé sur le problème `nonIsomorphisme`.

`ENTRÉE : deux graphe` G_1 `et` G_2
`SORTIE :` *oui* `si` G_1 `et` G_2 `ne sont pas isomorphes.`

Définition 9 (Isomorphisme) Rappelons que deux graphes (U, D) et (V, E) avec $D \subseteq U^2$ et $E \subseteq V^2$ sont *isomorphes* si il existe une bijection $f : U \to V$ telle que pour tous sommets $x \in U$ et $y \in U$ on ait : $(x, y) \in D \Leftrightarrow (f(x), f(y)) \in E$.

On ne connaît pas pour ce problème de solution (probabiliste ou non) en temps polynomial. Le problème `nonIsomorphisme` est ainsi considéré comme difficile. Le protocole issu de ce problème est le celui définit sur la figure 7.3.

Fait 34 Le protocole $< P, V >$ est consistant.

```
< P, V >_k
ENTRÉE : deux graphes G_1 et G_2
SORTIE : un booléen b

faire k fois
    vérificateur
        tire aléatoirement i ∈ {1,2} ;
        tire aléatoirement un graphe H isomorphe à G_i ;
        envoie H à prouveur ;
    prouveur
        calcule j ∈ {1,2} tel que H et G_j sont isomorphes ;
        envoie j à vérificateur ;
    vérificateur
        calcule verif := (i = j ∧ (j ∈ {1,2})) ;
        retourne 0 si verif = 0 ;
vérificateur
     retourne 1 ;
```

FIG. 7.3 – Protocole de preuve issu de NonIsomorphisme

preuve :
Démontrons l'équation (*cons*). Soit une instance positive $x := (G_1, G_2)$ du problème de non isomorphisme. Considérons une exécution du protocole $< P, V >$ sur l'entrée x. Soient i et H les valeurs des variables de même noms tirées par `vérificateur`. La relation d'isomorphisme est transitive. Ainsi, puisque G_1 et G_2 ne sont pas isomorphes et que H et G_i ne le sont pas, H et G_{3-i} ne le sont pas. Ainsi, `prouveur` peut exécuter l'instruction `calcule` $j \in \{0,1\}$ ... et calculer un entier j de valeur celle de i. Le booléen retourné $< P, V > (x)$ est donc vrai.

Pour prouver que ce protocole est significatif, il nous faut considérer tous les algorithmes possibles utilisables par `prouveur` pour essayer de tromper `vérificateur`. Il nous faut ainsi considérer toute l'information transmise par `vérificateur` à `prouveur` que ce dernier pourrait utiliser. Cette information est appelé une transcription.

Définition 10 Soit A l'exécution d'un protocole $< P, V >$ sur une entrée x. Pour tout instant t, la *transcription* de A à l'instant t est la séquence formée de l'entrée x et de l'ensemble des messages échangés par les deux participants aux protocoles. Elle sera notée $\mathcal{T}_{A,t}$.

Fait 35 Le protocole $< P, V >$ est significatif selon une sécurité inconditionnelle.

preuve :
Soit x une instance négative. Démontrons que pour tout algorithme P', on ait : $prob(< P', V > (x) = 1) \leq \frac{1}{2^k}$.

Supposons `prouveur` malhonnête. Il est malhonnête. Cependant, il est obligé d'envoyer à chaque passage de boucle un message j à `vérificateur`, car sinon ce serait une cause de sortie de protocole (`vérificateur` retournerait 0).

Aussi, sa partie de protocole P' utilise un algorithme D prenant en entrée toutes les informations possibles, c'est à dire à chaque instant t la transcription $\mathcal{T}_{A,t}$ où A est l'exécution courante du protocole et retournant un objet j envoyé à `vérificateur`. L'entier j envoyé par `prouveur` au l^{e}passage de boucle avec $l \in [1,k]$ et est donc égal à $D(\mathcal{T}_{A,t_l})$ où t_l est l'instant après l'envoi du graphe H_l par `vérificateur`.

Notons par un symbole en gras les variables aléatoires associées aux différents objets créés. Les graphes G_1 et G_2 sont isomorphes. Aussi, puisque `vérificateur` respecte sa partie de protocole V, les variables aléatoires $\mathbf{H_l}$ et $\mathbf{i_l}$ sont indépendantes. Puisque `vérificateur` suit le protocole, il n'informe pas `prouveur` de la valeur i_l. Aussi, le protocole est-il équivalent au protocole où `vérificateur` choisirait aléatoirement un graphe H_l isomorphe à G_1 et G_2 puis après réception de j_l tirerait aléatoirement l'entier i_l et vérifierait $i_l = j_l$. Dans un tel protocole, $\mathbf{i_l}$ et $\mathbf{j_l}$ sont indépendantes. Aussi, la probabilité $prob(i_l = j_l)$ est égale à $prob(i_l = 1 \wedge j_l = 1) + prob(i_l = 2 \wedge j_l = 2)$ c'est à dire $(prob(i_l = 1) + prob(i_l = 2)) \cdot \frac{1}{2} \leq \frac{1}{2}$.

Or, l'événement $< P', V > (x) = 1$ est égal à l'événement $\bigwedge_{l \in [1,k]} i_l = j_l$ et a une probabilité au plus égale à $\frac{1}{2^k}$. Ainsi, tout algorithme P' vérifie $prob(< P', V > (x) = 1) \leq \frac{1}{2^k}$.

L'étude précédente suggère plusieurs remarques :

Remarque 36 L'instruction $verif := (i = j \wedge (j \in \{1,2\}))$; pourrait être remplacée par $verif := (i = j)$; et offrir un protocole ayant même sécurité.

Remarque 37 `prouveur` si il est malhonnête n'a pas à s'épuiser en cherchant un algorithme lui permettant de tromper `vérificateur`. Il suffit de choisir à chaque passage de boucle $i_l = 1$. Cet algorithme lui assure une identification avec une probabilité optimale $\frac{1}{2^k}$. De même si il choisissait $i_l = 2$, ou d'ailleurs tout algorithme retournant une valeur dans $\{1,2\}$.

Remarque 38 La significativité repose sur l'aspect aléatoire du tirage de i. Si `prouveur` est capable de prédire le tirage booléen de `vérificateur` avec une probabilité p autre que $\frac{1}{2}$, la significativité est remise en cause.

Fait 39 Le protocole $< P, V >$ est à divulgation nulle.

preuve :
Ce protocole est clairement à divulgation nulle, car si `prouveur` respecte le protocole, l'entier j qu'il retourne est nécessairement égal à l'indice du graphe dont on a envoyé une copie isomorphe. L'information retournée par `prouveur` est déjà connue par `vérificateur`.

Ce protocole a pour avantage la simplicité de son écriture, mais a pour défaut de ne pas offrir d'implémentation efficace. En, effet on ne sait pas décider en temps polynomial (au moment où sont écrits ces lignes) si deux graphes sont isomorphes.

7.2 Deuxième protocole résolvant isomorphisme

La figure 7.4 définit un second protocole de preuve à divulgation nulle qui admet une implémentation efficace.

$< P, V >_k$
ENTRÉE : deux graphes G_1, G_2
SORTIE : un booléen

prouveur construit un isomorphisme $f : G_2 \to G_1$;

Faire k fois
    prouveur choisit aléatoirement un renommage g des sommets ;
             calcule le graphe $H := g(G_1)$ isomorphe à G_1 ;
             envoie H à vérificateur ;

    vérificateur choisit aléatoirement $j \in \{1, 2\}$;
             envoie j à prouveur ;

    prouveur calcule $h := g$ si $j = 1$ ou $h := g \circ f$;
             envoie h à vérificateur ;

    vérifica calcule $verif := (h$ est un isomorphisme $G_j \to H)$;
             retourne 0 si $verif = 0$;

vérificateur retourne 1 ;

FIG. 7.4 – Protocole de preuve issu de NonIsomorphisme

La preuve des deux prochains faits est laissé au lecteur.

Fait 40 Le protocole $< P, V >$ est consistant.

Fait 41 Le protocole $< P, V >$ est significatif selon une sécurité inconditionnelle.

Contrairement à l'exemple du premier protocole, où vérificateur recevait comme information j qu'il avait lui-même précédemment calculée, ce protocole fournit à vérificateur des informations nouvelles pour lui (ici des graphes isomorphes à G_1 ainsi que des isomorphismes h). Se pourrait-il qu'il puisse extraire

de ces informations une information qu'il n'aurait pu calculer avant le protocole ? On peut même imaginer que `vérificateur` triche et choisisse un entier j non pas aléatoirement mais de telle façon à obtenir cette information.

On démontre que pour ce protocole il n'en est rien. On prouve que quel que soit le sous protocole utilisé V' par un vérificateur malhonnête ou non, il existe un algorithme S' qui produit avec la même loi de probabilité l'ensemble des messages échangées entre `prouveur` et `vérificateur`. Cet algorithme est appelé un *simulateur*. Ainsi, les seules informations que reçoit `vérificateur` au cours du protocole $< P, V' >$ auraient pu être calculées directement par `vérificateur` et ce sans communiquer avec `prouveur`.

La définition formelle est la suivante :

Définition 11 (à divulgation nulle) Un protocole $< P, V >$ est à divulgation nulle si pour toute instance positive x, si pour tout algorithme V' il existe un algorithme S, appelé *simulateur*, tel que pour toute transcription T, on ait l'égalité des probabilités suivantes :

$$prob(< P, V' > (x) \rightarrow \mathcal{T}) = prob(S(x) \rightarrow \mathcal{T})$$

Fait 42 Le protocole de preuve `Isomorphisme` est à divulgation nulle selon une sécurité inconditionnelle.

preuve :
Soit `V'` un algorithme utilisé par `Vérificateur`. La seule liberté que peut prendre `vérificateur` est de choisir non pas aléatoirement l'entier $j \in \{1, 2\}$ envoyé à `prouveur` mais de le calculer selon un algorithme noté C' qui prend en entrée toutes les informations échangées entre les deux participants, c'est à dire qui calcule $j \in \{1, 2\}$ en appelant la routine $C'(T, H)$ où T est la transcription. Le protocole $< P, V' >$ est donc celui de la Figure 7.5. Soit `S'` le simulateur défini sur la Figure 7.6. Pour démontrer que toute transcription a la même probabilité d'être produite par le protocole $< P, V' >$ ou d'être produite par le simulateur S', on effectue un raisonnement par récurrence sur le nombre de tours. Supposons que pour un entier $1 \leq K \leq k$ toute transcription a la même probabilité d'être produite lors des $K - 1$ premiers tours de boucles du protocole $< P, V' >$ notés $< P, V' >_{<K}$ ou lors des K premiers tours de boucle du simulateur S' noté $S'_{<K}$.

Soit U une séquence de messages. Si U n'est pas de la forme (T, H, j, h) avec $j \in \{1, 2\}$ et h isomorphisme de G_j vers H, les probabilités $prob(< P, V' >_{\leq K} \rightarrow T)$ et $prob(S'_{\leq K} \rightarrow T)$ sont nulles et donc égales. Considérons donc le cas contraire. Clairement, on a :

- $prob(< P, V' >_{\leq K} \rightarrow U) = prob(< P, V' >_{<K} \rightarrow T) \cdot prob(< P, V' >_K \rightarrow (H, j, h))$
- $prob(S'_{\leq K} \rightarrow U) = prob(S'_{<K} \rightarrow T) \cdot prob(S'_K \rightarrow (H, j, h))$.

L'hypothèse de récurrence entraine $prob(< P, V' >_{<K} \rightarrow T) = prob(S'_{<K} \rightarrow T)$. Pour conclure, il nous suffit donc d'établir :

$$prob(< P, V' >_K \rightarrow (H, j, h)) = prob(S'_K \rightarrow (H, j, h))$$

$< P, V >_k$
ENTRÉE : deux graphes G_1, G_2

prouveur construit un isomorphisme $f : G_2 \to G_1$;
vérificateur calcule $T := ((G_1, G_2))$;
Faire k fois
    prouveur choisit aléatoirement un renommage g des sommets ;
            envoie $H := g(G_1)$ à vérificateur ;
    vérificateur calcule $j := C'(T, H)$ et $T := (T, H, j)$;
            envoie j à prouveur ;
    prouveur calcule $h := g$ si $j = 1$ ou $h := g \circ f$ sinon ;
            envoie h à vérificateur ;
    vérifica calcule $T := (T, h)$;
            calcule $verif := (h$ est un isomorphisme $G_j \to H)$;
            retourne 0 si $verif = 0$;
vérificateur retourne 1 ;

FIG. 7.5 – protocole $< P, V' >$

Soit n le nombre de sommets. Le nombre de renommages de sommets est exactement n !. Ainsi la probabilité de tirer un isomorphisme g est $\frac{1}{n\,!}$. Puisque le tirage de g est aléatoire et que le graphe H est entièrement déterminé par j et h ($H = h(G_1)$ si $j = 1$ et $H = h(f^{-1}(G_1))$ sinon) on a :

$$prob(< P, V' >_K \to (H, j, h)) = prob(C'(T, H) = j) \cdot \frac{1}{n\,!}$$

Intéressons nous au simulateur. La probabilité qu'un quadruplet de la forme (i, h, H, j) avec $i, j \in \{1, 2\}$ et h isomorphisme de G_j vers H soit produit par la séquence d'instructions :

tirer aléatoirement $i \in \{1, 2\}$
choisir aléatoirement une permutation h sur V ;
calculer $H := h(G_i)$;
calculer j := C'(T, H)

est $\frac{1}{2} \cdot \frac{1}{n!} \cdot prob(C'(T, H) = j)$. Notons p la probabilité de l'événement $i = C'(T, H)$. La probabilité qu'un quadruplet (i, h, H, j) soit produit par la K^{e} boucle de V' est $\frac{1}{2} \cdot \frac{1}{n!} \cdot prob(C'(T, H) = j) \cdot (p + (1-p) \cdot p + (1-p)^2 \cdot p + (1-p)^3 \cdot p + \ldots)$. Conséquence de l'égalité $p \cdot \sum_{l \geq 0} (1-p)^l = p \cdot \frac{1}{1-(1-p)} = 1$, cette probabilité est égale à $\frac{1}{2} \cdot \frac{1}{n!} \cdot prob(C'(T, H) = j)$. Ainsi, la probabilité qu'un triplet de la forme (h, H, j) est égal à :

$$prob(S'_K \to (H, j, h)) = prob(C'(T, H) = j) \cdot \frac{1}{n\,!} = prob(< P, V' >_K \to (H, j, h))$$

```
ENTRÉE : deux graphes G_1 = (V, D) et G_2 = (V, E)
SORTIE : une transcription
T :=((G_1,G_2)) ;
Faire k fois
    Faire
        tirer aléatoirement i ∈ {1, 2}
        choisir aléatoirement une permutation h sur V ;
        calculer H := h(G_i) ;
        calculer j := C'(T, H)
        si j = i alors
            T := concaténer(T, (H, j, h)) ;
    jusqu'à i = j ;
retourner(T) ;
```

FIG. 7.6 – simulateur de $< P, V' >$

7.3 Un 3^{e}protocole résolvant 3-coloriabilité

L'intérêt de ce protocole est de fournir à partir d'un problème NPComplet 3-coloriabilité c'est à dire :

ENTRÉE : un graphe G
SORTIE : *oui* si G est 3-coloriable.

un protocole de preuve à divulgation nulle. Ce résultat est important, car il permet de prouver que tout problème de NP peut donner naissance à un protocole à divulgation nulle. Il faut préciser toutefois que ce protocole utilise un sous protocole de mise en gage. Ainsi, sa sécurité est conditionnée à la sécurité du protocole de mise en gage utilisée. Le protocole défini par la Figure 7.7 sera étudié en TD.

$< P, V >_k$
ENTRÉE : un graphe G

prouveur calcule un coloriage f de G ;

Faire k fois
    prouveur choisit aléatoirement une permutation σ sur $[1, 3]$;
            calcule $g := \sigma \circ f$;
            met en gage séparément $g(v)$ pour chaque sommet v de G;

    vérificateur choisit aléatoirement une arête (u, v) de G ;
            envoie (u, v) au prouveur ;

    prouveur révèle $g(u)$ et $g(v)$ au vérificateur ;

    vérificateur calcule $verif := ((g(u) \neq g(v)) \wedge \{g(u), g(v)\} \subseteq [1, 3])$;
            retourne 0 si $verif = 0$;

vérificateur retourne 1 ;

FIG. 7.7 – protocole de preuve issu du 3-coloriage

Chapitre 8

Cryptologie quantique

Nous concluons ce cours en présentant très brièvement la dernière révolution qui a bouleversé la cryptologie. L'un des aspects négatifs de la cryptologie dite à clefs publiques est de fournir non pas une sécurité inconditionnelle mais une sécurité calculatoire voire pire : une sécurité calculatoire conditionnée à l'hypothèse que certains problèmes sont difficiles.

En cryptologie quantique, on obtient une sécurité inconditionnelle. Ceci cependant a un coût. Ces protocoles utilisent des particules élémentaires, les photons, dont le maniement est contraignant et coûteux financièrement.

Un principe fondamental en physique quantique est le principe d'incertitude d'Heisenberg qui établit l'imprévisibilité du comportement de particules élémentaires telles que le photon. Ainsi, non seulement, il est impossible de prédire le comportement d'un photon mais il est impossible même en le capturant à un instant t de connaître le comportement qu'il avait à cet instant.

Des cryptologues canadiens (Brassard et al.) ont tiré de cette incertitude physique une incalculabilité qu'ils ont mis à profit pour réaliser des protocoles à sécurité inconditionnelle.

Le modèle que nous présentons ici est un modèle simplifié qui permet de comprendre l'essentiel de ces protocoles. Nous supposerons que le mouvement d'un photon possède comme propriété une polarité (l'angle de polarisation) qui peut prendre quatre valeurs $00, 10, 01, 11$ (correspondant aux angles $0^o, 45^0, 90^o, 135^o$). Ces quatre valeurs appartiennent à deux *bases* 0 et 1 ; la première est associée aux valeurs 00 et 01 (les angles 0^o et 90^o) ; la seconde aux valeurs 10 et 11 (les angles 45^o et 135^o). Nous supposerons que la lecture de la polarité x ne peut être réalisée que de la façon suivante :

1. on choisit une base $B \in \{0, 1\}$.
2. on lit la polarité x si le bit de poids fort de x est B.
3. on lit une polarité aléatoire pris dans $\{\overline{B}0, \overline{B}1\}$ sinon.

Conséquences du principe d'Heisenberg, toute lecture de photon entraine la destruction de celui-ci, de plus il est impossible de copier un photon, c'est à dire d'en

fabriquer un ayant le même comportement.

D'un point de vue informatique, un photon peut être associé à un couple de bits (B, b), le premier formant la base $B \in \{0, 1\}$, le second étant le contenu du message. Le lecteur du message (B, b) doit ainsi choisir une base de lecture B'. Il lit le message $b' = b$ si il a choisit une bonne base de lecture $B' = B$ et lit un bit aléatoire sinon.

Remarque 43 Ce mécanisme peut s'illustrer ainsi. Supposons une boite avec deux portes 0 et 1. L'écrivain choisit une porte et y dépose un bit. Le lecteur lit le bon bit si il a choisi la bonne porte, lit un bit aléatoire sinon. Dès la lecture faite, le bit est détruit. Le lecteur n'a pas moyen de savoir si la porte choisie est la bonne ou non.

Exemple 14 Si Alice envoie le message $m =$ `00001111` dans la base $B =$ `00110011` et si Bob choisit pour base de lecture $B' =$ `01010101`, alors Bob lira comme message $m' =$ `0a0b1c1d` où a, b, c, d sont des bits tirés aléatoirement.

Fait 44 Si Oscar lit un bit sur le canal quantique et si il ne connaît pas la base B choisit aléatoirement par Alice, la probabilité qu'il devine correctement le bit est $\frac{3}{4}$.

preuve :
Soit B (resp. B') la base choisie par Alice (resp. Oscar). Soit b (resp. b') le bit envoyé par Alice (resp. lu par Oscar). Puisque Oscar ne connaît pas la base de Alice et que celle-ci est prise aléatoirement, la probabilité qu'il choisisse la bonne base est $\frac{1}{2}$. Ainsi la probabilité $p(b = b') = p(B = B') + \frac{1}{2} \cdot p(B \neq B') = \frac{3}{4}$.

8.1 Échange de message

Le protocole d'échange de messages par canal quantique consiste à utiliser le protocole dit à masque jetable (voir chapitre 1) dont on sait qu'il est inconditionnellement sûr pourvu que l'échange de clefs soit inconditionnellement sûr :

`ENTRÉE Alice :` x
`SORTIE Bob :` x
`Alice calcule la longueur` $l := |x|$;
`Alice et Bob échangent une clef` k `de longueur` l ;
`Alice envoie` $y := x + k$ `à Bob` ;
`Bob retourne` $x' := y + k$; `%on a` $x = x'$

Le protocole suivant suppose un canal qui permet de rendre publiques des informations. Ce canal écouté par tous est supposé sûr contre une attaque active : on ne peut pas modifier les données qui y circulent.

Dans le protocole ci-dessous, pour tout message m, pour tout indice $i \in \{1, |m|\}$, on note m_i le i$^{\text{e}}$bit de m et pour tout ensemble d'indices $I \subseteq \{1, |m|\}$ on

note m_I le message extrait de m en ne concevant que les bits d'indices dans I (on a : $|m_I| = |I|$). De plus envoyer sur le canal quantique un couple (B, m) où B et m sont deux mots binaires de même longueur consiste à envoyer successivement pour tout entier $i \in [1, |B|]$ le message m_i selon la base B_i. Voici le protocole d'échange de messages :

ENTRÉE Alice : un entier n
ENTRÉE Bob :
SORTIE Alice : une clef de session k de longueur n
SORTIE Bob : une clef de session k de longueur n

Alice publie une longueur n ;
Alice choisit aléatoirement une base B de longueur n ;
      choisit aléatoirement un mot m de longueur n ;
      envoie sur le canal quantique B, m ;
Bob   choisit aléatoirement une base B' de longueur n ;
      lit le message m' selon la base B' ;
Alice publie B ;
Bob   publie $I := \{i \mid B_i = B'_i\}$;
Alice partitionne aléatoirement I en I_1 et I_2 ;
      publie I_1, I_2 et $q := m_{I_1}$;
Bob   calcule $verif := (q = m'_{I_1})$;
      si $verif$
          publie ``nous n'avons pas été espionnés'';
          retourne m'_{I_2} ;
      sinon
          publie ``nous avons été espionnés'';
          quitte le protocole ;
Alice retourne m_{I_2} ;