Chiffrement, théorie de l'information et applications
Notes de complément au cours

JL Roch

Préambule. Ces notes reprennent les notions vues en cours sur l'entropie, la complexité, les fonctions à sens unique et les fonctions de hachage. Elles reprennent certains points d'un document co-rédigé par : **Jean-Guillaume Dumas, Franck Leprévost, Jean-Louis Roch, Valentin Savin, Sébastien Varrette**.

Contents

1 Fondements théoriques de la cryptographie

1.1 Théorie de l'information et modélisation du secret

En 1948, Claude E. Shannon [58] proposa une nouvelle théorie scientifique, appelée *théorie de l'information*, permettant de donner une mesure quantitative à la notion d'*information* contenue dans un message. A partir de cette théorie, il définit en 1949 [59] les bases permettant de caractériser la notion de secret. Ce paragraphe présente les principaux éléments conduisant à cette caractérisation.

1.1.1 Quantité d'information et entropie

Une chaîne de transmission numérique permet d'acheminer un message d'un point de départ à un point d'arrivée. Dans une chaîne de transmission numérique on distingue généralement la source du message, le milieu de transmission et le destinataire du message[1]. Un message numérique est une suite d'éléments émis par la source, pouvant chacun prendre une valeur parmi q possibles ; on appelle alphabet l'ensemble de ces valeurs, il sera noté $\mathcal{Q}$. Les éléments de $\mathcal{Q}$, qui peuvent être considérés comme des variables aléatoires discrètes, sont dits q-aires. Il y a plusieurs classes de modèles de source; nous allons considérer des *sources discrètes sans mémoire*, appelées simplement *sources discrètes* par la suite.
La sortie d'une telle source est une séquence d'éléments tirés dans l'alphabet $\mathcal{Q} = \{x_1, \ldots, x_q\}$. Chaque élément de l'alphabet est choisi aléatoirement d'après une loi de probabilité $P(x_1), \ldots, P(x_q)$ indépendante du temps, *i.e.* de la position de l'élément dans la séquence. En d'autres termes, une telle source est une variable aléatoire discrète, que nous allons noter X par la suite. Le support de X est l'ensemble des éléments de $\mathcal{Q}$ de probabilité non nulle[2].

$$\mathrm{supp}(X) = \{x \in \mathcal{Q} \mid P(x) > 0\}$$

La théorie de l'information formalise une remarque simple : plus l'événement donné par la source est probable, moins la quantité d'information correspondante est grande. Il apparaît donc qu'il existe un lien entre la *quantité d'information* fournie par une source et la distribution de probabilité de l'alphabet de cette source. Ainsi, on définit la quantité d'information d'un élément $x \in \mathrm{supp}(X)$ par :

$$I(x) = -\log_2 P(x) = \log_2 \frac{1}{P(x)}$$

La valeur moyenne de l'information fournie par l'ensemble des éléments de $\mathrm{supp}(X)$ joue un rôle très important dans la théorie de l'information.

[1] Le codage et décodage de source et le codage et le décodage de canal sont les degrés de liberté pour réaliser le système de transmission.

[2] Il s'agit en fait du support de la loi de probabilité de X. Il est également noté $\mathrm{supp}P(X)$ par d'autres auteurs.

Définition 1.1. *L'*entropie *d'une source discrète X est la valeur moyenne $H(X)$ de l'information fournie par l'ensemble des éléments de son support :*

$$H(X) = -\sum_{x \in \mathrm{supp}(X)} P(x) \log_2 P(x).$$

On peut également définir $H(X)$ par la somme des quantités d'information de tous les éléments x de la source X. Pour cela, lorsque $P(x) = 0$, on pose $P(x) \log_2 P(x) = 0$. Aussi, pour alléger l'écriture, nous omettrons parfois de préciser $x \in \mathrm{supp}(X)$.
Exemple. Si X est une source binaire qui émet 0 avec une probabilité p, alors

$$H(X) = -p \log_2 p - (1-p) \log_2 (1-p)$$

La fonction $H(p) = -p \log_2 p - (1-p) \log_2 (1-p)$ est appelée *fonction d'entropie binaire.*

1.1.2 Propriétés de l'entropie - Entropie conjointe et conditionnelle

L'entropie est une notion fondamentale pour la transmission d'information, en particulier pour la compression et la correction d'erreurs (codage). Mais nous ne présentons ici que les propriétés essentielles qui seront utilisées pour modéliser le secret (§1.1.3).

Théorème 1.1 (Première inégalité de l'entropie). *Soient X une source discrète et q le nombre d'éléments de* $\mathrm{supp}(X)$. *Alors*

$$0 \leq H(X) \leq \log_2(q)$$

De plus,

- $H(X) = 0$ *si et seulement s'il existe* $x \in \mathrm{supp}(X)$ *tel que* $P(x) = 1$ (*i.e.* $q = 1$),
- $H(X) = \log_2(q)$ *si et seulement si* $P(x) = \dfrac{1}{q}$ *pour tout* $x \in \mathrm{supp}(X)$.

Preuve. On a $H(X) = \displaystyle\sum_{x \in \mathrm{supp}(X)} P(x) \log_2 \frac{1}{P(x)} \geq 0$. De plus, $H(X) = 0$ si et seulement si tous les termes de cette somme sont nuls. Or, pour $x \in \mathrm{supp}(X)$, $P(x) \log_2 \dfrac{1}{P(x)} = 0$ si et seulement si $P(x) = 1$.
Pour la deuxième inégalité, notons $f(p_1, \ldots, p_q) = -\sum_{i=1}^{q} p_i \log_2 p_i$. Il suffit alors de prouver que le maximum de f, sous la contrainte $p_1 + \cdots + p_q = 1$, vaut $\log_2 q$. Cela revient à étudier les extrema de la fonction

$$\varphi(p_1, \ldots, p_{q-1}) = -\sum_{i=1}^{q-1} p_i \log_2 p_i - (1-(p_1+\cdots+p_{q-1})) \log_2 (1-(p_1+\cdots+p_{q-1}))$$

Les dérivées partielles de φ,

$$\frac{\partial \varphi}{\partial p_i} = -\frac{1}{\ln 2}(\ln p_i - \ln(1 - (p_1 + \cdots + p_{q-1}))),$$

s'annulent si $p_1 = \cdots = p_{q-1} = 1 - (p_1 + \cdots + p_{q-1})$. Il s'ensuit que φ admet un point critique pour $p_1 = \cdots = p_{q-1}(= p_q) = \frac{1}{q}$. On vérifie facilement que la matrice Jacobienne de φ est négative définie en ce point critique, donc qu'il correspond à un point de maximum. De plus, le maximum vaut $-\sum_{i=1}^{q} \frac{1}{q} \log_2 \frac{1}{q} = \log_2 q$. □.

Mathématiquement, un n-uplet de variables aléatoires discrètes $(X_1, \ldots, X_n)$ s'identifie à une seule variable aléatoire discrète dont le support est

$$\mathrm{supp}(X_1, \ldots, X_n) = \{(x_1, \ldots, x_n) \mid P(x_i, \ldots, x_n) > 0\}$$

Remarquons que lorsque les n variables sont mutuellement indépendantes, on obtient :

$$\begin{aligned} \mathrm{supp}(X_1, \ldots, X_n) &= \{(x_1, \ldots, x_n) \mid P(x_i) \cdots P(x_n) > 0\} \\ &= \prod_{i=1}^{n} \mathrm{supp}(X_i) \end{aligned}$$

Définition 1.2. *L'entropie conjointe de n sources discrètes $X_1, \ldots, X_n$ est définie par :*

$$H(X_1, \ldots, X_n) = - \sum_{(x_1, \ldots, x_n) \in \mathrm{supp}(X_1, \ldots, X_n)} P(x_1, \ldots, x_n) \log_2 P(x_1, \ldots, x_n)$$

Proposition 1.1. *Si les n sources sont mutuellement indépendantes, on a :*

$$H(X_1, \ldots, X_n) = \sum_{i=1}^{n} H(X_i)$$

Preuve. Il suffit de vérifier l'égalité pour $n = 2$; un simple raisonnement par récurrence permet ensuite d'établir l'égalité pour un nombre arbitraire de sources. Pour deux sources indépendantes X_1 et X_2 on obtient :

$$\begin{aligned} H(X_1, X_2) &= -\sum_{x_1} \sum_{x_2} P(x_1) P(x_2) \left(\log_2 P(x_1) + \log_2 P(x_2)\right) \\ &= \left(\sum_{x_1} P(x_1) \log_2 P(x_1)\right) \left(\sum_{x_2} P(x_2)\right) + \\ &\quad \left(\sum_{x_2} P(x_2) \log_2 P(x_2)\right) \left(\sum_{x_1} P(x_1)\right) \\ &= H(X_1) + H(X_2) \end{aligned}$$

sachant que $\sum_{x_1} P(x_1) = \sum_{x_2} P(x_2) = 1$. L'extension par récurrence, qui suit le même principe, est directe. □

Définition 1.3. *Soient X et Y deux sources discrètes. L'entropie conditionnelle de X étant donné l'événement $Y = y$ est définie par :*

$$H(X \mid Y = y) = - \sum_{x \in \text{supp}(X|y)} P(x \mid y) \log_2 P(x \mid y)$$

L'entropie conditionnelle de la source X étant donnée Y est définie par :

$$H(X \mid Y) = - \sum_{y \in \text{supp}(Y)} P(y) H(X \mid Y = y)$$

Théorème 1.2 (Règle de la chaîne). *Etant données n sources discrètes $X_1, \ldots, X_n$, l'égalité suivante est satisfaite :*

$$H(X_1, \ldots, X_n) = H(X_1) + H(X_2 \mid X_1) + \cdots + H(X_n \mid X_1, \ldots, X_{n-1})$$

Preuve. La preuve est donnée dans le cas $n = 2$. Le cas général en découle par un simple raisonnement par récurrence.

En remarquant que pour tout élément x_1 fixé $\sum_{x_2} P(x_2 \mid x_1) = 1$, on peut écrire :

$$\begin{aligned}
H(X_1) + H(X_2 \mid X_1) &= -\sum_{x_1} P(x_1) \log_2 P(x_1) \\
&\quad -\sum_{x_1} P(x_1) \sum_{x_2} P(x_2 \mid x_1) \log_2 P(x_2 \mid x_1) \\
&= -\sum_{x_1} P(x_1) \log_2 P(x_1) \sum_{x_2} P(x_2 \mid x_1) \\
&\quad -\sum_{x_1} P(x_1) \sum_{x_2} P(x_2 \mid x_1) \log_2 P(x_2 \mid x_1) \\
&= -\sum_{x_1, x_2} P(x_1, x_2)(\log_2 P(x_1) + \log_2 P(x_2 \mid x_1)) \\
&\quad -\sum_{x_1, x_2} P(x_1, x_2) \log_2 P(x_1, x_2) \\
&= H(X_1, X_2)
\end{aligned}$$

□

Théorème 1.3 (Deuxième et troisième inégalités de l'entropie). *Soient X, Y et Z trois sources. Alors :*

$$0 \leq H(X \mid Y) \leq H(X)$$

avec égalité si et seulement si X et Y sont indépendantes.

$$0 \leq H(X \mid Y, Z) \leq H(X \mid Z)$$

avec égalité si et seulement si $X \mid Z$ et $Y \mid Z$ sont indépendantes, i.e. *pour tout x, y, z on a $P(x, y \mid z) = P(x \mid z)P(y \mid z)$.*

Preuve. Il suffit de prouver la première inégalité, la deuxième en est une conséquence. Nous allons utiliser l'inégalité suivante, appelée parfois inégalité IT :

$$\log_2 t \leq (t-1)\log_2 e, \ \forall t > 0$$

avec égalité si et seulement si $t = 1$. Sa vérification est un simple exercice de recherche d'extrema.

Le fait que $0 \leq H(X \mid Y)$ découle directement de la définition de $H(X \mid Y)$. En remarquant que $\sum_y P(y \mid x) = 1$ (x fixé) on peut écrire :

$H(X) - H(X \mid Y) =$

$$\begin{aligned}
&= -\sum_x P(x)\log_2 P(x) + \sum_y P(y) \sum_x P(X \mid y)\log_2 P(X \mid y) \\
&= -\sum_x P(x)\log_2 P(x) \sum_y P(y \mid x) + \sum_{x,y} P(x \mid y)P(y)\log_2 P(X \mid y) \\
&= -\sum_{x,y} P(x,y)\log_2 P(x) + \sum_{x,y} P(x,y)\log_2 P(X \mid y) \\
&= -\sum_{x,y} P(x,y)\log_2 \frac{P(x)P(y)}{P(x,y)} \\
&\geq \sum_{x,y} P(x,y)\left(1 - \frac{P(x)P(y)}{P(x,y)}\right) \\
&= \sum_{x,y} P(x,y) - \sum_x P(x)\sum_y P(y) \\
&= 1 - 1 = 0
\end{aligned}$$

avec égalité si et seulement si $\dfrac{P(x)P(y)}{P(x,y)} = 1, \ \forall x, y$, autrement dit X et Y sont indépendantes. □

Définition 1.4. *L'information mutuelle des sources discrètes X et Y est définie par :*

$$I(X;Y) = H(X) - H(X \mid Y)$$

L'information mutuelle des sources X et Y conditionnées par Z est définie par :

$$I(X;Y \mid Z) = H(X \mid Z) - H(X \mid Y, Z)$$

Remarquons que d'après le théorème 1.2 on a

$$\begin{array}{ccccccc}
H(X,Y) & = & H(X) + H(Y \mid X) & = & H(Y) + H(X \mid Y) \\
H(X,Y \mid Z) & = & H(X \mid Z) + H(Y \mid X,Z) & = & H(Y \mid Z) + H(X \mid Y,Z)
\end{array}$$

et par conséquent

$$\begin{aligned} I(X;Y) &= I(Y;X) \\ I(X;Y \mid Z) &= I(Y;X \mid Z) \end{aligned}$$

Théorème 1.4. *Soient X, Y et Z trois sources. Alors*

$$\begin{aligned} 0 \leq I(X;Y) &\leq \min(H(X), H(Y)) \\ 0 \leq I(X;Y \mid Z) &\leq \min(H(X \mid Z), H(Y \mid Z)) \end{aligned}$$

Preuve. Les inégalités de gauche sont une conséquence du théorème 1.3. Les inégalités de droite découlent de la définition de l'information mutuelle et du fait qu'une entropie est toujours positive. □

1.1.3 Entropie et modélisation du secret

Le lien entre la notion de secret et la théorie de l'information est assez naturel. En effet, un support qui peut contenir un message secret (canal de transmission, enveloppe, ?) peut être modélisé par une *variable aléatoire*. Un message secret est alors une valeur que peut prendre cette variable parmi l'ensemble des valeurs (i.e. messages) possibles.

Considérons maintenant qu'un message clair M est chiffré en C à partir d'une clef K (symétrique ou asymétrique); suite à la remarque ci-dessus, M, C et K sont modélisés par des variables aléatoires, i.e. des sources d'information.

Supposons que le secret du message M est parfait : C ne doit donc donner aucune information sur M. Informellement, M doit rester aussi imprévisible (incertain) que l'on connaisse C ou pas, ce qui se formule grâce à la théorie de l'information :

$$H(M|C) = H(M). \tag{1}$$

Cette équation issue de la modélisation d'un chiffrement parfait entraîne des conséquences sur le choix de la clef K.

Considérons tout d'abord le cas où la clef K est symétrique. On a : $C = E_K(M)$; ainsi, connaissant K et M, il n'y a aucune incertitude sur le message C ce qui s'écrit :

$$H(C|M,K) = 0. \tag{2}$$

De même, la connaissance de C et K donne toute l'information sur le message $M = D_K(C)$, i.e. :

$$H(M|C,K) = 0. \tag{3}$$

En utilisant (1) et la règle de la chaîne (1.2), on déduit :

$$H(M) = H(M|C) \leq H(M,K|C) = H(M|K,C) + H(K|C).$$

Comme $H(M|C,K) = 0$ (3) et $H(K|C) \leq H(K)$ (1.3), on obtient :

$$H(K) \geq H(M). \tag{4}$$

Ainsi, dans tout cyrptosystème qui fournit un secret parfait, l'incertitude sur la clef secrète doit être au moins aussi grande que l'incertitude sur le texte en clair M. Ce résultat donne une borne inférieure[3] sur la taille minimale de la clef K supposée choisie aléatoirement et de manière uniforme: le nombre de bits de K doit être au moins aussi grand que le nombre de bits d'information dans le texte clair M.

Ce résultat peut être étendu au cas d'un cryptosystème à clef publique parfait. La clef K est maintenant asymétrique: K_d est secrète, connue de son seul propriétaire, alors que tout le monde connaît K_e. Une modélisation similaire à ci-dessus conduit à:

$$H(K_d|K_e) = H(K_d). \tag{5}$$

Mais, par définition d'un cryptosystème à clef publique, on a aussi : $D_{K_d} = E_{K_e}^{-1}$. La connaissance de K_e donne donc toute l'information sur K_d:

$$H(K_d|K_e) = 0.$$

On obtient donc

$$H(K_d) = 0, \tag{6}$$

autrement dit, il n'y a aucun secret sur la clef privée.

Le secret d'un cryptosystème à clef publique ne peut donc pas être caractérisé par la théorie de l'information de Shannon; ce secret ne vient pas de l'incertitude sur la clef privée K_d mais sur la difficulté intrinsèque à calculer K_d à partir de la seule connaissance de la clef K_e et de C. L'outil mathématique permettant de caractériser cette difficulté est la théorie de la *complexité algorithmique.*

1.2 Complexité algorithmique et fonctions à sens unique

Un système cryptographique à clé publique est basé sur ce qui est généralement appelé un *problème difficile.* Il faut préciser que l'adjectif *difficile* qualifie l'aspect algorithmique du problème et non pas son aspect mathématique. C'est le cas du cryptosystème RSA basé sur le problème de factorisation d'un entier ou du cryptosystème El-Gamal basé sur le problème du logarithme discret. Mathématiquement, la résolution de ces problèmes ne pose aucune difficulté et la solution naïve, basée sur la recherche exhaustive, convient parfaitement. Cependant, d'un point de vue algorithmique, cette solution est loin d'être satisfaisante car elle nécessiterait un temps de calcul beaucoup trop long.

Le but de cette section est d'introduire la notion de complexité algorithmique, ce qui permet de donner un sens précis au terme ambigu de *problème difficile* employé ci-dessus. Pour distinguer les problèmes "difficiles" des problèmes "faciles", on définit des classes de complexité qui sont des ensembles de problèmes . Nous nous limitons aux classes de complexité définies en fonction soit du nombre minimal d'opérations requises pour résoudre une instance d'un problème donné, soit de la taille de l'espace mémoire requis.

[3]La majoration $H(M) \geq H(K)$ peut sembler lourde mais est en fait atteinte. En effet, on montre facilement que pour le chiffrement de Vernam binaire (chapitre **??**), on a $H(K) = H(M)$. Donc, le chiffrement de Vernam est un exemple de chiffrement parfait.

Afin d'introduire les classes de complexité algorithmique, nous avons besoin de modéliser formellement la notion d'ordinateur et d'algorithme. Un modèle classique utilisé pour modéliser un ordinateur est la *machine de Turing*, introduit par A. Turing en 1936. Bien que ce modèle fut introduit avant la construction physique du premier ordinateur, il est toujours accepté comme le modèle de référence pour définir la complexité algorithmique.

1.2.1 Un modèle d'ordinateur : la machine de Turing

Une machine de Turing est un automate avec un nombre fini d'états et qui a la possibilité de lire et d'écrire dans une mémoire infinie. L'automate possède donc un alphabet propre, noté Γ. On peut toujours se réduire au cas d'un alphabet binaire auquel se rajoute un troisième symbole signifiant que la case mémoire vide et qui est la valeur initiale de toutes les cases. Précisément, $\Gamma = \{0, 1, b\}$, où le symbole b (blanc) correspond à une case mémoire vide. La mémoire de l'automate est constituée d'une suite infinie (à gauche comme à droite) de cases, suite qui est indexée par $\mathbb{Z}$. Finalement, l'ensemble d'états de l'automate, noté Q, est un ensemble fini contenant trois états spéciaux : un état initial q_0 et deux états finals q_r (rejet) et q_a (acceptation). Une machine de Turing est alors déterminée par une fonction de transition

$$t : (Q \setminus \{q_a, q_r\}) \times \Gamma \to Q \times \Gamma \times \{\text{gauche, droite}\}$$

qui doit être interprétée comme suit. Considérons qu'à un instant donné l'automate soit dans l'état $q \in Q$ et qu'il lise le symbole $\gamma \in \Gamma$ dans une case mémoire. Posons $(q', \gamma', d) = t(q, \gamma)$. Alors l'automate remplace le symbole γ par γ', passe dans l'état q' et sa tête de lecture/écriture se déplace dans la case mémoire à gauche ou à droite selon la valeur de d. Une telle machine s'appelle machine de Turing *déterministe*, en abrégé DTM; elle sera notée (Q, t).
D'un point de vue informel, on peut dire qu'une machine de Turing est constituée d'une partie *hardware* (alphabet, mémoire) et d'une partie *software* (la fonction de transition t qui peut être assimilée à l'implémentation d'un algorithme). Nous allons rendre ce point de vue plus clair dans la suite.

On appelle *configuration* de (Q, t) une séquence $C = xqy$, où $x, y \in \Gamma^*$ et $q \in Q$ est un état de l'automate. L'interprétation d'une telle configuration est que l'automate a la sequence xy en mémoire et qu'il est dans l'état q en train de lire la case mémoire la plus à gauche de y. Il est clair qu'en appliquant la fonction de transition t il existe une unique transition de la configuration C vers une nouvelle configuration C'. On écrira $C \to C'$. Une configuration $C = xqy$ est dite *finale* si $q \in \{q_a, q_r\}$.
On appelle *donnée d'entrée* d'une machine de Turing (Q, t) la donnée d'un mot binaire $x \in \{0, 1\}^*$. Le *calcul* de x consiste en l'unique chaîne de transitions d'états $C_0 \to C_1 \to \ldots$, telle que $C_0 = q_0 x$. Ce calcul s'arrête dès que l'on rencontre une configuration finale auquel cas le calcul est dit *fini*; sinon il est dit *infini*. Pour un calcul fini, le *temps de calcul*, noté $T(x)$, est égal au nombre de configurations de la chaîne de transitions et l'*espace mémoire du calcul*, noté

$M(x)$, est égal au nombre de cases mémoires utilisées. Il est clair que

$$M(x) \leq T(x)$$

Remarquons que la donnée d'entrée x peut être acceptée ou rejetée, selon la valeur de l'état de la configuration finale, et qu'il y a également une *donnée de sortie* y correspondant au contenu de la mémoire à l'arrêt du calcul.

D'un point de vue algorithmique une machine de Turing peut être vue comme un algorithme qui résout un problème donné. L'entrée x de la machine de Turing correspond à l'entrée de l'algorithme, les transitions données par t aux différentes étapes de l'algorithme, y à sa sortie et l'état final au succès ou à l'échec de l'algorithme. Cela nous permet par ailleurs de nous limiter aux problèmes de décision (*oui* ou *non*), c'est à dire d'ignorer y et s'intéresser seulement à l'état final de la machine q_a ou q_r. Notons également que le temps de calcul $T(x)$ correspond au nombre d'opérations binaires effectuées par l'algorithme.

1.2.2 Classes de complexité

Disposant avec la machine de Turing d'une modélisation précise d'un ordinateur, nous pouvons maintenant définir les différentes classes de complexité par rapport au temps de calcul ou à la mémoire nécessaires pour la résolution d'un problème donné et ce quel que soit l'algorithme qui le résout.

On dénotera par $|x|$ la taille de l'entrée x. A titre d'exemple, si l'entrée d'un algorithme est un entier n, alors x correspond à l'écriture binaire de n et donc $|x| = \log_2(n)$. Les classes de complexité *déterministes* principalement utilisées sont les suivantes :

LOG(TIME) – l'ensemble des problèmes $\mathcal{P}$ tels qu'il existe une DTM qui résout $\mathcal{P}$ et qui pour une entrée x prend un temps de calcul $T(x) = O(\log_2(|x|))$.

LOGSPACE – l'ensemble des problèmes $\mathcal{P}$ tels qu'il existe une DTM qui résout $\mathcal{P}$ et qui pour une entrée x prend un espace mémoire $M(x) = O(\log_2(|x|))$.

P(TIME) – l'ensemble des problèmes $\mathcal{P}$ tels qu'il existe une DTM qui résout $\mathcal{P}$ et un entier $k \in \mathbb{N}$ tels que pour toute entrée x le temps de calcul $T(x) = O(|x|^k)$.

PSPACE – l'ensemble des problèmes $\mathcal{P}$ tels qu'il existe une DTM qui résout $\mathcal{P}$ et un entier $k \in \mathbb{N}$ tels que pour toute entrée x l'espace mémoire $M(x) = O(|x|^k)$.

EXP(TIME) – l'ensemble des problèmes $\mathcal{P}$ tels qu'il existe une DTM qui résout $\mathcal{P}$ et qui pour une entrée x prend un temps de calcul $T(x) = O(1)^{|x|}$.

EXPSPACE – l'ensemble des problèmes $\mathcal{P}$ tels qu'il existe une DTM qui résout $\mathcal{P}$ et qui pour une entrée x prend un espace mémoire $/M(= O(1)^{|x|}$.

Les classes de complexité par rapport au temps de calcul seront simplement notées par LOG, P, EXP. En employant un langage moins formel, on peut

remplacer l'expression *il existe une DTM qui résout* $\mathcal{P}$ des définitions précédentes par *il existe un algorithme qui résout* $\mathcal{P}$. C'est ce que nous allons faire par la suite.

Nous n'avons parlé pour l'instant que des machines de Turing déterministes. Il existe également des machines de Turing dites *non déterministes*, en abrégé NDTM, qui sont des automates comme ceux décrits auparavant à une exception près, à savoir que l'automate est non-déterministe. Ainsi, pour une configuration donnée, il existe plusieurs transitions possibles vers d'autres configurations. Les classes de complexité *non déterministes* seront notées NLOG, NP et NEXP pour la complexité par rapport au temps de calcul, respectivement NLOGSPACE, NPSPACE, NEXPSPACE pour la complexité par rapport à l'espace mémoire utilisé. Elle sont définies de la même manière que les classes de complexité déterministes, en remplaçant les machines DTM par des machines NDTM.
Nous avons déjà remarqué qu'il est possible de définir les classes de complexité en se restreignant aux problèmes de décision. Dans ce cas on peut donner une définition équivalente des classes de complexité non déterministes, en évitant l'utilisation des NDTM. Ainsi, si $\mathcal{E}$ est une classe de complexité déterministe, on définit la classe N$\mathcal{E}$ comme l'ensemble des problèmes de décision $\mathcal{P}$ tel qu'*un certificat du "oui" soit vérifiable par un algorithme de complexité* $\mathcal{E}$. De la même manière on définit la classe CoN$\mathcal{E}$ comme l'ensemble des problèmes de décision $\mathcal{P}$ tel qu'*un certificat du "non" soit vérifiable par un algorithme de complexité* $\mathcal{E}$.
Pour mieux comprendre le sens du terme *certificat* employé dans les définitions précédentes, nous allons considérer deux exemples. Le premier est celui de la factorisation d'un entier. L'entrée du problème consiste en un entier $p \in \mathbb{N}$. La sortie du problème est "oui" si un couple d'entiers (p_1, p_2), tel que $p_1 p_2 = p$, a été trouvé et "non" dans le cas contraire. Un certificat du "oui" est un couple d'entiers (p_1, p_2) tels que $1 < p_1, p_2 < p$. La vérification du certificat consiste à calculer le produit des deux entiers. Si $p_1 p_2 = p$ le certificat est valide sinon il est faux. Il est clair qu'une telle vérification prend un temps polynomial par rapport à la taille de p (*i.e.* $\log_2 p$), donc "Factorisation" appartient à NP.
Le deuxième problème que nous allons considérer est celui de la primalité, donc le contraire du premier problème. L'entrée du problème est un entier $p \in \mathbb{N}$ et la sortie est "oui" si l'entier est premier ou "non" sinon. Cette fois un couple d'entiers comme ci-dessus devient un certificat du "non". La vérification du certificat est la même ; si $p_1 p_2 = p$ le certificat du "non" est valide, sinon il est faux. Donc le problème "Primalité" appartient à CoNP.
En fait, nous venons de remarquer qu si un problème $\mathcal{P}$ appartient à une classe non-déterministe N$\mathcal{E}$ alors le problème contraire $\bar{\mathcal{P}}$ appartient à CoN$\mathcal{E}$
Remarquons qu'il a été démontré récemment que le problème "Primalité" appartient à P [**?**]. Autrement dit il existe un algorithme en temps polynômial, décidant de la primalité d'un entier donné. Néanmoins, on ne connaît pas d'algorithme en temps polynomial pour résoudre le problème "Factorisation". C'est pourquoi il est considéré comme un *problème difficile* et la complexité de "Factorisation" est directement reliée, comme nous le verrons, à la robustesse du cryptosystème RSA.

Supposons maintenant que $\mathcal{E} = \mathcal{E}(\text{TIME})$ soit une classe de complexité déterministe par rapport au temps de calcul. Alors,

$$\mathcal{E} \subseteq \text{N}\mathcal{E} \subseteq \mathcal{E}\text{SPACE} = \text{N}\mathcal{E}\text{SPACE}$$

La première inclusion découle du fait qu'un automate DTM est un cas particulier d'automate NDTM. Autrement dit, si l'on est capable de calculer en un certain temps la sortie ("oui" ou "non") correspondant à une entrée x donnée, on peut alors vérifier un certificat donné dans (au plus) le même temps.
La deuxième inclusion et la troisième égalité peuvent être justifiées comme suit. Supposons que pour un problème $\mathcal{P}$ on dispose d'un algorithme capable de vérifier les certificats du "oui" d'une entrée x en temps $\mathcal{E}$ par rapport à $|x|$. Nous avons déjà remarqué que l'espace mémoire utilisé est inférieur au temps de calcul, *i.e.* $M(x) < T(x)$. Il s'ensuit que les certificats du "oui" peuvent être vérifiés en un espace mémoire $\mathcal{E}$ par rapport à $|x|$, donc $\text{N}\mathcal{E} \subseteq \text{N}\mathcal{E}\text{SPACE}$.
D'autre part, supposons que pour un problème $\mathcal{P}$ on dispose d'un algorithme capable de vérifier les certificats du "oui" d'une entrée x en un espace mémoire $\mathcal{E}$ par rapport à $|x|$. On peut alors définir un nouvel algorithme qui effectue une recherche exhaustive sur tous les certificats possibles. La sortie du problème $\mathcal{P}$, correspondant à l'entrée x, vaut "oui" si un certificat valide a été trouvé, ou "non" autrement. Bien évidemment, du point de vue du temps du calcul cet algorithme n'est pas satisfaisant. En ce qui concerne l'espace mémoire, on peut remarquer que lorsqu'un certificat a été vérifié, l'espace mémoire utilisé pour cette vérification peut être réutilisé pour vérifier un nouveau certificat. Ainsi, la recherche exhaustive n'augmente pas l'espace mémoire utilisé, donc le nouvel algorithme peut décider de la sortie correspondant à l'entrée x en un espace mémoire $\mathcal{E}$ par rapport à $|x|$. On obtient $\text{N}\mathcal{E}\text{SPACE} \subseteq \mathcal{E}\text{SPACE}$.
Or, l'inclusion $\mathcal{E}\text{SPACE} \subseteq \text{N}\mathcal{E}\text{SPACE}$ se justifie par les mêmes arguments que l'inclusion $\mathcal{E} \subseteq \text{N}\mathcal{E}$. En mettant ensemble ces inclusions on trouve

$$\text{N}\mathcal{E} \subseteq \text{N}\mathcal{E}\text{SPACE} \subseteq \mathcal{E}\text{SPACE} \subseteq \text{N}\mathcal{E}\text{SPACE}$$

et par conséquent $\text{N}\mathcal{E} \subseteq \mathcal{E}\text{SPACE} = \text{N}\mathcal{E}\text{SPACE}$

Définition 1.5. *Soit $\mathcal{E}$ une classe de complexité, déterministe ou non-déterministe, par rapport au temps de calcul ou à l'espace mémoire utilisé.*
On dit qu'un problème $\mathcal{P}$ se $\mathcal{E}$-réduit à un problème $\mathcal{R}$, et on note $\mathcal{P} \leq_{\mathcal{E}} \mathcal{R}$, s'il existe un algorithme de complexité $\mathcal{E}$ qui résout $\mathcal{P}$ en disposant des résultats d'une procédure qui résout $\mathcal{R}$ (le choix des entrées de cette procédure appartenant à l'algorithme).
On dit qu'un problème $\mathcal{R}$ est $\mathcal{E}$-dur si pour tout problème $\mathcal{P} \in \mathcal{E}$, $\mathcal{P} \leq_{\mathcal{E}} \mathcal{R}$.
On dit qu'un problème $\mathcal{R}$ est $\mathcal{E}$-complet si $\mathcal{R} \in \mathcal{E}$ et qu'il est $\mathcal{E}$-dur.

L'interprétation de cette définition est que $\mathcal{P} \leq_{\mathcal{E}} \mathcal{R}$ si le problème $\mathcal{R}$ est plus difficile que $\mathcal{P}$. Ainsi, un problème $\mathcal{R}$ est $\mathcal{E}$-dur s'il est plus difficile que tous les problèmes de la classe $\mathcal{E}$.
Notons que si $\mathcal{R} \in \mathcal{E}$ et $\mathcal{P} \leq_{\mathcal{E}} \mathcal{R}$ alors en mettant ensemble la procédure qui résout $\mathcal{R}$ et l'algorithme qui résout $\mathcal{P}$ et en disposant des résultat de cette

procédure, on obtient un nouvel algorithme qui résout $\mathcal{P}$ et dont la complexité est $\mathcal{E}$. Ainsi :

$$(\mathcal{R} \in \mathcal{E} \text{ et } \mathcal{P} \leq_{\mathcal{E}} \mathcal{R}) \Rightarrow (\mathcal{P} \in \mathcal{E})$$

L'exemple standard d'un problème NP-complet est le problème du *sac à dos* qui est le suivant. Etant donnés un ensemble d'entiers $\mathcal{S} = \{a_1, a_2, \ldots, a_n\}$ et un entier M, existe-t-il un sous-ensemble $\{a_{i_1}, \ldots, a_{i_p}\} \subseteq \mathcal{S}$ tel que $\sum_{k=1}^{p} a_{i_k} = M$? Un certificat du "oui" est un ensemble d'entiers $\{b_1, \ldots, b_p\}$. La vérification du certificat consiste à vérifier que chaque $b_i \in \mathcal{S}$ et que $\sum_{k=1}^{p} b_k = M$ ce qui nécessite un temps de calcul polynômial par rapport à la taille des données d'entrée (à savoir, $\sum_{i=1}^{n} \log_2 |a_i| + \log_2 |M|$). Il s'ensuit que le problème du *sac à dos* appartient à NP. On peut également montré qu'il est NP-dur, donc NP-complet. Plus de détails sur ce sujet, ainsi qu'une liste de 300 problèmes NP-complets, peuvent être trouvés dans l'ouvrage de référence sur les problèmes NP-durs et NP-complets [27].

D'autres classes de complexité existent pour d'autres modèles de machine en particulier les machines de Turing probabilistes qui trouvent leur application pratique avec les algorithmes probabilistes, comme le test de Miller-Rabin (§**??**) par exemple. La technique de réduction et la notion de complétude s'étend à ces différentes classes de manière analogue à ce qui a été fait ici dans le cas de P et NP. Grâce à cet outil, il est possible de montrer qu'un problème donné est au moins aussi coûteux qu'à résoudre (en ordre) qu'un grand ensemble de problèmes. Cela permet de formaliser la notion de difficulté intrinsèque d'un problème, indépendamment de l'algorithme qui le résout. Cette difficulté intrinsèque est le fondement théorique de la cryptographie à clef publique à partir des fonctions à sens unique.

1.2.3 Classes de complexité et cryptographie à clef publique

Dans un système de chiffrement à clef publique, le fait que la fonction de chiffrement E soit publique alors que la fonction de déchiffrement D soit secrète semble à priori contradictoire : si l'on connaît E, on connaît forcément D puisque $D = E^{-1}$. En fait, en remplaçant "inconnu" par "extrèmement long à calculer sur un ordinateur" (i.e. plusieurs années en temps par exemple ou 10^{20} opérations à l'échelle de la terre ou 10^{100} opérations à l'échelle de l'univers, ...), les fonctions E et D d'un système de cryptographie à clef publique doivent vérifier :

- $D = E^{-1}$ pour garantir $D(E(M)) = M$;
- il est facile (i.e. très rapide) de calculer $C = E(M)$ à partir de M;
- il est difficile (i.e. très long) de retrouver M à partir de C si l'on ne connaît pas K_d. Par contre, D_{K_d} doit être facile (rapide) à évaluer pour pouvoir déchiffrer le message puisque $M = D(C)$.

Cette définition intuitive peut être formalisée grâce à la théorie de la complexité, en considérant les fonctions E et $D = E^{-1}$ comme des problèmes.

Dire que E est facile à calculer revient à dire que sa complexité intrinsèque (i.e. le coût minimal d'un algorithme calculant E) est faible. Idéalement, la complexité du problème E doit être linéaire en temps (i.e. $\mathcal{O}(|x|)$ où x est l'entrée) ou, plus généralement en se basant sur les classes de complexité vues précédemment, au plus polynomiale :

$$E \in \mathrm{P(TIME)} = \mathcal{P}.$$

Dire que $D = E^{-1}$ est difficile à calculer revient à dire que sa complexité intrinsèque est considérable devant celle de E, donc typiquement non-polynomiale :

$$D \notin \mathrm{P},$$

Une telle fonction E est appelée *fonction à sens unique*. Idéalement, la complexité de D devrait être au moins exponentielle ($D \in$EXP(TIME)).

Pour la cryptographie à clef publique, il est en plus nécessaire que le propriétaire de la clé K_d possède en plus un algorithme rapide qui, connaissant K_d, calcule D. On dit que E est une *fonction à sens unique chausse-trappe* (ou à porte cachée), K_d jouant le rôle de la trappe secrète. En terme de complexité, K_d joue le rôle du certificat vu précédemment avec les classes de complexité non déterministe (§1.2.2). Ainsi, en utilisant la classe $\mathcal{P}$ comme classe des problèmes faciles à résoudre, il faudrait choisir les problèmes E et D tels que :

$$\begin{cases} \text{le problème } P_E : \textit{ calculer } E \textit{ avec en entrée } K_e \textit{ et } M \text{ appartient à } \mathcal{P}; \\ \text{le problème } P_D : \textit{ calculer } D \textit{ avec en entrée } K_e \textit{ et } M \text{ n'appartient pas à } \mathcal{P}; \\ \text{le problème } P_{D_{\text{trappe}}} : \textit{ calculer } D \textit{ avec en entrée } K_d \textit{ et } M \text{ appartient à } \mathcal{P}; \end{cases} \tag{7}$$

En assimilant K_d à un certificat, et sous l'hypothèse (non prouvée à ce jour) que $\mathcal{P} \neq N\mathcal{P}$, un bon candidat est donc une fonction à sens unique chausse-trappe telle que :

$$P_E \in \mathcal{P} \text{ et } P_D \in \mathrm{N}\mathcal{P}\text{-complet} \tag{8}$$

Une conséquence nécessaire est que l'on ne prendra jamais pour construire un système cryptographique un problème $P_E \in \mathcal{P}$ qui est tel que l'on connaît un algorithme efficace (polynomial de petit degré par exemple) pour P_D.

L'existence de fonctions à sens unique est le fondement théorique de la cryptographie à clef publique. Pourtant, même si cette existence est conjecturée, elle n'est pas prouvée à ce jour.

Et de plus, même prouvée, cette condition fondamentale est nécessaire mais non suffisante. En effet, pour casser un message, il n'est pas nécessaire de pouvoir résoudre le problème général mais il suffit d'être capable de résoudre l'instance avec l'entrée x spécifique utilisée (i.e. valeur des paramètres du cryptosystème, clé privée K_d, même éventuellement le message chiffré C) , quitte à choisir un bon algorithme pour cette instance x particulière. Aussi, même si la

complexité algorithmique est nécessaire pour la justification d'un cryptosystème à clef publique, elle ne suffit pas.

A titre d'exemple, indépendamment de la complexité théorique du problème général de la factorisation d'entiers, une très grande majorité des entiers sont peu coûteux à factoriser. Aussi, pour justifier de la sécurité d'un cryptosystème, il est nécessaire d'étudier, outre sa complexité théorique (borne inférieure), les meilleurs algorithmes permettant de le résoudre (borne supérieure) et d'évaluer précisément le coût effectif en nombre d'opérations. On considère comme impossible d'exécuter un programme qui nécessite plus de 10^{150} opérations binaires, ce qui est au-delà de l'échelle de l'univers. Aussi on s'intéresse à utiliser des instances de problème qui demandent une telle puissance de calcul. On peut considérer que les problèmes étudiés le plus à l'échelle humaine, i.e. depuis l'Antiquité, concernent l'arithmétique; ces problèmes sont particulièrement utilisés dans les cryptosystèmes à clef publique.

Dans la section §2, nous expliciterons des fonctions (conjecturées) à sens unique basées sur l'exponentielle (E) et le logarithme (D) discrets. Pour ces deux problèmes, on prouvera que $E \in \mathcal{P}$ et $D \in \mathrm{N}\mathcal{P}$.

Pour obtenir de telles fonctions, il a fallu changer la vision des messages et des clés comme de simples séquences de bits. Les méthodes modernes de cryptographie, et tout particulièrement les méthodes à clefs publiques, sont basées sur cette vision d'un message comme un entier, ou plutôt comme une suite d'entiers compris entre 0 et n. Si l'on pose $L = \lfloor \log_2 n \rfloor$, on crypte alors le message par blocs de L bits. Chaque bloc M représente alors un nombre entre 0 et $n-1$ et donc, un élément de l'anneau des classes résiduelles modulo n. Le bloc M est chiffré en $C = F(M) \pmod{n}$ où F est une fonction dans les entiers.

L'intérêt de calculer en arithmétique modulaire est double. D'une part, comme nous le verrons dans le paragraphe suivant, les calculs "modulo n" sont très rapides (de coût $O(\log^2 n)$ avec les algorithmes les plus naïfs et quasi-linéaire). D'autre part, les itérations d'une fonction F même simple calculées en arithmétique modulo n tendent à avoir un comportement aléatoire[4]. Connaissant F et n, il apparaît difficile de résoudre l'équation : trouver x tel que $F(x) \equiv a \pmod{n}$, donc d'inverser la fonction F.

Pour cela, dans le paragraphe suivant, les opérations de base en arithmétique sont présentés et la complexité effective en nombre d'opérations des meilleurs algorithmes connus pour les résoudre est analysée. Cette complexité est, comme on l'a vu dans cette section, un point fondamental en cryptographie à clef publique.

1.3 Théorie des nombres et algorithmes de base en arithmétique

L'objectif de cette section n'est pas de fournir le détail (et notamment les preuves [15, 39, 40] des résultats proposés mais d'introduire au lecteur la plupart des

[4]Ce type de calcul est d'ailleurs utilisé dans la plupart des générateurs de nombres aléatoires.

notions mathématiques utilisées dans la cryptographie à clé publique. L'accent est mis sur la complexité effective (coût en nombre d'opérations) des meilleurs algorithmes connus liés à la manipulation des nombres.

On notera $\mathbb{Z}$ l'ensemble des entiers et $\mathbb{N}$ l'ensemble des entiers positifs. Pour un entier $n \in \mathbb{Z}$ on notera $|n| = \max(n, -n) \in \mathbb{N}$ sa valeur absolue, notation à ne pas confondre avec la taille (section précédente) de l'entier n qui est $\log_2 |n|$.

1.3.1 Arithmétique modulaire

Définition 1.6. *Soient a et b deux entiers. On dit que a divise b ou encore que a est un diviseur de b et on note $a \mid b$, s'il existe un entier c tel que $b = ac$. Dans ce cas, on dit également que b est un multiple de a.*

De cette définition découlent les propriétés suivantes.

Proposition 1.2. *Soient a, b et c des entiers. Alors*

1. $a|a$
2. *si* $a|b$ *alors* $a|bc$
3. *si* $a|b$ *et* $b|c$ *alors* $a|c$
4. *si* $a|b$ *et* $a|c$ *alors* $a|b+c$

Définition 1.7 (Division Euclidienne). *Soient a et b deux entiers tels que $b \neq 0$. Il existe un unique couple d'entiers (q, r) tels que $a = bq + r$ et $0 \leq r < |b|$. On appelle q le quotient de a par b et r le reste de la division de a par b.*

Définition 1.8. *Soient a, b et n trois entiers, $n > 0$. On dit que a est congru à b modulo n et on note $a \equiv b \mod n$, si n est un diviseur de $a - b$.*

Si $n > 0$, $a \equiv b \mod n$ si et seulement si le reste de la division de a par n est égal au reste de la division de b par n.

Proposition 1.3. *Soient a, b, c, a', b' et n des entiers, $n > 0$. Alors*

1. $a \equiv a \mod n$
2. *si* $a \equiv b \mod n$ *alors* $b \equiv a \mod n$
3. *si* $a \equiv b \mod n$ *et* $b \equiv c \mod n$ *alors* $a \equiv c \mod n$
4. *si* $a \equiv a' \mod n$ *et* $b \equiv b' \mod n$ *alors* $a + a' \equiv b + b' \mod n$ *et* $ab \equiv a'b' \mod n$
5. *si* $a \equiv b \mod n$ *et* $c > 0$ *alors* $a^c \equiv b^c \mod n$
6. *si* $a \equiv b \mod n$ *et* $a, b > 0$ *alors* $c^a \equiv c^b \mod n$

D'après les trois premières propriétés de la proposition précédente, la relation de congruence modulo n est une relation d'équivalence sur $\mathbb{Z}$. On notera $\mathbb{Z}/n\mathbb{Z}$ l'ensemble quotient; dans cet ensemble, deux entiers sont identifiés s'ils sont congrus modulo n.
Ainsi, l'ensemble $\mathbb{Z}/n\mathbb{Z}$ s'identifie à l'ensemble des restes des divisions par n. Pour un entier $a \in \mathbb{Z}$ on notera $\bar{a}$ l'élément de $\mathbb{Z}/n\mathbb{Z}$ qui lui correspond ; $\bar{a}$ peut être vu comme le reste le de division de a par n. Cela nous permet d'identifier $\mathbb{Z}/n\mathbb{Z}$ à l'ensemble des entiers positifs strictement inférieurs à n :

$$\mathbb{Z}/n\mathbb{Z} = \{\bar{0}, \bar{1}, \bar{2}, \ldots, \overline{n-1}\}$$

La propriété 4 de la proposition précédente permet de définir une opération d'addition et une opération de multiplication sur $\mathbb{Z}/n\mathbb{Z}$ par :

Addition $\bar{a} + \bar{b} = \overline{a+b}$ pour tout $\bar{a}, \bar{b} \in \mathbb{Z}/n\mathbb{Z}$

Multiplication $\bar{a}\bar{b} = \overline{ab}$ pour tout $\bar{a}, \bar{b} \in \mathbb{Z}/n\mathbb{Z}$

Ces deux opérations font de $\mathbb{Z}/n\mathbb{Z}$ un *anneau commutatif unitaire*, c'est à dire que les propriétés usuelles de commutativité, associativité, élément neutre et distributivité de la multiplication par rapport à l'addition sont satisfaites.
Complexité de la multiplication et de la division euclidienne d'entiers. Pour analyser le coût des algorithmes d'arithmétique, on utilise la complexité $M(t)$ de la multiplication de deux entiers de t bits. L'algorithme de multiplication naïf, qui consiste à faire les t^2 produits bit à bit, a un coût de t^2 multiplications et de t^2 additions de bits, soit environ $2.t^2$. On a donc

$$M(t) = \mathcal{O}(t^2).$$

Un algorithme récursif [57] basé sur la transformée de Fourrier permet de réaliser la multiplication en $\mathcal{O}(t \log t \log\log t)$; on en déduit que la multiplication d'entiers a une complexité asymptotique presque linéaire :

$$M(t) = \mathcal{O}(t^{1+\epsilon})$$

avec $\epsilon > 0$ arbitrairement petit.
L'algorithme naïf de division euclidienne d'un entier a par un entier b consiste à deviner un nouveau chiffre du quotient de manière à faire décroître le dividende jusqu'à obtenir un entier inférieur à b, le reste euclidien. Cet algorithme est trivialement de coût $\mathcal{O}(\log b.(\log a - \log b))$. Cependant, un algorithme récursif de type Diviser pour Régner, basé sur une itération de Newton pour le calcul du réciproque d'un entier [34] a un coût

$$\mathcal{O}(M(\log(a+b)).$$

De plus, il est possible réciproquement de calculer la multiplication de deux entiers de t bits avec un coût de l'ordre de la division d'un entier de $2.t$ bits par un entier de t bits. On en déduit donc que le problème de la division euclidienne d'entiers a une complexité de même ordre que $M(t)$.

Convention et notation pour la complexité des opérations modulo n. En pratique, on a donc $t \leq M(t) \leq 2.t^2$ et asymptotiquement $M(t) = \mathcal{O}(t^{1+\epsilon})$ avec $\epsilon \geq 0$ arbitrairement petit. Les opérations de base modulo n (addition, soustraction, multiplication) peuvent donc être réalisés très rapidement, en temps réel presque linéaire de la taille des opérandes. Cette propriété est fondamentale en cryptographie pour réaliser les opérations de chiffrement et de déchiffrement.
Aussi, dans toute la suite, lorsqu'on manipulera un entier n, codé donc sur $t = \log_2 n$ bits, le coût des opérations arithmétiques de bases avec n (multiplication, reste euclidien modulo n) sera noté $M(\log_2 n) = \mathcal{O}(\log^{1+\epsilon} n)$ où en pratique $0 \leq \epsilon \leq 1$; asymptotiquement lorsque n grandit, ϵ tend vers 0.

1.3.2 Notion de nombre premier

Définition 1.9. *Soit p un entier positif, $p \geq 2$. On dit que p est premier si ses seuls diviseurs positifs sont 1 et p.*

Proposition 1.4. *Soit p un nombre premier et a, b deux entiers tels que $p|ab$. Alors $p|a$ ou $p|b$.*

Proposition 1.5 (Décomposition en facteurs irréductibles). *Soit a un entier strictement positif. Alors, il existe un unique entier $r > 0$, une unique famille de nombres premiers $p_1, \ldots, p_r$ et une unique famille d'entiers $n_1, \ldots, n_r$ strictement positifs tels que*

$$a = p_1^{n_1} p_2^{n_2} \cdots p_r^{n_r}$$

Cette écriture est appelée la décomposition de a en facteurs irréductibles.

1.3.3 Plus Grand Commun Diviseur (pgcd)

Définition 1.10 (pgcd). *Soit $\{a_1, \ldots, a_n\}$ une famille d'entiers dont au moins un est non nul. On dit qu'un entier d est un diviseur commun de $(a_1, \ldots, a_n)$ si $d|a_i$ pour tout $i = 1, n$.*
Si d est un diviseur commun d'une famille d'entiers $\{a_1, \ldots, a_n\}$ alors

$$d \leq \min_{a_i \neq 0} \{|a_1|, \ldots, |a_n|\}$$

Par conséquent il existe un plus grand diviseur commun qui sera noté $\mathrm{pgcd}(a_1, \ldots, a_n)$.

Définition 1.11. *Deux entiers strictement positifs a et b sont dit premiers entre eux si* $\mathrm{pgcd}(a, b) = 1$.

Proposition 1.6. *Soit $\{a_1, \ldots, a_n\}$ une famille d'entiers dont au moins un est non nul et D un entier positif diviseur commun de $(a_1, \ldots, a_n)$. Alors*

$$D = \mathrm{pgcd}(a_1, \ldots, a_n) \Leftrightarrow \forall d \text{ diviseur commun de } (a_1, \ldots, a_n) : d|D$$

Proposition 1.7. *Soit $\{a_1, \ldots, a_n\}$ une famille d'entiers dont au moins un est non nul et D le plus petit entier positif de la forme*

$$D = a_1 b_1 + \cdots a_n b_n$$

avec $b_i \in Z$ pour $i = 1, \ldots, n$. Alors $D = \text{pgcd}(a_1, \ldots, a_n)$.

1.3.4 Algorithmes d'Euclide

L'algorithme d'Euclide (voir [34] page 334) permet de calculer efficacement le pgcd de deux nombres. Il découle du théorème suivant :

Théorème 1.5 (Euclide). *Soient $a, b \in \mathbb{Z}$ tels que $b \neq 0$ et r le reste de la division euclidienne de a par b. Alors pgcd$(a, b) =$ pgcd(b, r).*

L'algorithme d'Euclide consiste alors à réitérer les manipulations suivantes :

- Effectuer la division euclidienne de a par b. Soit r le reste. On notera $r = a\%b$.
- Remplacer a par b et b par r (on a donc $0 \leq r < b$ d'après la définition de la division euclidienne).
- Le pgcd est le dernier reste non nul.

Algorithme 1: Euclide (pgcd des deux entiers)

Données : a, b entiers
Résultat : $\text{pgcd}(a, b)$
si $a < 0$ **alors** $a \leftarrow -a$;
si $b < 0$ **alors** $b \leftarrow -b$;
tant que $b > 0$ **faire**
    temp $\leftarrow b$;
    $b \leftarrow a\%b$;
    $a \leftarrow$ temp;
fin
retourner a ;

Complexité de l'algorithme. Si $a > b$, la complexité de l'algorithme d'Euclide est de $\mathcal{O}(\log_2^3 a)$. Nous allons détailler le calcul de la complexité pour l'algorithme d'Euclide étendu 2 (voir plus loin).

Notons qu'on peut utiliser l'algorithme d'Euclide de manière récursive pour calculer le pgcd d'une famille d'entiers $\{a_1, \ldots, a_n\}$ à l'aide de l'égalité suivante :

$$\text{pgcd}(a_1, \ldots, a_n) = \text{pgcd}(\ \text{pgcd}(a_1, \ldots, a_{n-1}), a_n\ )$$

Exemple Posons $a = 30$ et $b = 21$. Puisque $30 = 1 \times 21 + 9$ et $9 = 3 \times 3 + 0$, on trouve :

$$\begin{aligned} \text{pgcd}(30, 21) &= \text{pgcd}(21, 9) \\ &= \text{pgcd}(9, 3) \\ &= 3 \end{aligned}$$

Algorithme d'Euclide étendu Il existe une version étendue de l'algorithme d'Euclide, appelée "Euclide étendu", qui permet de calculer les coefficients de Bezout dont la définition est rappelée maintenant (voir aussi proposition 1.7).

Théorème 1.6 (Bezout). *Soient a, $b \in \mathbb{Z}$ dont au moins un est non nul et $d = pgcd(a, b)$. Alors il existe un couple d'entiers (u, v) tels que*

$$au + bv = d$$

Les entiers u et v sont appelés coefficients de Bezout.

L'algorithme d'Euclide étendu permet de construire effectivement les coefficients de Bezout. L'idée est la suivante. Supposons que a et b soient des entiers positifs et que $b \neq 0$ et soit $a = bq + r$ avec $0 \leq r < b$ la division euclidienne de a par b. Soient u' et v' les coefficients de Bezout de b et r. Alors :

$$\begin{aligned} d &= \text{pgcd}(a, b) = \text{pgcd}(b, r) \\ &= bu' + rv' = bu' + (a - bq)v' \\ &= av' + (u' - qv')b \end{aligned}$$

Ainsi, on peut prendre $u = v'$ et $v = u' - qv'$; les coefficients de Bezout de (a, b) peuvent donc être calculés à partir des coefficients de Bezout de (b, r). Cette remarque conduit directement à l'algorithme de calcul des coefficients de Bezout (algorithme 2).

Complexité de l'algorithme. Supposons que $a > b$. La complexité de la division euclidienne de a par b est $M(\log a) = \mathcal{O}(\log_2^2(a))$. Il reste à déterminer le nombre de divisions calculées par l'algorithme. Soient r le reste de la division de a par b et r' le reste de la division de b par r. Alors :

- si $r > \dfrac{b}{2}$ alors soit $r' = b - r < \dfrac{b}{2}$,
- si $r \leq \dfrac{b}{2}$ alors soit $r' = r \leq \dfrac{b}{2}$

Ansi $r' < \dfrac{b}{2}$, donc après deux divisions b est diminué de moitié. Le nombre maximum de divisions euclidiennes effectuées par l'algorithme est alors de $2\log_2(b)$. Il s'ensuit que le coût de l'algorithme est $\log_2(b).M(\log a) = \mathcal{O}(\log_2^2(a)\log_2(b)) = \mathcal{O}(\log_2^3 a)$. D'autres algorithmes de calcul du pgcd ont été proposés qui donnent une meilleure majoration de la complexité de ce problème. Une version

Algorithme 2: Euclide étendu (coefficients de Bezout de deux entiers)

Données : a, b entiers
Résultat : (d, u, v) tels que $d = \text{pgcd}(a, b) = au + bv$
si $a < 0$ **alors** $a \leftarrow -a$;
si $b < 0$ **alors** $b \leftarrow -b$;
$u \leftarrow 1, \quad v \leftarrow 0$;
$u_{\text{aux}} = 0, \quad v_{\text{aux}} = 1$;
tant que $b > 0$ **faire**
    effectuer la division euclidienne $a = bq + r$;
    $a \leftarrow b,\ b \leftarrow r$;
    $\text{tmp} \leftarrow u_{\text{aux}},\ u_{\text{aux}} \leftarrow u - qu_{\text{aux}},\ u \leftarrow \text{tmp}$;
    $\text{tmp} \leftarrow v_{\text{aux}},\ v_{\text{aux}} \leftarrow v - qv_{\text{aux}},\ v \leftarrow \text{tmp}$;
fin
retourner (a, u, v) ;

récursive, de type Diviser pour Régner, de cet algorithme d'Euclide étendu conduit à un algorithme [56] dont le coût est celui de $\mathcal{O}(\log\log a + b)$ multiplications d'entiers de la même taille que a et b, soit une complexité asymptotique $\log\log(a+b).M(\log a + b) = \mathcal{O}(\log_2^{1+\epsilon} a + b)$ presque linéaire en la taille des opérandes. Ainsi, le calcul du pgcd peut être effectué de manière très efficace, en temps réel.

1.3.5 Inverse modulaire

Définition 1.12. *On dit qu'un entier a est inversible modulo n s'il existe un entier b tel que $ab \equiv 1 \mod n$.*

Notons qu'un entier a est inversible modulo n si et seulement si $\bar{a}$ est un élément inversible de $\mathbb{Z}/n\mathbb{Z}$. En effet, si $ab \equiv 1 \mod n$ alors $\bar{a}\bar{b} = \overline{ab} = \bar{1}$.

Proposition 1.8. *Soient a et n deux entiers avec $n \geq 2$. Alors, a est inversible modulo n si et seulement si* $\text{pgcd}(a, n) = 1$.

De plus, l'algorithme d'Euclide étendu nous permet de calculer l'inverse de a modulo n. En effet, soient u et v les coefficients de Bezout de a et n, alors

$$d = \text{pgcd}(a, n) = au + nv$$

Donc $au - d = nv$ est divisible par n, ainsi $au \equiv d \mod n$. Si $d = 1$ on obtient $au \equiv 1 \mod n$, donc u est l'inverse de a modulo n. On notera $u = a^{-1} \mod n$.

1.3.6 Fonction indicatrice d'Euler

Proposition 1.9. *Soient n et a deux entiers avec $n \geq 2$. Les affirmations suivantes sont équivalentes :*

1. *$\bar{a}$ est un générateur du groupe additif* $(\mathbb{Z}/n\mathbb{Z}, +)$.

2. *$\bar{a}$ est un élément inversible dans l'anneau $\mathbb{Z}/n\mathbb{Z}$.*

3. $\text{pgcd}(a, n) = 1$.

On note $(\mathbb{Z}/n\mathbb{Z})^\times$ l'ensemble des éléments inversibles de $\mathbb{Z}/n\mathbb{Z}$ et par $\varphi(n)$ son cardinal. La fonction φ est appelée la fonction indicatrice d'Euler. Elle peut être calculée par la relation :

$$\varphi(n) = n \prod_{p \text{ premier}, p|n} (1 - \frac{1}{p})$$

Ainsi $(\mathbb{Z}/n\mathbb{Z})^\times$ est un groupe multiplicatif de cardinal $\varphi(n)$. Rappelons que pour tout élément g d'un groupe fini $(G, \times)$, on a $g^{\text{card}(G)} = 1$. On obtient ainsi la proposition suivante.

Proposition 1.10. *Soient a et n deux entiers tels que $n \geq 2$ et* $\text{pgcd}(a, n) = 1$. *Alors,*

$$a^{\varphi(n)} \equiv 1 \mod n$$

ce qui s'écrit aussi $\bar{a}^{\varphi(n)} = \bar{1}$ dans $\mathbb{Z}/n\mathbb{Z}$.

1.3.7 Petit théorème de Fermat

D'après la proposition 1.9, si p est un nombre premier, tous les éléments non nuls de $\mathbb{Z}/p\mathbb{Z}$ sont inversibles. Donc $\mathbb{Z}/p\mathbb{Z}$ est un corps.

Théorème 1.7. *Soit p un nombre premier; il existe un seul corps à p éléments, qui est noté $\mathbb{F}_p$. Le corps $\mathbb{Z}/p\mathbb{Z}$ sera donc noté $\mathbb{F}_p$.*

Une conséquence de la proposition 1.10 est le théorème suivant.

Théorème 1.8. *Pour tout nombre premier p et pour tout entier a, on a*

$$a^p \equiv a \mod p$$

ce qui s'écrit aussi $\bar{a}^p = \bar{a}$ dans $\mathbb{F}_p$.

1.3.8 Exponentiation rapide

Soient a et e deux entiers positifs. En remarquant que

- si e est paire, $a^e = \left(a^{\frac{e}{2}}\right)^2$
- si e est impaire $a^e = \left(a^{\frac{e}{2}}\right)^2 \cdot a$

on obtient un algorithme itératif (algorithme **??** pour calculer a^e; cet algorithme est appelé *exponentiation rapide* ou *élévation récursive au carré.*

Complexité de l'algorithme. Le produit de deux entiers a, b modulo n a une complexité $M(\log n) = \mathcal{O}(\log^{1+\epsilon} n)$. Ainsi, le calcul de $p \cdot \texttt{tmp}$ ou de $\texttt{tmp} \cdot \texttt{tmp}$ se fait en $M(\log n) = \mathcal{O}(\log_2^{1+\epsilon} n)$, car $p, \text{tmp} < n$. Sachant qu'à chaque itération l'exposant e est divisé par deux, l'algorithme exécutera $\log_2 e$ itérations. Ainsi, la complexité totale de l'algorithme est en $\mathcal{O}(M(\log n).\log_e) = \mathcal{O}(\log^{1+\epsilon} n \log e)$.

Algorithme 3: Exponentiation modulaire rapide

Données : a, e, n entiers positifs
Résultat : $a^e \mod n$
$p \leftarrow 1$;
$\text{tmp} \leftarrow a \mod n$;
tant que $e > 0$ **faire**
  si e *est impair* **alors** $p \leftarrow p \cdot \text{tmp} \mod n$;
  $\text{tmp} \leftarrow \text{tmp} \cdot \text{tmp} \mod n$;
  $e \leftarrow \lfloor \frac{e}{2} \rfloor$;
fin
retourner p ;

1.3.9 Théorème chinois des restes

Théorème 1.9 (des restes chinois). *Soient $n_1, \ldots, n_k$ des entiers strictement positifs premiers entre eux deux à deux. L'application*

$$\begin{array}{cccccccc} & \mathbb{Z}/(n_1 \cdots n_k)\mathbb{Z} & \to & \mathbb{Z}/n_1\mathbb{Z} & \times & \cdots & \times & \mathbb{Z}/n_k\mathbb{Z} \\ x & \mod (n_1 n_2 \cdots n_k) & \mapsto & (x \mod n_1) & \times & \cdots & \times & (x \mod n_k) \end{array}$$

est un isomorphisme d'anneaux.

En d'autre termes, pour toute famille d'entiers $(x_1, \ldots, x_k)$ il existe un entier x, unique modulo n, tel que $x \equiv x_i \mod n_i$ pour tout $i = 1, \ldots, k$.
L'entier x peut être trouvé par récurrence comme suit :

si $k = 1$ alors $x = x_1$ convient.

si $k = 2$ on peut écrire $n_1 u_1 + n_2 u_2 = 1$, où u_1, u_2 sont les coefficients de Bezout. On pose $x = n_1 u_1 x_2 + n_2 u_2 x_1$. Alors

$$x - x_1 = n_1 u_1 x_2 (x_2 - x_1) \quad \text{et} \quad x - x_2 = n_2 u_2 x_2 (x_1 - x_2)$$

donc x convient.

si $k = r$ et le résultat est vrai pour $k = r - 1$; donc il existe un entier x' tel que $x' \equiv x_i \mod n_i$, pour tout $i = 1, r - 1$. Notons $n' = n_1 \cdots n_{r-1}$. On peut alors trouver x en procédant comme dans le cas $k = 2$ en remplaçant :
$n_1 \leftarrow n'$ et $n_2 \leftarrow n_r$;
$x_1 \leftarrow x'$ et $x_2 \leftarrow x_r$.

1.3.10 Résidus quadratiques

Définition 1.13. *Soient a et n deux entiers avec $n > 0$. On dit que a est un carré modulo n ou encore que a est un résidu quadratique de n, s'il existe un entier b tel que $b^2 \equiv a \mod n$. On note $(\mathbb{Z}/n\mathbb{Z})^2$ l'ensemble des éléments de $\mathbb{Z}/n\mathbb{Z}$ qui sont des résidus quadratiques de n.*

Proposition 1.11. *Si p est un nombre premier impair, alors :*

1. *un entier a est un résidu quadratique de p si et seulement si* $a^{\frac{p-1}{2}} = 1 \mod p$.

2. *le cardinal de* $(\mathbb{F}_p)^2$ *est* $\dfrac{p+1}{2}$.

Notons que si $n = 2k+1$ est un entier impair, alors $n^2 - 1 = 4(k^2+k)$ est un multiple de 8, sachant que $k^2 + k$ est toujours pair. Cela permet d'énoncer la proposition suivante.

Proposition 1.12. *Soit p un nombre premier impair. Alors 2 est un résidu quadratique de p si et seulement si* $\dfrac{p^2-1}{8}$ *est pair.*

La proposition suivante est une conséquence directe du théorème des restes chinois.

Proposition 1.13. *Soient $n_1, \ldots, n_k$ des entiers strictement positifs premiers entre eux deux à deux et $n = n_1 \cdots n_k$. Alors, un entier a est un résidu quadratique de n si et seulement si a est un résidu quadratique de n_i pour tout $i = 1, \ldots, k$.*

1.3.11 Symboles de Legendre et de Jacobi

Définition 1.14 (Symbole de Legendre). *Soient $a \in \mathbb{Z}$ et p un nombre premier impair. Le symbole de Legendre, noté* $\left(\frac{a}{p}\right)$, *est défini par:*

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{si } p \text{ divise } a \\ 1 & \text{si } a \neq 0 \text{ est un résidu quadratique de } p \\ -1 & \text{sinon} \end{cases}$$

Le symbole de Legendre admet un certain nombre de propriétés qui sont listées ici:

$$\left(\frac{a}{p}\right) = a^{\left(\frac{p-1}{2}\right)} \mod p \tag{9}$$

$$\left(\frac{i}{p}\right) = i \text{ pour } i \in \{0,1\} \tag{10}$$

$$\left(\frac{-1}{p}\right) = (-1)^{\left(\frac{p-1}{2}\right)} \tag{11}$$

$$\left(\frac{a}{p}\right) = \left(\frac{a \mod p}{p}\right) \tag{12}$$

$$\left(\frac{xy}{p}\right) = \left(\frac{x}{p}\right)\left(\frac{y}{p}\right) \tag{13}$$

$$\left(\frac{2a}{p}\right) = \left(\frac{a}{p}\right)(-1)^{\left(\frac{p^2-1}{8}\right)} \tag{14}$$

$$\left(\frac{p}{q}\right)\left(\frac{q}{p}\right) = (-1)^{\left(\frac{(p-1)(q-1)}{4}\right)} \tag{15}$$

La dernière propriété s'appelle loi de réciprocité quadratique. La première propriété permet de calculer directement $\left(\frac{a}{p}\right)$ tandis que les propriétés suivantes permettent d'aboutir à un calcul récursif.

Le symbole de Jacobi étend la définition du symbole de Legendre à tous les entiers impairs.

Définition 1.15 (Symbole de Jacobi). *Soit n un entier impair. Sa décomposition en facteurs s'écrit :*

$$n = \prod_{i=1}^{k} p_i^{e_i}$$

Soit $a \geq 0$. Le symbole de Jacobi $\left(\frac{a}{n}\right)$ est défini par:

$$\left(\frac{a}{n}\right) = \prod_{i=1}^{k} \left(\frac{a}{p_i}\right)^{e_i}$$

En particulier, $\left(\frac{1}{n}\right) = 1$ et $\left(\frac{0}{n}\right) = 0$.

Plusieurs règles permettent de calculer $\left(\frac{a}{n}\right)$ en temps polynomial sans factoriser n :

$$\left(\frac{m_1}{n}\right) = \left(\frac{m_2}{n}\right) \text{ Si } m_1 \equiv m_2 \mod n \tag{16}$$

$$\left(\frac{2}{n}\right) = \begin{cases} 1 & \text{si } n \equiv \pm 1 \mod 8 \\ -1 & \text{si } n \equiv \pm 3 \mod 8 \end{cases} \tag{17}$$

$$\left(\frac{xy}{n}\right) = \left(\frac{x}{n}\right)\left(\frac{y}{n}\right) \tag{18}$$

$$\left(\frac{m}{n}\right) = \begin{cases} -\left(\frac{n}{m}\right) & \text{si } m \equiv n \equiv 3 \mod 4 \\ \left(\frac{n}{m}\right) & \text{sinon} \end{cases} \tag{19}$$

En particulier de (18), si $m = 2^s t$ avec t impair, alors $\left(\frac{m}{n}\right) = \left(\frac{2}{n}\right)^s \left(\frac{t}{n}\right)$.
L'égalité (19) correspond à la loi de réciprocité généralisée et reste valable pour m entier impair.
L'algorithme 4 implémente le calcul du symbole de Jacobi directement à partir des propriétés ci-dessus.

Complexité de l'algorithme. Les opérations effectuées par l'algorithme sont les mêmes que celles effectuées pour le calcul du pgcd (algorithme d'Euclide, algorithme 1). Sa complexité se calcule de la même manière. En supposant $n > m$, elle est majorée par $\mathcal{O}(\log_2^3 n)$ en utilisant des algorithmes naïfs et $\mathcal{O}(M(\log n).\log\log n)$ en utilisant des algorithmes asymptotiques.

Exemple :

Prenons $a = 51$ et $n = 97$. On a alors:

$$\begin{aligned} \left(\frac{51}{97}\right) &= \left(\frac{97}{51}\right) = \left(\frac{97 \mod 51}{51}\right) = \left(\frac{46}{97}\right) = \left(\frac{2}{97}\right)\left(\frac{23}{97}\right) = -\left(\frac{23}{97}\right) \\ &= \left(\frac{97}{23}\right) = \left(\frac{5}{23}\right) = \left(\frac{23}{5}\right) = \left(\frac{3}{5}\right) = \left(\frac{5}{3}\right) = \left(\frac{2}{3}\right) = -1 \end{aligned}$$

Algorithme 4: Calcul du symbole de Jacobi

Données : m, n, entiers positifs

Résultat : $J = \left(\frac{m}{n}\right)$ le symbole de Jacobi

$J \leftarrow 1$;
tant que $m \neq 1$ **faire**
  $m \leftarrow m\%n$;
  si $m = 0$ **alors retourner** $J = 0$;
  $s \leftarrow 0$;
  tant que $2 \mid m$ **faire**
    $m \leftarrow \frac{m}{2}$;
    $s \leftarrow s + 1$;
  fin
  si $n\%8 = 3$ *ou* $n\%8 = 5$ **alors** $J \leftarrow J \times (-1)^s$;
  tmp $\leftarrow m$;
  $m \leftarrow n$;
  $n \leftarrow$ tmp;
  si $m\%4 = 0$ *et* $n\%4 = 0$ **alors** $J \leftarrow J \times (-1)$;
fin
retourner J ;

2 Fonctions à sens unique - Problème du Logarithme Discret (DLP)

Nous avons vu (§1.2.3) que la cryptographie à clef publique est basée sur le fait de disposer d'une fonction à sens unique (FSU dans la suite, en anglais *one way function*). Autrement dit, il faut trouver une fonction E de chiffrement qui soit rapide à calculer mais que son inverse $D = E^{-1}$ soi extrêmement longue à calculer. Comme on l'a vu de bonnes FSU sont des fontions telles que la recherche de x à partir de $F(x)$ soit un problème mathématique réputé difficile (typiquement NP-complet).

Les sections suivantes présentent des fonctions à sens uniques qui sont couramment utilisées en cryptographie à clé publique. Ces fonctions sont essentiellement basées sur la difficulté du problème du logarithme discret; les principaux algorithmes pour résoudre ce problème sont donc présentés et leur coût en nombre d'opérations , qui caracatérise le niveau de sécurité, est analysé.

2.1 Un premier exemple de FSU : l'Exponentiation Modulaire

L'exemple le plus courant de fonction à sens unique est l'exponentiation modulaire, qui se construit comme suit:

- Soient p et q deux entiers premiers et $n = p.q$.

- Soit e un entier inférieur à n, premier avec $\phi(n) = (p-1)(q-1)$.

La fonction d'exponentiation modulaire est alors définie par :

$$\begin{array}{rccc} F_e : & \mathbb{Z}_n & \longrightarrow & \mathbb{Z}_n \\ & x & \longrightarrow & x^e \mod n \end{array}$$

Cette fonction est rapide à calculer (en fait en temps linéaire de e et donc de n) grâce à un algorithme de type exponentiation rapide, encore appelé *élévation récursive au carré* (voir §1.3.8, page 23).

La fonction inverse de F_e n'est autre que la fonction F_d, ou $d = e^{-1} \mod \phi(n)$. Autrement dit, on a : $d.e \equiv 1 \mod (p-1)(q-1)$.

En effet, si $y = F_e(x)$, alors $y^d = x^{e.d} \mod n = x^{1+\lambda\phi(n)} \mod n = x$.

Le calcul de $d = e^{-1} \mod \phi(n)$ peut être fait facilement à partir de l'algorithme d'Euclide Etendu exposé dans la section 1.3.5, page 22, puisque e et $\phi(n)$ sont premiers entre eux.

En fait, la véritable difficulté pour calculer d est liée à l'évaluation de $\phi(n) = (p-1)(q-1)$.

Celle-ci est aisée si on connait la valeur de p et de q. Sinon, le problème se ramène à factoriser l'entier n en produits de nombres premiers (cf la section précédente §3). Cependant, malgré tous les travaux menés sur le problème de la factorisation d'entiers depuis 300 ans, aucun algorithme rapide n'a été publié (donc a priori découvert) à ce jour[5]. C'est sur cette constatation qu'est fondée la confiance portée au sytème cryptographique RSA, exposé au §3.

2.2 Un autre exemple de FSU basée sur la difficulté du logarithme discret

La fonction exponentielle dans un groupe cyclique (i.e. $exp_G() = g^x$ dans G) est très utilisée en cryptographie pour la construction de fonctions à sens unique. En effet, son inverse, le logarithme discret (Discrete Logarithm Problem ou DLP en anglais) est un problème considéré extrêmement coûteux à résoudre pour de très nombreuses instances. On poura citer par exemple le protocole d'échange de clé de Diffie-Hellman (§**??**), l'algorithme de chiffrement El-Gamal (§4) ou encore le standard de signature numérique DSS (§5.5).

Cette section définit le logarithme discret, décrit le problème associé. Les paragraphes suivant décrivent différents algorithmes proposés pour le résoudre.

Considérons le cas particulier de la fonction puissance modulo un nombre premier p, qui est de type exponentielle dans un groupe. Si g est un générateur de $\mathbb{Z}_p^*$, cette fonction est définie par :

$$\begin{array}{rccc} F_g : & \mathbb{Z}_p^* & \longrightarrow & \mathbb{Z}_p^* \\ & x & \longrightarrow & g^x \mod p \end{array}$$

[5] Ainsi, et bien qu'en constante évolution, le plus grand nombre produit de deux nombres premiers factorisé en 2005 (au moment de la rédaction de ce chapitre) avait (seulement) 200 chiffres (voir `http://www.crypto-world.com/FactorRecords.html`). Ce record fut obtenu en mai 2005 par F. Bahr, M. Boehm, J. Franke et T.Kleinjung.

Etant donné x, il est facile de calculer $y = g^x \bmod p$ (toujours par un algorithme de type exponentiation rapide,voir §1.3.8).

Par contre, on ne connaît pas à ce jour de méthode rapide pour calculer $x = \log_g y \bmod p$, qui est appelé *logarithme discret* de y. Cette constatation est la base du protocole d'échange de clé de Diffie-Hellman (§**??**, page **??**), et de l'algorithme de chiffrement ElGamal (§**??**, page **??**).

3 Le système cryptographique à clé publique RSA

Ce sytème, proposé par Rivest, Shamir et Adleman en 1978 est aujourd'hui le plus connu et le plus utilisé de part sa simplicité.

3.1 Description de RSA.

Dans le système RSA un utilisateur crée son couple (clé plublique, clé privée) en utilisant la procédure suivante :

1. Choisir au hasard deux grands nombres premiers p et q. Il faut que p et q contiennent au moins 100 chiffres décimaux chacun.
2. Calculer $n = pq$
3. Choisir un petit entier e qui est premier avec $\phi(n) = (p-1)(q-1)$
4. Calculer d, l'inverse par la multiplication de e modulo $\phi(n)$.
5. Publier la paire $K_e = (e, n)$ comme sa clé publique RSA.
6. Garder secrète la paire $K_d = (d, n)$ qui est sa clé privée RSA.

On a alors :

Chiffrement RSA : $E_{K_e}(M) = M^e \ mod \ n$

Déchiffrement RSA : $D_{K_d}(\tilde{M}) = \tilde{M}^d \ mod \ n$

Exemple. Prenons $p = 47$ et $q = 59$ (Ces valeurs sont faibles et ne correspondent évidemment pas à des clés réelles).

- on calcule $n = p * q = 47 * 59 = 2773$
- on choisit e, premier par rapport à $\phi(n)$. Prenons par exemple $e = 17$.
- On calcule alors, par l'algorithme d'Euclide étendu[6], d tel que $d.e = 1 \bmod (p-1).(q-1)$, soit $d = 157$.

On a alors un couple clef publique (17 , 2773) clef privée (157 , 2773).
Pour chiffrer B c'est la valeur $01000010 = 66$ on calculera donc $(66^{17}) \pmod{2773}$ c'est-à-dire 872. Pour retrouver le message d'origine on calculera $(1553^{157}) \pmod{872}$ qui donne bien 66.

[6]sous Maple par exemple, cet algorithme correspond à la commande **igcdex**

3.2 Efficacité et robustesse de RSA.

Grâce aux algorithmes vus dans la section précédente :

- Il est facile de générer des grands nombres premiers, tout au moins en acceptant un taux d'erreur (cf test de Miller-Rabin §**??**, page **??**). Dans le cas de RSA, l'erreur n'est pas trop grave : en effet, si l'on commet une erreur en croyant que p et q sont premiers, le destinataire se rendra rapidement compte que les nombres ne sont pas premiers : soit la clef d n'est pas inversible, soit certains blocs du message décrypté sont incompréhensibles. Dans ce cas, on peut procéder à un changement de système RSA (recalcul de p et q).

- le calcul du couple (e, d) est extrèmement facile : il suffit d'appliquer l'algorithme d'Euclide étendu;

- enfin chiffrement et déchiffrement sont réalisés par exponentiation modulaire. Nous avons vu que cette exponentiation pouvait être réalisée assez efficacement.

La sécurité fournie par RSA repose essentiellement sur la difficulté à factoriser de grands entiers. En effet, si un attaquant peut factoriser le nombre $n = pq$ de la clef publique, il peut alors déduire directement $\phi(n) = (p-1)(q-1)$ et donc calculer la clé privée à partir de la clé publique par l'algorithme d'Euclide étendu. Donc, si l'on dispose d'un algorithme rapide pour factoriser de grands entiers, casser RSA devient facile aussi.

Après 20 ans de recherche, aucun moyen plus efficace que la factorisation de n n'a été publié pour casser RSA. Cependant, la réciproque : "si factoriser de grands entiers est dur alors casser RSA est dur" n'a pas été prouvée.
On peut cependant remarquer que si l'on choisit une petite clef publique e (par exemple $e \leq \log n$) alors casser RSA permet de factoriser n en temps polynomial.

En effet, supposons que l'on ait cassé RSA; on connaît donc la clef secrète d. Comme $ed - 1 \equiv 0 \mod (p-1)(q-1)$, on a :

$$\exists \lambda \in \mathbb{N} \ : \lambda.(pq - p - q + 1) = ed - 1$$

Or $pq = n$ et on peut suposser p et q différents de 2 et 3; ainsi p et q sont inférieurs à $\frac{n}{4}$, d'où $pq - p - q + 1 \geq \frac{n}{2}$. Finalement λ vérifie :

$$\lambda \leq \frac{2e.d}{n}.$$

De plus $d < n$ donc $\lambda < 2e$. Comme e est supposé petit, on peut donc essayer de déterminer p et q en essayant de manière exhaustive toutes les valeurs possibles pour $\lambda = 1 \ldots 2e - 1$. Plus précisément, posons

$$S_\lambda = n + 1 - \frac{ed + 1}{\lambda}.$$

Si λ est la bonne valeur on a $S_\lambda = p + q$. Comme le produit $p.q = n$ est connu, on en déduit que p et q sont alors les deux racines entières de l'équation $X^2 - S_\lambda X + n = 0$.

D'où l'algorithme pour calculer la factorisation $p.q$ de n :

Pour $\lambda = 1, 2, ..., 2e$ **faire** $S_\lambda := n + 1 - \frac{ed+1}{\lambda}$

Si S_λ **est entier alors** calculer les 2 racines p et q de l'équation : $X^2 - S_\lambda + n = 0$.

Si p **et** q **sont entiers, alors** $n = p.q$ est une factorisation non triviale de n.

Donc, si e est petit, il est plus facile de factoriser n (un problème réputé difficile si p et q sont très grands) que de casser RSA. C'est sur ce point que réside la confiance portée à RSA.

4 Le système cryptographique à clé publique d'ElGamal

Le problème du logarithme discret (Discret Logarithm Problem (DLP) en anglais) et la fonction à sens unique associée, abordés aux §2.2 et §2.2) sont utilisés dans le système cryptographique à clé publique ElGamal.

4.1 Description de ElGamal dans $\mathbb{Z}_p^*$

- Soit p un nombre premier tel que le problème du logarithme discret est difficile dans $\mathbb{Z}_p^*$
- Soit $g \in \mathbb{Z}_p^*$ un élément primitif.
- Soit s un nombre et $\beta = g^s$
- La clé publique est alors le triplet $K_e = (p, g, \beta)$.
- La clé secrète est le nombre $K_d = s$.

Chiffrement : Soit M le texte clair à chiffrer et $k \in \mathbb{Z}_{p-1}$ un nombre aléatoire secret.

$$E_{K_e}(M) = (y_1, y_2) \text{ avec } \begin{cases} y_1 = g^k \bmod p \\ y_2 = M.\beta^k \bmod p \end{cases}$$

Déchiffrement : Pour $y_1, y_2 \in \mathbb{Z}_p^*$, on définit :

$$D_{K_d}(y_1, y_2) = y_2.(y_1^s)^{-1}$$

En effet, $y_2.(y_1^s)^{-1} = M.\beta^k.(g^{k.s})^{-1} = M.g^{s.k}.(g^{k.s})^{-1} = M$

4.2 Généralisation de ElGamal

Bien que décrit dans le groupe multiplicatif $\mathbb{Z}_p$ et de la même façon que DLP, le chiffrement ElGamal peut se généraliser à d'autres classes de groupe. C'est d'ailleurs ce qui en fait sa force: de nombreux travaux portent actuellement sur la définition de nouveaux groupes dans lesquel le calcul du logarithme discret est difficile dans l'optique de les utiliser dans le cadre du protocole d'échange de clés de Diffie-Hellman ou encore du système de chiffrement ElGamal. C'est la cas par exemple du groupe additif d'une courbe elliptique sur un corps fini $\mathbb{Z}_p$.

5 Fonctions de Hachage et signatures électroniques

5.1 Notion de fonction de hachage

Définition 5.1 (Fonction de Hachage). *Une fonction de hachage H est une application facilement calculable qui transforme une chaîne binaire de taille quelconque t en une chaine binaire de taille fixe n, appelée* empreinte de hachage.

En général, $t > n$ si bien que cette fonction est surjective et on parle alors de collision entre des entrées x et x' lorsque

$$\begin{cases} x \neq x' \\ H(x) = H(x') \end{cases}$$

Remarque : si y est tel que $y = H(x)$, alors x est appelé la *préimage* de y;

Les propriétés de base d'une fonction de hachage sont la compression et la facilité de calcul. Elle peut également bénéficier des propriétés additionnelles suivantes:

- *résistance à la préimage* : étant donné y, il est calculatoirement difficile de trouver un x tel que $y = H(x)$;
- *résistance à la seconde préimage* : étant donné x, il est calculatoirement difficile de trouver $x' \neq x$ tel que $H(x) = H(x')$;
- *résistance à la collision* : il est calculatoirement difficile de trouver x et x' tels que $H(x) = H(x')$;

Définition 5.2 (Fonction de Hachage à Sens Unique). *Une fonction de hachage à sens unique*[7] *est une fonction de hachage qui vérifie les propriétés additionnelles de résistance à la préimage et à la seconde préimage.*

Définition 5.3 (Fonction de Hachage résistante aux collisions). *Une fonction de hachage résistante aux collisions*[8] *est une fonction de hachage qui vérifie les propriétés additionnelles de résistance à la seconde préimage et à la collision.*

[7] One-Way Hash Function
[8] Collision Resistant Hash Function

Les fonctions de hachage fournissent l'un des objets les plus important pour la cryptographie moderne. Elles sont principalement utilisées dans le cadre des signatures électroniques et pour les tests d'intégrités de données dans la mesure où elles permettent d'obtenir facilement le "résumé" d'une source binaire, utilisées comme une identification "unique" de cette source.

Nous verrons en détail au §5.2 l'utilisation des fonctions de hachage dans le cadre des signatures électronique.
Concernant le test d'intégrité, l'idée est de joindre au message M que l'on souhaite transmettre l'empreinte $H(M)$. Ainsi, lorsque le destinataire recoit un message $\tilde{M}$, éventuellement altéré au cours du transport, il lui suffit de calculer l'empreinte $H(\tilde{M})$ du message reçu avec celle qui était jointe. Si elles diffèrent, le message a été altéré.

5.1.1 Classification fonctionnelle

Les fonctions de hachage sont utilisées dans deux cas:

1. Les codes de détection de modifications[9] (MDC)qui utilisent des fonctions de hachage "*sans clé*"[10] (publiques) et qui permettent de vérifier l'intégrite d'une chaîne binaire transmise sur un canal.

2. Les codes d'authentification de messages[11] (MAC)qui utilisent des fonctions de hachage "*avec clé*"[12] et qui permettent non seulement de vérifier l'intégrite d'informations binaires transmises sur un réseau mais également d'authentifier la source d'une donnée.

5.1.2 Construction d'une fonction de hachage

Les fonctions de hachage cryptographiques les plus connues sont MD5[51], SHA-1[46], SHA-256, RIPE-MD[22].

Ces fonctions sont en effet relativement résistantes aux collisions (même si des collisions ont été trouvées sur MD5 et SHA-1) et aléatoires dans le sens où leur résultat est relativement non prédictible avec une apparente indépendance entre les entrées et les sorties.

De façon générale, ces fonctions sont construites de la façon suivante:

- une fonction de compression h (voir fig 1) est d'abord définie (sa spécification est plus ou moins compliquée selon le type de fonction de hachage que l'on souhaite obtenir). Une telle fonction applique deux entrées (de tailles respectives n et b bits) en une sortie de n bits.

- Pour calculer l'empreinte d'un message M, on effectue d'abord un *padding* qui complète ce message de sorte que sa taille soit un multiple de b. La

[9] Modification Detection Code
[10] Unkeyed Hash Functions
[11] Message Authentification Code
[12] Keyed Hash Function

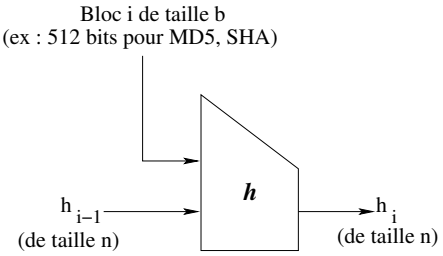

Figure 1: Fonction de compression d'une fonction de hachage

chaîne ainsi obtenu est découpées en blocs successifs de b bits:

$$M = M_1 M_2 \dots M_{k-1} M_k \text{ avec } |M_i| = b \forall i \in [1, k]$$

On itère alors la fonction h comme décrit dans la figure 2.

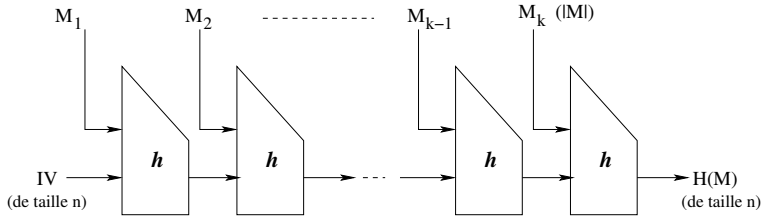

Figure 2: Itération de la fonction de compression pour le calcul d'empreinte

IV (Initial Value) est une chaîne (de taille n) fixée mais publique. Le théorème de Merkle-Damgård nous assure que si h est résistante aux collisions, il en est de même pour H.

Dans le cadre des MAC, IV est une clef secrète K de sorte que la fonction de hachage permet non seulement de vérifier l'intégrité de la source de données mais également d'authentifier son expéditeur (seul celui qui possède la clé K a pu calculer l'empreinte correcte à joindre au message).

5.2 Signatures Numériques

Les signatures manuscrites ont été longtemps utilisées pour prouver l'identité de leur auteur ou du moins l'accord du signataire avec le contenu du document. Avec des documents numériques, les objectifs d'une signature sont les suivants :

- Une signature est authentique. Elle convaint le destinataire que le signataire a délibérément signé le document.
- Une signature ne peut être falsifiée (imitée). Elle est la preuve que le signataire a délibérement signé le document.

- Une signature n'est pas réutilisable. Elle fait partie du document et une personne mal intentionnée ne peut pas déplacer la signature sur un autre document.
- Un document signé est inaltérable. Une fois le document signé, il ne peut plus être modifié.
- Une signature ne peut pas être reniée. La signature et le document sont des objets physiques et le signataire ne peut prétendre plus tard ne pas avoir signer le document.

Bien que des solutions à base de cryptosystèmes à cléf secrète et d'arbitre existent (voir [55] page 38), on utilise en général les cryptosystèmes à clé publique et une fonction de hachage à sens unique. Le principe général de la signature d'un document en combinant ces deux objets est illustré dans la figure 3.

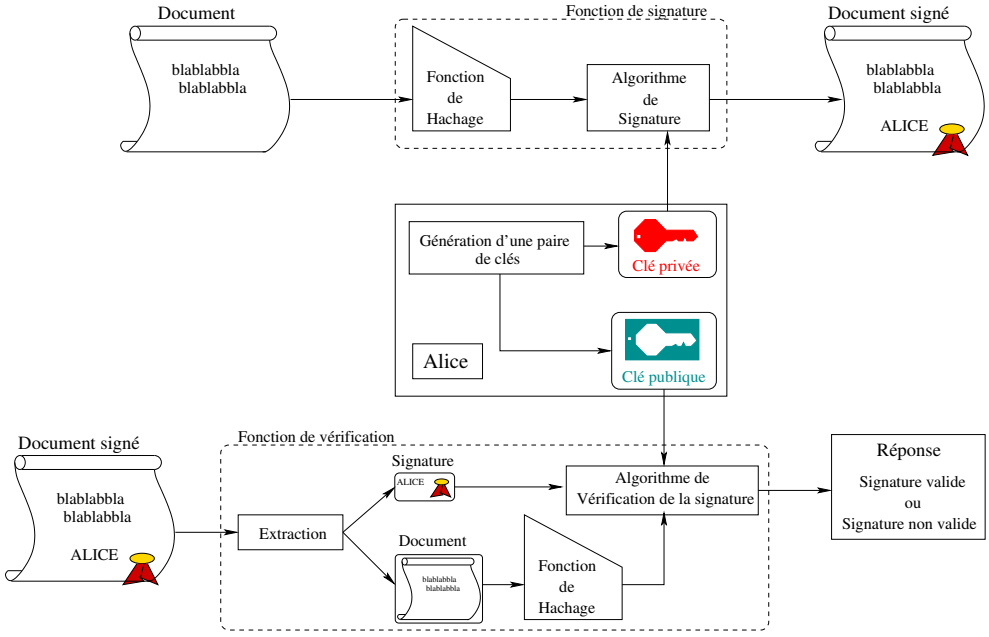

Figure 3: Principe de la signature et de sa vérification à partir d'un cryptosystèmes à clé publique et d'une fonction de hachage à sens unique

On voit d'abord qu'en pratique, ce n'est pas le document à transmettre qui est directement signé mais son empreinte. L'intérêt de cette approche tient à l'efficacité des algorithmes de chiffrement à clé publiques qui restent globalement lente. Un temps précieux serait donc perdu dans la signature d'un document complet, potentiellement de très grande taille. A l'inverse, la signature d'une empreinte garantit une entrée de petite taille (fixe) et donc un temps de calcul de la signature considérablement réduit.

Ensuite, c'est la clé privée d'Alice qui est utilisée pour générer la signature d'un document (à partir de son empreinte). On garantit ainsi que seule Alice (qui possède la clé privée) a pu signer le document. Réciproquement, c'est sa clé publique, accessible au monde entier (en particulier grâce à l'utisation d'architectures PKI - voir chapitre **??**) qui permet à tout le monde de vérifier la validité de la signature.

A ce stade, la spécification des algorithmes de signature et de vérification mentionnés dans la figure 3 n'a pas encore été donnée. En fait,on en dénombre principalement trois à l'heure actuelle :

- le premier est basé sur RSA et sera abordé au §5.3.
- le second est basé sur El Gamal et sera abordé au §5.4.
- le traisième est également basé sur DLP et fait l'objet du standard DSS[13][47] implémentant l'algorithme DSA[14]. Ce standard sera abordé au §5.5.

5.3 Signatures RSA

Génération des paramètres

Le procédé de génération des paramètres est identique à celui utilisé pour l'algorithme de cryptage RSA : Alice choisit deux nombres premiers au hasard p et q, et un exposant e tel que $3 \leq e < (p-1)(q-1)$ et $\text{pgcd}(e,(p-1)(q-1)) = 1$. Alice calcule $n = pq$ et d tel que $1 < d < (p-1)(q-1)$ et $ed \equiv 1 \bmod (p-1)(q-1)$. La clef publique de Alice est (n,e), et sa clef secrète est d.

Génération d'une signature

Un message à signer[15] m est ici un élément de l'ensemble $\{0,\ldots,n-1\}$. La signature du document m par Alice est simplement

$$s = m^d \mod n.$$

Vérification d'une signature

Bob reçoit de Alice le message m et sa signature s. Il récupère la clef publique (n,e) de Alice, et vérifie l'identité :

$$m = s^e \mod n.$$

La sécurité ici est donc celle du cryptosystème RSA. Ceci dit, la présentation faite ici est volontairement simplifiée par souci de clarté, et en l'état sujette à des attaques. Nous ne les détaillons toutefois pas ici.

[13] Digital Signature Standard

[14] Digital Signature Algorithm

[15] Si l'on souhaite signer des documents de taille plus importante, on signera en pratique le résultat par une fonction de hachage du message m. La construction des fonctions de hachage n'est pas détaillée ici. Disons simplement qu'il s'agit de fonctions prenant en entrée des fichiers de taille arbitraire et produisant un sortie un fichier de 128 ou 160 bits. Ces fonctions sont très véloces, et doivent éviter les collisions en pratique : très schématiquement, si h désigne une telle fonction, il doit être impossible en pratique de produire deux documents distincts m_1 et m_2 tels que $h(m_1) = h(m_2)$. Nous omettons ici tous les détails.

5.4 Signatures El Gamal

Tout comme l'on dérive des algorithmes de signatures de protocoles basés sur la factorisation, on peut en construire à partir de protocoles à clef publique basés sur le problème du logarithme discret. Nous décrivons ici le protocole de signature électronique de El Gamal dans le cas où le groupe sous-jacent est ${\mathbb{F}_p}^*$.

Génération des paramètres

Alice choisit un nombre premier p et g une racine primitive modulo p. Alice choisit au hasard un entier $a \in \{1, \ldots, p-2\}$, et calcule $A = g^a \mod p$. La clef pubblique de Alice est (p, g, A) et la clef secrète est a.

Génération d'une signature

Alice signe un texte représenté par un élément m de $\{1, \ldots, p-2\}$. Alice choisit un entier au hasard $k \in \{1, \ldots, p-2\}$, tel que $\text{pgcd}(k, p-1) = 1$, si bien que k^{-1} désigne l'inverse de k modulo $p-1$. Alice calcule

$$r = g^k \mod p,\ s = k^{-1}(m - ar) \mod (p-1).$$

La signature du document m est le couple (r, s).

Vérification d'une signature

Bob reçoit de Alice le message m et sa signature (r, s). Il récupère la clef publique (p, g, A) de Alice, et vérifie la congruence :

$$A^r r^s \equiv g^m \mod p.$$

En effet, si la signature est obtenue comme décrit plus haut, on doit avoir

$$A^r r^s \equiv g^{ar} g^{kk^{-1}(m-ar)} \equiv g^m \mod p.$$

Si l'on sait résoudre le problème du logarithme discret dans $\mathbb{F}_p^*$, alors on peut calculer la clef secrète a de Alice à partir de sa clef publique, et donc impersonaliser Alice, et falsifier des signatures. Pour des raisons de sécurité, il est nécessaire de préter attention au choix des paramètres. Nous n'allons toutefois pas plus loin ici.

5.5 Le standard DSA

DSA [47] désigne le Digital Signature Algorithm, et est un standard du NIST. Il s'agit en fait d'une variante du protocole de El Gamal, tenant compte des différentes attaques existantes.

Génération des paramètres

Alice génère un nombre premier q de 160 bits :

$$2^{159} < q < 2^{160}.$$

Alice choisit ensuite un nombre premier p faisant entre 512 et 1024 bits, vérifiant les conditions suivantes :

- $2^{511+64t} < p < 2^{512+64t}$ pour un entier t tel que $0 \leq t \leq 8$,

- le nombre premier q divise $p-1$.

La seconde condition assure que $\mathbb{F}_p^*$ possède un sous-groupe d'ordre q. Soit $\tilde{g}$ une racine primitive modulo p, et notons

$$g = \tilde{g}^{\frac{p-1}{q}} \mod p$$

un générateur du sous-groupe de $\mathbb{F}_p^*$ d'ordre q. Alice choisit un élément a au hasard dans l'ensemble $\{1, \ldots, q-1\}$, et calcule

$$A = g^a \mod p.$$

La clef publique de Alice est (p, q, g, A). Le problème du logarithme discret sous-jacent se passe dans le groupe d'ordre q.

Génération d'une signature

Alice souhaite signer le texte $m \in \{1, \ldots, q-1\}$. Alice choisit un entier $k \in \{1, \ldots, q-1\}$, et calcule

$$r = (g^k \mod p) \mod q,$$

et

$$s = k^{-1}(m + ar) \mod q,$$

où k^{-1} désigne l'inverse de k modulo q. La signature de m est alors le couple (r, s).

Vérification d'une signature

Bob veut vérifier la signature (r, s) du texte m envoyée par Alice. Il récupère tout d'abord la clef publique (p, q, g, A) de Alice. Il vérifie ensuite que les formats sont respectés, à savoir que

$$1 \leq r, s \leq q-1.$$

Si celà n'est pas le cas, la signature est considérée comme non-valide et rejettée. Si les formats sont respectés, Bob vérifie l'équation suivante :

$$r = ((g^{s^{-1}m \mod q} A^{rs^{-1} \mod q}) \mod p) \mod q.$$

References

[1] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. August 2002. preprint, http://www.cse.iitk.ac.in/users/manindra/primality.ps.

[2] A.V. Aho. Algorithms for finding patterns in strings. In J. van Leuwen, editor, *Algorithms and Complexity*, pages 253–300. Elsevier, 1990.

[3] Eric Bach. How to generate factored random numbers. *SIAM Journal on Computing*, 17(2):179–193, April 1988. Special issue on cryptography.

[4] Eric Bach. Explicit bounds for primality testing and related problems. *Mathematics of Computation*, 55(191):355–380, July 1990.

[5] Daniel V. Bailey and Christof Paar. Efficient arithmetic in finite field extensions with application in elliptic curve cryptography. *Journal of Cryptology*, 14(3):153–176, 2001.

[6] Michael Ben-Or. Probabilistic algorithms in finite fields. In *22th Annual Symposium on Foundations of Computer Science*, pages 394–398, Los Alamitos, California, USA, October 1981. IEEE Computer Society Press.

[7] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, November 1984.

[8] Dan Boneh and Ramarathnam Venkatesan. Breaking RSA may not be equivalent to factoring. In *Advances in Cryptology – EUROCRYPT '98*, volume 1403, pages 59–71, 1998. http://theory.stanford.edu/ dabo/papers/no_rsa_red.ps.gz.

[9] Joe P. Buhler, Hendrik W. Lenstra, and Carl Pomerance. *The development of the number field sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer-Verlag, 1993.

[10] Joe P. Buhler, Hendrik W. Lenstra, and Carl Pomerance. *Factoring integers with the number field sieve*, pages 50–94. Volume 1554 of *Lecture Notes in Mathematics* [9], 1993.

[11] Stefania Cavallar. Strategies in filtering in the number field sieve. Technical report, Centrum voor Wiskunde en Informatica, May 2000. http://www.cwi.nl/ftp/CWIreports/MAS/MAS-R0012.ps.Z.

[12] Stefania Cavallar, Bruce Dodson, Arjen K. Lenstra, Walter Lioen, Peter L. Montgomery, Brian Murphy, Herman te Riele, Karen Aardal, Jeff Gilchrist, Gérard Guillerm, Paul Leyland, Joël Marchand, François Morain, Alec Muffett, Chris Putnam, Craig Putnam, and Paul Zimmermann. Factorization of a 512-bit rsa modulus. In Bart Preneel, editor, *Advances in cryptology: EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14–18, 2000: proceedings*, volume 1807 of *Lecture Notes in Computer Science*, pages xiii + 608, New York, NY, USA, 2000. Springer-Verlag Inc.

[13] Henri Cohen. *A course in computational algebraic number theory*, volume 138 of *Graduate Texts in Mathematics*. Springer-Verlag, 1st edition, 1993. `http://www.cs.berkeley.edu/~christos/`.

[14] Don Coppersmith. Fast evaluation of logarithms in fields of characterstic two. *IEEE Transactions on Information Theory*, IT-30:587–594, 1984.

[15] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001. `http://www.mitpress.mit.edu/algorithms/`.

[16] Joan Daemen and Vincent Rijmen. AES Proposal: Rijndael. Technical report, 1998. `http://citeseer.ist.psu.edu/daemen98aes.html`.

[17] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer-Verlag, 1st edition, 2001. `http://www.iaik.tu-graz.ac.at/research/krypto/AES/`.

[18] J. H. Davenport. Primality testing revisited. In Paul S. Wang, editor, *Proceedings of ISSAC '92. International Symposium on Symbolic and Algebraic Computation*, pages 123–129, New York, NY 10036, USA, 1992. ACM Press.

[19] J. Chassin de Kergommeaux and P. Codognet. Parallel logic programming systems. *ACM Computing Surveys*, 26(3):295–336, Septembre 1994.

[20] Michel Demazure. *Cours d'algèbre. Primalité, Divisibilité, Codes*, volume XIII of *Nouvelle bibliothèque Mathématique*. Cassini, Paris, 1997.

[21] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[22] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. The hash function ripemd-160. version ellectronique uniquement, 1997. `http://www.esat.kuleuven.ac.be/~bosselae/ripemd160.html`.

[23] J.-G. Dumas, T. Gautier, and C. Pernet. FFLAS: Finite Field Linear Algebra Subroutines. In T. Mora, editor, *International Symposium on Symbolic and Algebraic Computation, ISSAC'02*, pages 63–74, Lille, July 2002. ACM Press.

[24] Pierre Dusart. *Autour de la fonction qui compte le nombre de nombres premiers*. PhD thesis, Université de Limoges, 1998.

[25] Pierre Dusart. The k^{th} prime is greater than $k(\ln k + \ln\ln k - 1)$ for $k \geq 2$. *Mathematics of Computation*, 68(225):411–415, January 1999.

[26] Niels Ferguson and Bruce Schneier. *Practical Cryptography*. Wiley Publishing, Inc, 1st edition, 2003. `http://www.macfergus.com/pc/`.

[27] M. R. Garey and Johnson D. S. *Computers and intractibility, a guide to the theory of NP-completeness*. W. H. Freeman and Co., San Francisco, 1979.

[28] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra.* Cambridge University Press, New York, NY, USA, 1999.

[29] M. Giesbrecht. *Nearly optimal algorithms for canonical matrix forms.* PhD thesis, University of Toronto, Department of Computer Science, Canada, 1993.

[30] Torbjörn Granlund. *The GNU multiple precision arithmetic library*, 2002. Version 4.1, `www.swox.com/gmp/manual`.

[31] Kevin Hammond. Parallel functionnal programming: an introduction. In Hoon Hong, editor, *First International Symposium on Parallel Symbolic Computation (PASCO'94)*, Lecture Notes Series in Computing, pages 181–194, Linz, Autriche, 1994.

[32] David Kahn. *The Code-Breakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet.* Scribner, 1996.

[33] Donald E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming.* Addison-Wesley, 3rd edition, 1997. `http://www-cs-faculty.stanford.edu/~knuth/taocp.html`.

[34] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming.* Addison-Wesley, 3rd edition, 1997. `http://www-cs-faculty.stanford.edu/~knuth/taocp.html`.

[35] Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming.* Addison-Wesley, 3rd edition, 1997. `http://www-cs-faculty.stanford.edu/~knuth/taocp.html`.

[36] Neal Koblitz. *Algebraic Aspects of Cryptography*, volume 3 of *Algorithms and Compuitation in Mathematics.* Springer-Verlag, 1998.

[37] Rudolf Lidl and Harald Niederreiter. *Introduction to Finite Fields and Their Applications.* Cambridge University Press, revised edition, 1994.

[38] Bruno Martin. *Codage, cryptologie et applications.* Presses polytechniques et universitaires romandes, 2004.

[39] James L. Massey. Lecture noes on applied digital information theory i, 1998. `http://www.isi.ee.ethz.ch/education/public/pdfs/aditI.pdf`.

[40] James L. Massey. Lecture noes on applied digital information theory ii, 1998. `http://www.isi.ee.ethz.ch/education/public/pdfs/aditI.pdf`.

[41] Jean-Pierre Massias and Guy Robin. Bornes effectives pour certaines fonctions concernant les nombres premiers. *Journal de Théorie des Nombres de Bordeaux*, 8:213–238, 1996.

[42] J. McGraw. SISAL: Streams and Iterations in a Sigle-Assignment Language – Reference Manual. Technical Report Manual M-146, Lawrence Livermore National Lab., 1985.

[43] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography.* Computer Sciences Applied Mathematics Engineering. CRC Press, Inc., 1st edition, 1996. `http://www.cacr.math.uwaterloo.ca/hac/`.

[44] Gary L. Miller. Riemann's hypothesis and tests for primality. In *Conference Record of Seventh Annual ACM Symposium on Theory of Computation*, pages 234–239, Albuquerque, New Mexico, May 1975.

[45] B. A. Murphy and R. P. Brent. On quadratic polynomials for the number field sieve. In *Proc. Fourth Australasian Theory Symposium (CATS'98, Perth, Feb. 1998), special issue of Australian Computer Science Communications*, volume 20, pages 199–213, 1998.

[46] NIST. FIPS-180-1 SECURE HASH STANDARD. Technical report, Federal Information Processing Standards Publication, 1993. `http://www.itl.nist.gov/fipspubs/fip180-1.htm`.

[47] NIST. FIPS-196 DIGITAL SIGNATURE STANDARD (DSS). Technical report, Federal Information Processing Standards Publication, 1994. `http://www.itl.nist.gov/fipspubs/fip186.htm`.

[48] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley Longman, 1st edition, 1994. `http://www.cs.berkeley.edu/~christos/`.

[49] John M. Pollard. A Monte Carlo method for factorization. 15(3):331–334, 1975.

[50] Michaël O. Rabin. A probabilistic algorithm for testing primality. *Journal of Number Theory*, 12, 1980.

[51] R. Rivest. RFC 1321 - The MD5 Message-Digest Algorithm. Technical report, MIT Laboratory for Computer Science and RSA Data Security, April 1992.

[52] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signature and public-key cryptosystems. *Communication of the ACM*, 21(2), 1978.

[53] Jean-Louis Roch, Jean-Guillaume Dumas, Eric Tannier, Sébastien Varrette, Yves Denneulin, and Grégory Mounié. Théorie des codes, 1997–2005.

[54] J. Barkley Rosser and Lowell Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois Journal of Mathematics*, 6:64–94, 1962.

[55] Bruce Schneier. *"Cryptographie Appliquée"*. Vuibert, Wiley and International Thomson Publishing, NY, 2nd edition, 1997. .

[56] A. Schonhage. Schnelle berechnung von kettenbruchtenwicklungen. *Acta Informatica*, 1:139–144, 1971.

[57] A. Schonhage and V. Strassen. Schnelle multiplikation grosser zahlen. *SIAM Journal on Computing*, 7:281–292, 1971.

[58] C.E. SHANNON. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 et 623–565, 1948. http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf.

[59] C.E. SHANNON. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):656–715, 1949. http://www.cs.ucla.edu/~jkong/research/security/shannon1949.pdf.

[60] Victor Shoup. Searching for primitive roots in finite fields. *Mathematics of Computation*, 58(197):369–380, January 1992.

[61] Victor Shoup. Fast construction of irreducible polynomials over finite fields. *Journal of Symbolic Computation*, 17(5):371–391, May 1994.

[62] Victor Shoup. NTL 5.3: A library for doing number theory, 2002. `www.shoup.net/ntl`.

[63] Robert M. Solovay and Volker Strassen. A fast Monte-Carlo test for primality. *SIAM Journal on Computing*, 6(1):84–85, March 1977.

[64] Douglas R. Stinson. *Cryptography: Theory and Practice.* Chapman & Hall/CRC Press, 2nd edition, 2002. `http://www.cacr.math.uwaterloo.ca/~dstinson/CTAP2/CTAP2.html`.

[65] J. van Leuwen. *Algorithms and Complexity.* Elsevier, 1990.

[66] Samuel S. Wagstaff, Jr. *Cryptanalysis of number theoretic ciphers.* Chapman-Hall / CRC, 2003.