# Introduction à la cryptographie

Jean Goubault-Larrecq
LSV, ENS Cachan

6 novembre 2014

**Résumé**

On présente quelques bases de cryptographie. Ceci est la version 3 du 06 novembre 2014. La version 2 datait du 09 octobre 2013. La version 1 datait du 07 novembre 2012.

## 1 Introduction

Le but de la cryptographie est d'assurer la sécurité des échanges de messages en présence d'un ou plusieurs attaquants, ou intrus, potentiels.

On notera typiquement $M$ le message en clair, $K$ une clé, $\{M\}_K$ le message $M$ chiffré par $K$, $I$ l'intrus.

On cherchera à assurer des *propriétés* de sécurité par des *moyens* algorithmiques :

| Propriété | Moyen | |
|---|---|---|
| Secret | chiffrement $\{M\}_K$ | $I$ ne peut pas reconstruire/rien déduire du clair $M$ |
| Authenticité | signature $[M]_K$ | $I$ ne peut pas se faire passer pour l'émetteur de $M$ |
| Intégrité | hachage $h(M)$, MAC $h(M,K)$ | $I$ ne peut pas modifier $M$ sans que ceci ne soit détecté |

Et il y a bien d'autres propriétés :

- distribution de clés (protocole de Diffie-Hellman)
- en e-commerce : non-duplication (factures), non-répudiation (commandes) ;
- en vote électronique : éligibilité (du votant), anonymat (du vote), absence de reçu, non-coercition, vérification individuelle (du vote par le votant), vérification des décomptes (des votes) ;
- preuves à connaissance nulle, secrets partagés, etc.

## 2 Les anciennes méthodes de chiffrement

$M = a_0 a_1 \dots a_{n-1}$ : mot sur $\Sigma$ (alphabet, ensemble fini de lettres) $= \{0, 1, \dots, k-1\}$.

- César ($\sim -50$) : $K \in \Sigma$, $\{M\}_K = (a_0 + K \bmod k)(a_1 + K \bmod k) \dots (a_{n-1} + K \bmod k)$.

– Vigenère ($\sim 1550$) : $K = b_0 b_1 \dots b_{m-1} \in \Sigma^m$, $\{M\}_K = (a_0 + b_0 \bmod k)(a_1 + b_1 \bmod k) \dots (a_{n-1} + b_{n-1 \bmod m} \bmod k)$.
– Masque jetable (one-time pad ; Vernam, 1926) : $K = b_0 b_1 \dots b_{n-1} \in \Sigma^m$, $\{M\}_K = (a_0 + b_0 \bmod k)(a_1 + b_1 \bmod k) \dots (a_{n-1} + b_{n-1} \bmod k)$.

Différence entre Vigenère et masque jetable : $m$ fixe dans le premier, $m = n$ dans le second.

César facile à casser (AlKindi, $\sim 850$) : les lettres les plus fréquentes en français sont E > A > I > S, T, etc. (Analyse *statistique*). De plus, il y a peu de clés.

Vigenère presque aussi facile à casser : revient à $m$ chiffrements de César indépendants (si l'on connaît $m$) ; on peut retrouver $m$ avec forte probabilité (Babbage $1854$, Kasiski $1863$) en calculant le pgcd des distances entre sous-mots identiques dans le chiffré.

A l'opposé, le masque jetable est *inconditionnellement sûr* (à condition de tirer la clé *au hasard*, uniformément : ne pas partager la clé avec d'autres, de ne pas mégoter sur la longueur de la clé, ne pas réutiliser la clé). En résumé, aucune analyse statistique ne permet de retrouver le clair :

**Théorème 2.1** *Pour le chiffrement par masque jetable, la distribution des clairs $M$, même connaissant le chiffré $\{M\}_K$ est la* distribution uniforme *:*

$$Pr_K[\{M\}_K = M' \mid M'] = \frac{1}{|\Sigma|^{|M|}}.$$

*Démonstration.* À $M'$ fixé, il y a bijection entre $K$ et les $M$ tels que $\{M\}_K = M'$. □ Attention, ça demande à retirer la clé au hasard à chaque chiffrement. D'où les questions suivantes :

– Comment distribue-t-on la clé aux deux acteurs qui veulent communiquer ? Une solution : Diffie-Hellman, voir section 7.
– Comment tire-t-on une clé au hasard ? Voir section 8.

# 3 Théorie de l'information, entropie, et chiffrement

Shannon a fondé la théorie de l'information dans le but d'étudier les possibilités de compression de signaux, et de correction d'erreurs dans les transmissions de signaux [1]. Les signaux, et le bruit qui s'y rajoutent, sont vus comme des variables aléatoires. La notion fondamentale dans ce domaine est celle d'*entropie* d'une variable aléatoire.

**Définition 3.1** *Soit $X$ une variable aléatoire (discrète) sur un ensemble fini $A$, et posons $p_a = Pr[X = a]$ (la* loi *de $X$). L'*entropie *de $X$ est :*

$$H(X) = -\sum_{a \in A} p_a \log p_a.$$

Le logarithme est pris *en base* 2 : $\log 2^n = n$. (On notera $\ln x$ le logarithme népérien de $x$.)

## 3.1 Compression

L'intérêt de la notion d'entropie est sans doute le plus clair en matière d'algorithmes de compression (ce que Shannon appelle un canal non bruité, ou un *code*) : pour tout $M \in \Sigma^*$, $code(M) \in \{0,1\}^*$, et la fonction $code$ est inversible (codage *sans perte*). On souhaite que $code(M)$ soit le plus court possible en moyenne.

Un *bit* est un chiffre binaire, 0 ou 1.

**Exemple 3.2** *Un code idiot : $code(M) = M$, capacité $\log|\Sigma|$. Intuitivement, il faut $\log|\Sigma|$ bits pour écrire chaque lettre.*

**Exemple 3.3** *Codage de Huffman. On fabrique un arbre binaire. Aux feuilles, on trouve les lettres $a \in \Sigma$. On s'arrange pour que les lettres les plus fréquentes soient aux feuilles les moins profondes dans l'arbre comme suit. Algorithme : initialement $S = \{(\{a\}, Pr[X = a], T_a) \mid a \in \Sigma\}$, où $T_a$ est l'arbre réduit à une unique feuille étiquetée $a$ ; tant que $S$ contient au moins deux éléments, en prendre deux, $(E, p, T)$ et $(E', p', T')$ avec $p$, $p'$ les plus bas possibles, et les remplacer par $(E \cup E', p + p', \overset{T}{\diagup}\!\!\overset{T'}{\diagdown})$ ; à la fin, $S = \{(\Sigma, 1, T)\}$, et retourner $T$.*

*On pose $code(a) =$ chemin depuis la racine de $T$ jusqu'à la feuille $a$, où « droite »=1, « gauche »=0 ; $code(a_0 a_1 \dots a_{m-1}) = code(a_0)code(a_1)\dots code(a_{m-1})$.*

**Définition 3.4** *Un* code préfixe *est une fonction $code\colon \Sigma \to \{0,1\}^*$ tel que pour tous $a, b \in \Sigma$ avec $a \neq b$, $code(a)$ n'est pas un préfixe de $code(b)$ (et réciproquement).*

Le code de Huffman est préfixe. Un code est préfixe si et seulement si on peut organiser les chaînes $code(a)$, $a \in \Sigma$ sous forme d'un arbre binaire, et aux feuilles les lettres $a$ (ou bien rien). Tout code préfixe est décodable de façon unique : lire les bits dans l'arbre depuis la racine jusqu'à arriver à une feuille, émettre la lettre écrite à cette feuille (si pas de lettre, erreur), et repartir à la racine. (Le code Morse fonctionne déjà comme ça.)

**Lemme 3.5 (Locatelli-Mabon)** *Soit $X$ de loi $p_a = Pr[X = a]$, $a \in \Sigma$, et soit $q_a \geq 0$, $a \in \Sigma$, tels que $\sum_{a\in\Sigma} q_a \leq 1$. On a l'*inégalité de Locatelli-Mabon *:*

$$H(X) \leq -\sum_{a\in\Sigma} p_a \log q_a,$$

*avec égalité si et seulement si $p_a = q_a$ pour tout $a \in \Sigma$.*

On utilise comme convention $\log 0 = -\infty$, $0.(-\infty) = 0$.

*Démonstration.* Si certains $p_a$ sont nuls, on remplace $\Sigma$ par $\{a \in \Sigma \mid p_a \neq 0\}$. Ceci nous permet de supposer $p_a \neq 0$ pour tout $a \in \Sigma$. On a maintenant :

$$\begin{aligned} H(X) + \sum_{a\in\Sigma} p_a \log q_a &= \sum_{a\in\Sigma} p_a \log\left(\frac{q_a}{p_a}\right) = \sum_{a\in\Sigma} p_a \ln\left(\frac{q_a}{p_a}\right)\frac{1}{\ln 2} \\ &\leq \sum_{a\in\Sigma} p_a \left(\frac{q_a}{p_a} - 1\right)\frac{1}{\ln 2} \\ &= \left(\sum_{a\in\Sigma} q_a - \sum_{a\in\Sigma} p_a\right)\frac{1}{\ln 2} \leq 0. \end{aligned}$$

Si $p_a \neq q_a$ pour au moins un $a \in \Sigma$, alors on peut supposer que c'est pour un $a$ tel que $p_a \neq 0$. En effet, sinon $p_a = q_a$ pour tout $a$ tel que $p_a = 0$, et la somme de ces $q_a$ (i.e., des $p_a$) vaut 1, ce qui implique que $q_a = 0 = p_a$ pour tout $a$ tel que $p_a = 0$.

Donc, si $p_a \neq q_a$ pour au moins un $a$, avec $p_a \neq 0$, alors le terme $\ln(q_a/p_a)$ est strictement supérieur à $q_a/p_a - 1$, et l'inégalité est stricte. □

**Lemme 3.6 (Kraft)** *Soit code un code préfixe sur $\Sigma$. On a l'*inégalité de Kraft *:*

$$\sum_{a \in \Sigma} 2^{-|code(a)|} \leq 1.$$

*Démonstration.* Dessinons l'arbre binaire du code préfixe. Définissons le poids de chaque nœud (qui est un mot binaire $w$) par le nombre $2^{-|w|}$. Le poids de tout nœud interne $w$ est la somme des poids de ses deux fils $w0$ et $w1$. Par récurrence sur la hauteur de l'arbre, le poids de la racine, $2^0 = 1$, est la somme des poids des feuilles. Or $\sum_{a \in \Sigma} 2^{-|code(a)|}$ est la somme des poids de certaines des feuilles, celles qui sont étiquetées par une lettre. □

Notons $|w|$ le nombre de bits dans le mot binaire $w$. On ne peut pas compresser plus qu'avec $H(X)$ bits par lettre :

**Lemme 3.7** *Pour tout code préfixe code sur $\Sigma$, $E(|code(X)|) \geq H(X)$.*

*Démonstration.* On applique l'inégalité de Kraft (Lemme 3.6), et on obtient $q_a = 2^{-|code(a)|}$ qui satisfait les hypothèses de Locatelli-Mabon (Lemme 3.5). Donc $H(X) \leq -\sum_{a \in \Sigma} p_a \log 2^{-|code(a)|} = E(|code(X)|)$. □

La borne du Lemme 3.7 est optimale (à très peu de choses près).

**Lemme 3.8** *Soit $X$ une variable aléatoire de loi $(p_a)_{a \in \Sigma}$, avec $p_a \neq 0$ pour tout $a \in \Sigma$. Il existe un code préfixe (dit* de Shannon*) sur $\Sigma$ tel que $|code(a)| = \lceil -\log p_a \rceil$ pour tout $a \in \Sigma$.*

*Démonstration.* Posons $\lambda_a = \lceil -\log p_a \rceil$.

Trions les lettres $a_1, a_2, \ldots, a_n$ de $\Sigma$ de sorte que $p_{a_1} \geq p_{a_2} \geq \ldots \geq p_{a_n}$. Posons $P_{a_i} = p_{a_1} + p_{a_2} + \ldots + p_{a_i - 1}$, et écrivons ce nombre en base 2 : $0, b_1 b_2 \ldots b_k \ldots$, écriture que l'on tronque après $\lambda_{a_i}$ bits. On pose $code(a_i) =$ la suite des $\lambda_{a_i}$ premiers bits, $b_1 b_2 \ldots b_{\lambda_{a_i}}$.

Pour tout $j > i$, $P_{a_j} \geq P_{a_i} + 1/2^{\lambda_{a_i}}$, donc $P_{a_i}$ et $P_{a_j}$ diffèrent sur au moins un de leurs $\lambda_{a_i}$ premiers bits. Or $code(a_i)$ est de longueur $\lambda_{a_i}$, et $code(a_j)$ est de longueur $\geq \lambda_{a_i}$, donc $code(a_i)$ et $code(a_j)$ diffèrent sur au moins un bit : aucun ne peut être préfixe de l'autre. □

En résumé :

**Théorème 3.9 (Shannon)** *Soit $X$ une variable aléatoire sur $\Sigma$ :*

1. *la limite de compression d'un code préfixe est l'entropie : pour tout code préfixe code, $E(|code(X)|) \geq H(X)$ ;*
2. *cette limite est atteinte, à un bit près : le code de Shannon vérifie $E(|code(X)|) < H(X) + 1$.*

*Démonstration.* Pour le code de Shannon, $E(|code(X)|) = \sum_{a\in\Sigma} p_a\lceil -\log p_a\rceil < \sum_{a\in\Sigma} p_a(-\log p_a + 1) = H(X) + 1$. □

**Remarque 3.10** *On peut démontrer que le code de Huffman est optimal : la longueur moyenne du code de Huffman est inférieure ou égale à la longueur moyenne de n'importe quel code préfixe. On a donc $H(X) \leq E(|code(X)|) < H(X) + 1$ pour le code de Huffman.*

On identifie $H(X)$ à la *quantité d'information*, mesurée en bits, d'une lettre aléatoire $a$ tirée par $X$. Ceci représente l'*incertitude*, ou l'*ambiguïté* que l'on a quant à la connaissance d'une lettre aléatoire $a$.

▷ **Exercise 3.11**
Soit $\Sigma$ un alphabet, $b$ et $c$ deux lettres distinctes de $\Sigma$, et $\Sigma'$ l'alphabet formé des lettres restantes de $\Sigma$ plus une nouvelle lettre notée $\{b, c\}$. Soit $X'$ une variable aléatoire de loi $(p'_{a'})_{a'\in\Sigma'}$ sur $\Sigma'$, et $X$ la variable aléatoire sur $\Sigma$ de loi $(p_a)_{a\in\Sigma}$ avec $p_a = p'_a$ pour tout $a \neq b, c$, $p_b = \alpha p'_{\{b,c\}}$, $p_c = (1-\alpha)p'_{\{b,c\}}$. Alors :

$$H(X) = H(X') + p'_{\{b,c\}} H(B_\alpha),$$

où $B_\alpha$ est la variable aléatoire sur $\{0, 1\}$ qui choisit $1$ avec probabilité $\alpha$. Notons que $H(B_\alpha) = -[\alpha\log\alpha + (1-\alpha)\log(1-\alpha)]$.

## 3.2 Propriétés de l'entropie

**Lemme 3.12** *Soit $n = |\Sigma|$. Pour toute variable aléatoire $X$ sur $\Sigma$, $H(X) \leq \log n$, avec égalité si et seulement si $X$ est de loi uniforme sur $\Sigma$.*

*Démonstration.* $H(X) \leq \log n$ : par l'inégalité de Locatelli-Mabon avec $q_a = 1/n$. Le lemme 3.5 énonce aussi que l'égalité est stricte si $q_a \neq 1/n$ pour au moins un $a \in \Sigma$. □

**Lemme 3.13** *$H(X) \geq 0$, avec égalité si et seulement si la loi de $X$ donne la masse $1$ à un (unique) élément.*

*Démonstration.* $H(X) \geq 0$ évident, car $-p\log p \geq 0$ pour tout $p \in [0, 1]$ : fonction de dérivée $-\log p - 1$, de dérivée double $-1/p \leq 0$, la dérivée décroît donc de $+\infty$ à $-1$, s'annule pour $p = 1/2$, donc $-p\log p$ croît de $0$ à $1/2$ lorsque $p$ croît de $0$ à $1/2$, et décroît ensuite de $1/2$ à $0$. Au passage, on note que $-p\log p$ ne s'annule que pour $p = 0$ ou $p = 1$.

Si $H(X) = 0$, où $X$ est de loi $(p_a)_{a\in\Sigma}$, alors tous les $-p_a\log p_a$ sont nuls, donc ils valent tous $0$ ou $1$. Mais exactement un d'entre eux peut valoir $1$, les autres valent $0$. □

On s'intéressera à des lois *conjointes* sur deux lettres $a$ et $b$. Ceci revient à se donner deux variables aléatoires $X, Y$, et une loi conjointe $p_{ab} = Pr[X = a, Y = b]$. Les lois *marginales* sont : $p^1_a = \sum_b p_{ab}$ (loi de $X$), $p^2_b = \sum_a p_{ab}$ (loi de $Y$). Les variables $X$ et $Y$ sont *indépendantes* si et seulement si $p_{ab} = p^1_a p^2_b$ pour tous $a, b$.

**Définition 3.14** *On appelle* entropie conjointe *de deux variables aléatoires $X$, $Y$ la quantité :*

$$H(X,Y) = -\sum_{a,b} p_{ab} \log p_{ab}$$

*égale à l'entropie de la variable aléatoire $(X,Y)$.*

L'entropie conjointe est donc le nombre de bits dans lesquels on peut coder le couple $(X,Y)$, en exploitant le fait que la connaissance de $X$ donne des informations sur celle de $Y$, et réciproquement.

**Lemme 3.15** *On a $H(X,Y) \leq H(X) + H(Y)$, avec égalité si et seulement si $X$ et $Y$ sont indépendantes.*

*Démonstration.* Par Locatelli-Mabon avec $q_{ab} = p^1_a p^2_b$,

$$H(X,Y) \quad \leq \quad -\sum_{a,b} p_{ab} \log(p^1_a p^2_b) = H(X) + H(Y),$$

et l'inégalité est stricte à moins que $p_{ab} = p^1_a p^2_b$ pour tous $a, b$. □

▷ **Exercise 3.16**
On peut considérer $\Sigma^n$, l'ensemble des mots de longueur $n$ sur $\Sigma$, comme un alphabet aussi. Soit $X$ une variable aléatoire sur $\Sigma$, d'entropie $H(X)$. Quelle est l'entropie $H(X^n)$, $X^n$ étant la variable aléatoire des mots de longueurs $n$ où chaque lettre est tirée indépendamment selon la loi de $X$ ? On montrera que cette entropie est de la forme $an$ lorsque $n \to +\infty$, où $a$, le taux d'entropie par caractère, est une quantité familière.

▷ **Exercise 3.17**
Si $f$ est une fonction concave ($f(\lambda x + (1-\lambda)y) \geq \lambda f(x) + (1-\lambda) f(y)$ pour tout $\lambda \in [0,1]$), montrer que :

$$f(E(X)) \geq E(f(X)).$$

Si de plus $f$ est strictement concave ($f(\lambda x + (1-\lambda)y) > \lambda f(x) + (1-\lambda) f(y)$ pour tout $\lambda \in [0,1]$, $\lambda \neq 0, 1$), montrer l'inégalité est stricte sauf si la loi de $X$ donne la masse $1$ à un (unique) élément.

Si $X$, $Y$ sont deux variables aléatoires, et que $p(a \mid b)$ est la probabilité conditionnelle $Pr[X = a \mid Y = b]$, on peut définir l'entropie $H(X \mid Y = b) = -\sum_a p(a \mid b) \log p(a \mid b)$ de cette loi conditionnelle. On prend la moyenne sur $b$ :

**Définition 3.18** *Soient $X$ et $Y$ deux variables aléatoires, et $p(a \mid b) = Pr[X = a \mid Y = b]$, $p^2_b = Pr[X = b]$. L'*entropie conditionnelle *de $X$, conditionnelle selon $Y$, est :*

$$H(X \mid Y) = \sum_b p^2_b H(X \mid Y = b).$$

Ceci mesure la quantité d'information moyenne supplémentaire contenue dans $X$, connaissant $Y$.

**Lemme 3.19** *$H(X \mid Y) = H(X,Y) - H(Y)$. C'est une quantité : positive ou nulle, et nulle si et seulement si $X$ dépend de façon univoque de $Y$ (i.e., il existe une fonction $f$ telle que $X = f(Y)$) ; inférieure ou égale à $H(X)$, et égale à $H(X)$ si et seulement si $X$ et $Y$ sont indépendantes.*

*Démonstration.* On a $p(a \mid b)p_b^2 = p_{ab}$, où $(p_{ab})_{a,b}$ est la loi conjointe. Donc :

$$\begin{aligned} H(X \mid Y) + H(Y) &= -\sum_b p_b^2 \sum_a p(a \mid b) \log p(a \mid b) - \sum_b p_b^2 \log p_b^2 \\ &= -\sum_{a,b} p_{ab} \log p(a \mid b) - \sum_b p_b^2 \log p_b^2 \\ &= -\sum_{a,b} p_{ab} \log p(a \mid b) - \sum_{a,b} p_{ab} \log p_b^2 \\ &= -\sum_{a,b} p_{ab} \log(p(a \mid b)p_b^2) = -\sum_{a,b} p_{ab} \log p_{ab} = H(X,Y). \end{aligned}$$

On a $H(X \mid Y) \geq 0$, puisque $H(X \mid Y)$ est la moyenne des entropies $H(X \mid Y = b)$, qui sont positives ou nulles, par le lemme 3.13. De plus, $H(X \mid Y) = 0$ si et seulement si $H(X \mid Y = b) = 0$ pour tout $b$, si et seulement si pour tout $b$, $X$ donne la masse 1 à un unique élément $a$, dépendant de $b$. Posons $a = f(b)$, on a alors $X = f(Y)$.

Le reste est par le lemme 3.15. □

**Définition 3.20** *L'*information mutuelle *entre $X$ et $Y$ est $I(X;Y) = H(X) - H(X \mid Y) = H(Y) - H(Y \mid X) = H(X) + H(Y) - H(X,Y)$.*

Par le lemme 3.19 :

**Lemme 3.21** *$I(X;Y) = I(Y;X)$. $I(X;Y)$ est positive ou nulle, nulle si et seulement si $X$ et $Y$ sont indépendantes.*

## 3.3 Secret parfait et équivocation

Shannon [2] a fondé une théorie du secret parfait. On considère que l'on peut chiffrer un message $M$ parmi un ensemble fini $\{M_1, \ldots, M_n\}$, avec des clés parmi un ensemble fini (arbitrairement larges).

**Définition 3.22 (Secret parfait)** *Une algorithme de chiffrement assure le* secret parfait *si et seulement si $Pr_{M,K}[M \mid \{M\}_K = E] = Pr_M[M]$.*

Autrement dit, connaître le chiffré $E$ ne donne pas d'information supplémentaire sur le clair $M$.

Il est équivalent de demander que les variables aléatoires $M$ et $\{M\}_K$ soient indépendantes. Ceci est le cas du masque jetable.

Shannon définit l'*équivocation* (du message $M$) comme étant $H(M \mid \{M\}_K)$, où $M$ et $K$ sont tirés au hasard (selon une certain loi conjointe). C'est le nombre de bits qui nous manquent réellement pour connaître $M$ sachant $\{M\}_K$.

**Explication :** fixons $E = \{M\}_K$ le chiffré observé, et choisissons un code optimal pour la variable aléatoire $M$ conditionnée par $E$ ($p_a = Pr_K[M = a \mid \{M\}_K = E]$ ; notons que l'alphabet est ici l'ensemble de tous les messages $M$). La longueur moyenne du code de $M$ sachant $E$ est $H(M \mid \{M\}_K = E)$, à 1 près (théorème 3.9). On a donc besoin de $H(M \mid \{M\}_K = E)$ bits pour décrire le message inconnu $M$

en moyenne. Lorsque $E$ varie, on a donc besoin de $H(M \mid \{M\}_K)$ bits, à 1 près, pour décrire $M$ en moyenne, et donc un nombre de messages à tester pour trouver $M$ compris de l'ordre de :

$$2^{H(M|\{M\}_K)}$$

avec forte probabilité.

**Exemple 3.23** *Dans le cas du masque jetable, $H(M \mid \{M\}_K) = H(M)$ est maximal. Pas de meilleure stratégie que d'énumérer tous les $M$ possibles par ordre décroissant de probabilités.*

**Exemple 3.24** *Si $\{M\}_K = M$ (chiffrement idiot consistant à émettre le message en clair), $H(M \mid \{M\}_K) = 0$ : $2^0 = 1$ seul message en clair possible à tester. C'est pareil si $M = f(\{M\}_K)$ pour une certaine fonction $f$ de déchiffrement, indépendante de la clé $K$.*

Les exercices suivants quantifient à quel point les chiffrements de César et de Vigenère sont mauvais.

▷ **Exercise 3.25**
On suppose $M$ tiré parmi les mots sur $\Sigma$ de longueur $n$, par un tirage uniforme et indépendant sur les lettres. Calculer $H(M \mid \{M\}_K)$ pour un chiffrement de César. Les clés $K$ seront prises aléatoires uniformes dans $\Sigma$. Comparer avec $H(M)$ (exercice 3.16).

▷ **Exercise 3.26**
Même exercice avec un codage de Vigenère avec clés de $k$ caractères. On supposera la longueur $n$ des messages multiple de $k$.

# 4 Complexité et modèles de sécurité calculatoires

La théorie de Shannon est un début intéressant, notamment parce qu'elle permet de quantifier exactement le niveau de sécurité d'un schéma de chiffrement, mais :

1. ne traite que d'attaquants *passifs* (qui se contentent d'écouter, et n'interagissent pas dans les échanges) ;
2. ne traite que du *secret* (pas authentification, etc.) ;
3. ignore la difficulté *calculatoire* du déchiffrement ou du décryptage (voir discussion sur l'entropie et les codes : le code existe, mais on n'a pas dit comment le trouver par exemple ; illustration : si $X = f(Y)$, alors $H(X \mid Y) = 0$, ce qui dans la théorie de Shannon énonce que $X$ est facile à trouver à partir de $Y$ ; mais $f$ n'est peut-être pas calculable en temps moins de $2^{|Y|}$, ce qui rend le calcul impraticable).

La cryptographie moderne est (à part pour le cas quantique) est fondée sur la notion de *faisabilité* des calculs. On identifie :

faisable = calculable en temps polynomial (en la taille de l'entrée).

Plus précisément, $f\colon \Sigma^* \to \Sigma^*$ est faisable si et seulement s'il existe $k$ tel que pour tout $x$ (dont la taille sera notée $n$), $f(x)$ est calculable en temps $O(n^k)$ sur une machine de Turing randomisée (on a le droit de faire des tirages de bits aléatoires uniformes indépendants).

Une fonction $f$ est *à sens unique* (« one-way ») si et seulement si $f$ est faisable, mais n'a aucun inverse à droite faisable. (Il n'existe pas de fonction $g$ faisable telle que $f(g(y)) = y$ pour tout $y$.)

On modélise les fonctions de chiffrement par la notion suivante.

**Définition 4.1** *Une fonction $f\colon M, K \mapsto \{M\}_K$ de deux arguments est* à sens unique à trappe *(« trapdoor one-way ») si et seulement si :*

1. *$f$ est à sens unique de $\Sigma^* \times \Sigma^*$ vers son image ;*
2. *pour tout $K$, il existe une fonction faisable $g_K$ (la* trappe*, i.e., l'opération de déchiffrement) telle que $f(g_K(M), K) = M$ pour tout $M$ dans l'image de $f(\_, K)$.*

On ne connaît aujourd'hui aucune fonction à sens unique à trappe (prouvée). En fait, on ne sait pas comment prouver qu'un inverse de fonction faisable n'est pas faisable. (Impliquerait certains conjectures difficiles telles que $\mathbf{P} \neq \mathbf{NP}$ par exemple.)

On suppose que certaines fonctions documentées, et calculables, en sont. On s'en convainc par la *cryptanalyse*, c'est-à-dire l'étude de la résistance des schémas de chiffrement à l'intégralité des attaques connues à un temps $t$.

# 5 Chiffrements symétriques modernes

Un chiffrement symétrique est une fonction à sens unique à trappe (définition 4.1) pour laquelle $M, K \mapsto g_K(M)$ est faisable. Autrement dit, on peut déchiffrer efficacement en connaissant juste le chiffré et la clé : la clé de déchiffrement est la même que la clé de chiffrement $K$.

Exemples : Triple DES, AES (voir transparents), TwoFish, Trivium, etc..

**Exemple 5.1** *TEA [3] est un chiffrement très pratique à utiliser, notamment par les pirates (eh oui...) en raison de sa taille. Il chiffre des blocks de 64 bits (un tableau* `m[]` *de deux* `long`*s) avec une clé de 128 bits (le tableau* `key[]` *de quatre* `long`*s). Voici la variante dite « nouvelle » de TEA :*

```
void encipher(unsigned long m[], unsigned long key[])
{
  unsigned long sum = 0, delta = 0x9E3779B9;
  int i;
  for(i = 0; i < 32; i++) {
    m[0] += ((m[1] << 4 ^ m[1] >> 5) + m[1]) ^ (sum + key[sum & 3]);
    sum += delta;
    m[1] += ((m[0] << 4 ^ m[0] >> 5) + m[0]) ^ (sum + key[sum >> 11 & 3]);
  }
}
```

*Autrement dit, soit $\oplus$ l'opération ou exclusif bit à bit (implémentée par ^ ; on notera que $(M \oplus N) \oplus N = M$), soit $p(x) = (16x \bmod 2^{32} \oplus \lfloor x/32 \rfloor)$, on définit les suites :*

$$\begin{aligned}
sum_i &= i \times \texttt{delta} \\
key1_i &= sum_i + \texttt{key[}sum_i \bmod 4\texttt{]} \\
key2_i &= sum_{i+1} + \texttt{key[}sum_{i+1}/2048 \bmod 4\texttt{]} \quad \textit{(« key schedule »)} \\
m0_0 &= \texttt{m[0]} \\
m1_0 &= \texttt{m[1]} \\
m0_i &= m0_{i-1} + [(p(m1_{i-1}) + m1_{i-1}) \oplus key1_i] \\
m1_i &= m1_{i-1} + [(p(m0_i) + m0_i) \oplus key2_i] \qquad (i \geq 1)
\end{aligned}$$

*où toutes les additions sont modulo $2^{32}$. Le chiffrement est $m0_{32}, m1_{32}$.*

▷ **Exercise 5.2**
Comment calcule-t-on l'inverse pour xTEA ? Pour les programmeurs, écrire la fonction de déchiffrement de xTEA.

# 6 Un peu d'arithmétique

**Définition 6.1** *Un* groupe cyclique *est (un groupe isomorphe à) $(\mathbb{Z}/N\mathbb{Z}, 0, +)$, pour un certain $N \in \mathbb{N}$, $N \geq 1$.*

*Un* générateur *$g$ d'un groupe $(G, 1, .)$ est un élément tel que tout $x \in G$ s'écrit $g^n$ pour un certain $n \in \mathbb{N}$.*

**Lemme 6.2** *Un groupe abélien fini est cyclique si et seulement s'il a un générateur.*

*Démonstration.* $\mathbb{Z}/N\mathbb{Z}$ admet 1 comme générateur. Réciproquement, soit $g$ un générateur de $G$. La fonction (dite *exponentielle* en base $g$) $n \in \mathbb{Z} \mapsto g^n$ est un morphisme de groupes surjectif. Posons $N$ le plus petit entier naturel tel que $g^N = 1$. Si $n$ et $n'$ diffèrent par un multiple de $N$, ils ont même exponentielle. Donc l'exponentielle se restreint à un morphisme de groupes de $\mathbb{Z}/N\mathbb{Z}$ sur $G$. Ce morphisme est injectif : si $g^n = g^{n'}$ (disons, avec $n < n'$), alors $g^{n'-n} = 1$, donc $g^{n'-n \bmod N} = 1$. La minimalité de $N$ implique que $n' - n = 0 \bmod N$. L'exponentielle est donc un isomorphisme de groupes entre $G$ et $\mathbb{Z}/N\mathbb{Z}$. □

Si $g$ est un générateur de $G$, on peut former la fonction inverse de l'exponentielle en base $g$ (voir démonstration ci-dessus). On notera que le nombre $N$ est nécessairement l'*ordre*, c'est-à-dire le nombre d'éléments du groupe $G$ (puisque $G \cong \mathbb{Z}/N\mathbb{Z}$).

**Définition 6.3** *Soit $g$ un générateur d'un groupe cyclique $G$. La fonction* logarithme discret *de base $g$ est définie par : $\log_g x$ est l'unique entier $n$ modulo l'ordre $N$ du groupe $G$ tel que $g^n = x$.*

$\mathbb{Z}/N\mathbb{Z}$ a beaucoup de générateurs en général.

**Lemme 6.4** *Un élément $g \in \mathbb{Z}/N\mathbb{Z}$ est un générateur si et seulement s'il est premier avec $N$.*

*Démonstration.* Ici, un générateur est un élément $g$ tel que tout entier modulo $N$ est égal à $kg$ pour un certain entier $k$. Si $g$ est premier avec $N$, le théorème de Bezout nous donne l'existence d'entiers $a$, $b$ tels que $ag + bN = 1$. Donc tout entier $i$ modulo $N$ s'écrit $kg$ modulo $N$, avec $k = ai$. Si $g$ n'est pas premier avec $N$, soit $d$ leur pgcd. Tous les nombres de la forme $kg$ modulo $N$ sont multiples de $d$, donc les non-multiples de $d$ ne peuvent pas s'écrire $kg$ modulo $N$. □

**Définition 6.5** *La* fonction $\varphi$ d'Euler *est définie par : pour tout $N \geq 1$, $\varphi(N)$ est le nombre de générateurs de $\mathbb{Z}/N\mathbb{Z}$.*

C'est donc le nombre d'entiers entre 0 et $N-1$ qui sont premiers avec $N$.

**Lemme 6.6** *Pour tout entier naturel $N$, que l'on écrira sous forme de produit $\prod_{i=1}^{m} p_i^{n_i}$ de puissances de nombres premiers $p_i$ distincts deux à deux,*

$$\varphi(N) = \prod_{i=1}^{m} (p_i - 1) p_i^{n_i - 1}.$$

*Démonstration.* Si $p$ est premier, alors $\varphi(p) = p - 1$ (seul 0 n'est pas premier avec $p$). Plus généralement, pour tout $n \geq 1$, $\varphi(p^n) = (p-1)p^{n-1}$ : les entiers entres 0 et $p^n - 1$ qui ne sont *pas* premiers avec $p^n$ sont ceux qui sont multiples de $p$, et ils sont au nombre de $p^{n-1}$.

On démontre ensuite que si $N$ et $N'$ sont premiers entre eux, alors $\varphi(NN') = \varphi(N)\varphi(N')$, ce qui suffira à impliquer le résultat. Pour ceci, on forme la fonction $f$ qui à tout $k$ premier avec $NN'$ entre 0 et $NN' - 1$ associe le couple $(k \mod N, k \mod N')$, et on va montrer que c'est une bijection.

Comme $k$ est premier avec $NN'$, il est premier avec $N$ et avec $N'$, donc $k$ mod $N$ est premier avec $N$ (et entre 0 et $N-1$) et $k$ mod $N'$ est premier avec $N'$ (et entre 0 et $N'-1$). Réciproquement, comme $N$ et $N'$ sont premiers entre eux, par le théorème de Bezout il existe deux entiers $a$ et $b$ tels que $aN + bN' = 1$. Pour tout $k_1$ premier avec $N$ entre 0 et $N-1$, et pour tout $k_2$ premier avec $N'$ entre 0 et $N'-1$, leur antécedent par $f$ est nécessairement $k = k_2 aN + k_1 bN'$ modulo $NN'$. En effet, si $f(k) = (k_1, k_2)$, alors, modulo $NN'$, on a $k = k(aN + bN') = kaN + kbN' = k_2 aN + k_1 bN'$ puisque $k$ et $k_2$ ne diffèrent que d'un multiple de $N'$, donc $kaN$ et $k_2 aN$ diffèrent d'un multiple de $N'$, et de même pour $kbN'$ et $k_1 bN'$. Et l'on vérifie que, modulo $N$, $k_2 aN + k_1 bN' = k_1 bN' = k_1(1 - aN) = k_1$, et de même, $k_2 aN + k_1 bN' = k_2$ modulo $N'$. □

**Lemme 6.7 (Euler)** *Soit $N \geq 2$. Pour tout entier $x$ premier avec $N$, $x^{\varphi(N)} = 1$ modulo $N$.*

*Démonstration.* On peut d'abord se restreindre à choisir $x$ entre 0 et $N-1$, remplaçant $x$ par $x$ mod $N$. Soit $E$ l'ensemble des entiers entre 0 et $N-1$ qui sont premiers avec $N$, et notons $f$ la fonction qui à tout $e \in E$ associe $xe$ mod $N$. Pour tout $e \in E$, $f(e)$ est dans $E$, sinon $xe$ ne serait pas premier avec $N$, et donc $x$ ou $e$ ne serait pas non plus premier avec $N$. De plus, $f$ est injective. En effet, si $xe = xe'$ modulo $N$, avec $e$ et $e'$ premiers modulo $N$, alors $x(e - e') = 0$ modulo $N$. Comme $x$ et $N$ sont premiers

entre eux, ceci implique que $e - e'$ est un multiple de $N$, ce qui est impossible à moins que $e = e'$, puisque $0 \leq e, e' \leq N - 1$. Donc $f$ est une injection de $E$ (de cardinal $\varphi(N)$) dans $E$ (de même cardinal), c'est donc une bijection.

Calculons le produit $\prod_{e \in E} e$, modulo $N$. Comme $f$ est une permutation, c'est aussi $\prod_{e \in E} f(e)$. Mais ceci vaut $\prod_{e \in E} xe = x^{\varphi(N)} \prod_{e \in E} e$. Donc $(x^{\varphi(N)} - 1) \prod_{e \in E} e$ est nul modulo $N$, donc un multiple de $N$. Or $\prod_{e \in E} e$ est premier avec $N$, donc $x^{\varphi(N)} - 1$ est un multiple de $N$. □

Ceci généralise le petit théorème de Fermat : si $p$ est premier et $x$ est non nul modulo $p$, alors $x^{p-1} = 1$ modulo $p$. On en déduit facilement sa forme usuelle : si $p$ est premier, alors pour tout $x$, $x^p = x$ modulo $p$.

De cette dernière forme, on obtient aussi la variante suivante, dont nous aurons besoin dans le chiffrement RSA.

**Lemme 6.8** *Soit $N \geq 2$, et supposons que $N$ est un produit d'entiers premiers distincts $\prod_{i=1}^{k} p_i$. Pour tout entier $x$, $x^{1+\varphi(N)} = x$ modulo $N$.*

*Démonstration.* Pour chaque $i$, posons $N_i$ le produit des $p_j - 1$ pour $j \neq i$ ; modulo $p_i$ on a : si $p_i$ ne divise pas $x$, alors $x^{1+\varphi(N)} = x^{1+p_i N_i - N_i} = (x^{N_i})^{p_i} x^{1-N_i} = x^{N_i} x^{1-N_i} = x$ par le petit théorème de Fermat ($x^{1-N_i}$ a un sens, comme inverse de $x^{N_i - 1}$ modulo $p$, précisément parce que $x$ est premier avec $p$) ; sinon, $x^{1+\varphi(N)}$ et $x$ sont tous les deux nuls modulo $p_i$. Comme $x^{1+\varphi(N)} - x$ est nul modulo chacun des $p_i$, il l'est encore modulo leur ppcm $N$. □

Le résultat suivant est le plus complexe de tout le cours.

**Lemme 6.9** *Soit $p$ un nombre premier. Le groupe multiplicatif $(\mathbb{Z}/p\mathbb{Z}^*, 1, .)$ est un groupe cyclique d'ordre $p - 1$.*

*Démonstration.* D'abord, c'est un groupe. En particulier, l'inverse de $x$ est l'entier $k$ modulo $p$ tel que $kx = 1$, obtenu par le théorème de Bezout ($kx + k'p = 1$ pour certains $k$, $k'$, puisque $x$ et $p$ sont premiers entre eux).

On appelle *élément primitif* modulo $p$ un entier $x$ entre 1 et $p - 1$ qui est un générateur de $\mathbb{Z}/p\mathbb{Z}^*$. Pour montrer que $\mathbb{Z}/p\mathbb{Z}^*$ est un groupe cyclique, il suffit de trouver un élément primitif modulo $p$. Notons que si le lemme que nous cherchons à démontrer est juste, il y aura beaucoup d'éléments primitifs ($\varphi(p - 1)$ en tout).

On note que comme $\mathbb{Z}/p\mathbb{Z}^*$ est un groupe, $\mathbb{Z}/p\mathbb{Z}$ est un corps commutatif. Comme dans les réels ou les complexes, on peut démontrer le théorème d'interpolation de Lagrange, ce qui implique en particulier que tout polynôme de degré $d$ en une variable, à coefficients dans $\mathbb{Z}/p\mathbb{Z}$, a au plus $d$ racines modulo $p$.

Dans le cas particulier du polynôme $x^d - 1$, on peut dire mieux : si $d$ divise $p - 1$, alors il a exactement $d$ racines modulo $p$. En effet, écrivons $p - 1 = kd$, puis $x^{p-1} - 1 = (x^d)^k - 1 = (x^d - 1)(1 + x^d + x^{2d} + \ldots + x^{(k-1)d})$. Si $x^d - 1$ avait strictement moins de $d$ racines, le côté droit aurait strictement moins de $d + (k - 1)d = kd = p - 1$ racines. Mais le petit théorème de Fermat nous dit que le côté gauche est un polynôme à exactement $p - 1$ racines.

Appelons *ordre* d'un élément $x$ modulo $p$ le plus petit entier $m \geq 1$ tel que $x^m = 1$ modulo $p$.

On note que : $(*)$ si $x^d = 1$ modulo $p$, alors $d$ est un multiple de l'ordre de $x$. En effet, si l'ordre $m$ de $x$ ne divisait pas $d$, en notant $r$ le reste de la division euclidienne de $d$ par $m$, on aurait $x^m = x^r$, contredisant la minimalité de $m$.

On note aussi que : $(\dagger)$ si $x$ est d'ordre $m$ et $x'$ est d'ordre $m'$, et que $m$ et $m'$ sont premiers entre eux, alors $xx'$ est d'ordre $mm'$. Posons $k$ l'ordre de $xx'$. Alors, modulo $p$, $x^{m'k} = x^{m'k}(x'^{m'})^k = (xx')^{m'k} = ((xx')^k)^{m'} = 1$. Donc par $(*)$, $m$ divise $m'k$. Comme $m$ et $m'$ sont premiers entre eux, $m$ divise $k$. De même, $m'$ divise $k$. Donc $mm'$ divise $k$. Réciproquement, on vérifie que $(xx')^{mm'} = (x^m)^{m'}(x'^{m'})^m = 1$ modulo $p$.

Si $d = q^k$ est la puissance $k$ième d'un nombre premier qui divise $p-1$, on a vu que le polynôme $x^d - 1$ avait $d$ racines modulo $p$. L'ordre de chacune doit diviser $d$ par $(*)$, donc les ordres des racines de $x^d - 1$ sont de la forme $q^j$ avec $j \leq k$. Une de ces racines doit avoir pour ordre exactement $q^k = d$, sinon toutes ces racines seraient d'ordre au plus $q^{k-1}$, donc racines du polynôme $x^{q^{k-1}} - 1 \ldots$ mais ce dernier polynôme n'a que $q^{k-1} < d$ racines modulo $p$.

Écrivons $p-1$ sous forme de produit $\prod_{i=1}^n q_i^{k_i}$ de puissances de nombres premiers distincts deux à deux. Pour chaque $i$, soit $x_i$ un élément d'ordre $q_i^{k_i}$. Par $(\dagger)$, $\prod_{i=1}^n x_i$ est un élément d'ordre $\prod_{i=1}^n q_i^{k_i} = p-1$, et est donc un élément primitif modulo $p$. □

Il est facile de trouver un générateur de $\mathbb{Z}/p\mathbb{Z}^*$ : comme il y en a $\varphi(p-1)$, il suffit de tirer au hasard uniformément un nombre entre 1 et $p-1$, avec probabilité $\varphi(p-1)/p$ (proche de 1) il sera un générateur. On recommence tant qu'il n'est pas un générateur, ceci termine avec probabilité 1 (et en $p/\varphi(p-1)$ itérations en moyenne). Pour tester si $x$ est un générateur, il suffit de vérifier que $x^{(p-1)/q} = 1$ modulo $p$ pour tous les diviseurs premiers $q$ de $p-1$.

# 7 Distribution de clé, le protocole de Diffie-Hellman(-Merkle)

But : fabriquer une clé aléatoire partagée entre $A$ et $B$, *sans avoir à la distribuer* (c'est-à-dire à la créer puis à la transmettre à $A$ et à $B$).

$A$ et $B$ se mettent d'accord sur un grand nombre premier $p$, et un générateur $g$ de $\mathbb{Z}/p\mathbb{Z}^*$. Ils peuvent vérifier que $p$ est premier, et que $g$ est un générateur, en temps polynomial probabiliste. Vérifier qu'ils ont le même $p$ et le même $g$, en revanche, nécessite une authentification (voir section 10).

**Définition 7.1 (Diffie-Hellman)** *Le protocole de Diffie-Hellman [4] est :*

1. *$A$ tire $x_A$ au hasard uniformément entre 1 et $p-2$, et envoie $g^{x_A}$ mod $p$ à $B$ ($x_A$ reste privé à $A$) ;*
2. *$B$ tire $x_B$ au hasard uniformément entre 1 et $p-2$, et envoie $g^{x_B}$ mod $p$ à $A$ ($x_B$ reste privé à $B$) ;*
3. *la clé partagée est $K = g^{x_A x_B}$ modulo $p$.*

L'exponentiation modulaire se calcule rapidement (elle est faisable). On utilise la récurrence $M^0 = 1$, $M^{2n} = (M^n)^2$, $M^{2n+1} = (M^n)^2.M$ modulo $p$, et on récurse

sur la présentation en binaire de l'exposant. On note que les calculs intermédiaires ne grossissent pas au-delà de $p-1$, car on calcule modulo $p$.

▷ **Exercise 7.2**
Comment $A$ fait-elle pour calculer $K$, sans avoir connaissance de $x_B$ ? Et $B$ ?

▷ **Exercise 7.3**
Supposons que $\log_g$ soit une fonction faisable. Montrer qu'alors Diffie-Hellman est cassé, au sens où il existe une fonction faisable (exploitable par un attaquant) qui prend les messages publics $g^{x_A}$ et $g^{x_B}$ (modulo $p$) du protocole de Diffie-Hellman, et retourne la clé partagée.

Le meilleur algorithme (« générique », voir plus bas) pour calculer $\log_g$ dans un groupe cyclique est l'algorithme pas-de-bébé-pas-de-géant, en temps $O(\sqrt{p})$. C'est un algorithme non polynomial, la taille de l'entrée étant une exponentielle de $p$ : le logarithme discret dans $\mathbb{Z}/p\mathbb{Z}^*$ est un problème réputé calculatoirement difficile.

Le *modèle générique* est un modèle de calcul où les seules opérations permises sur les éléments d'un groupe $G$ (ici, $\mathbb{Z}/p\mathbb{Z}^*$) sont les opérations du groupe. Par exemple, on s'interdit de convertir $n \in \mathbb{Z}/p\mathbb{Z}^*$ en l'unique entier (encore noté $n$) de $\mathbb{Z}$ entre $0$ et $p-1$ pour faire des calculs dans $\mathbb{Z}$.

Victor Shoup a montré que : 1. l'algorithme pas-de-bébé-pas-de-géant est optimal dans le modèle générique, et 2. la meilleure attaque sur Diffie-Hellman dans le modèle générique est aussi en complexité asymptotiquement proportionnelle à $\sqrt{p}$. Dans le modèle générique, Diffie-Hellman est donc aussi difficile que le calcul du logarithme discret.

▷ **Exercise 7.4**
Utiliser Diffie-Hellman pour fabriquer un système de chiffrement de type masque jetable. (Le mécanisme envisagé est appelé à *clés éphémères*.) Est-il inconditionnellement sûr ?

# 8 Génération de nombres pseudo-aléatoires

Un générateur pseudo-aléatoire est un algorithme déterministe en temps polynomial prenant en entrée une graine $s \in \Sigma^*$ (courte, aléatoire), et fournissant une suite $x_0(s), x_1(s), \ldots, x_n(s)$ de mots de $\Sigma^*$ (de longueur totale grande devant $|s|$), qui doit ressembler le plus possible à une suite aléatoire.

Par « ressembler », on entend que la suite soit *incompressible*.

En d'autres termes, c'est une fonction faisable $f$ de $\Sigma^m$ vers $\Sigma^n$, avec $m \ll n$, et telle que la loi de $f(X)$ ($X$ variable aléatoire uniforme sur $\Sigma^m$) soit incompressible.

C'est impossible de façon inconditionnelle : $H(f(X) \mid X) = 0$, et $f(X)$ est toujours compressible au même nombre de bits que la source $X$. On demande donc qu'on ne puisse pas trouver de code dont la fonction de décodage soit calculable (complexité de Kolmogorov), voire calculable en temps polynomial.

**Exemple 8.1** *On utilise parfois le générateur pseudo-aléatoire* linéaire congruentiel *:* $x_0(s) = s$, $x_{n+1}(s) = ax_n(s) + b \mod m$, *avec* $a$ *premier avec* $m$.

*Il est très facilement* prédictible *: connaissant n'importe quel* $x_i(s)$, *on connaît* $x_{i+1}(s)$, $x_{i+2}(s)$, *etc. Même si on ne connaît pas* $a$ *et* $b$ *initialement, il suffit de connaître*

*$x_0(s)$, $x_1(s)$ et $x_2(s)$ pour connaître tout le reste de la suite ($a = (x_2(s)-x_1(s))/(x_1(s)-x_0(s))$, $b = x_2(s) - ax_1(s)$).*

Un générateur prédictible est toujours compressible. Par exemple, on peut compresser le générateur linéaire congruentiel en juste les données $s$, $a$, $b$, $m$ (fonction de décodage laissée en exercice).

Réciproquement, un générateur compressible est prédictible : la prédiction est effectuée par le décodeur.

Un bon générateur (aléatoire ou) pseudo-aléatoire est indispensable, par exemple dans Diffie-Hellman pour tirer $x_A$ et $x_B$. Si l'attaquant casse le générateur (prédit $x_A$, ou $x_B$), il peut reconstruire la clé partagée $g^{x_A x_B}$ sans avoir à casser le logarithme discret.

**Définition 8.2 (Blum-Micali)** *Soit $p$ premier, $g$ un générateur de $\mathbb{Z}/p\mathbb{Z}^*$, $n \ll p$. À partir de la graine $s$, on pose $z_n(s) = s$, $z_{i-1} = g^{z_i} \bmod p$. Le générateur sort les bits de poids fort de $z_0(s)$, $z_1(s)$, …, $z_n(s)$ (dans l'ordre inverse), autrement dit $x_i(s) = 1$ si et seulement si $z_i(s) \geq (p-1)/2$, $x_i(s) = 0$ sinon.*

**Théorème 8.3** *Si le générateur de Blum-Micali est prédictible en temps polynomial, alors on peut calculer $\log_g$ en temps polynomial.*

On n'en donnera pas la démonstration, voir [5].

Ceci est un argument par *réduction* : casser Blum-Micali est au moins aussi difficile que de casser le logarithme discret (un problème réputé dur).

Un autre générateur est celui de Blum-Blum-Shub : $x_n(s) = s$, $x_{i-1}(s) = x_i(s)^2 \mod N$, où $N$ est un produit de deux grands nombres premiers $p$, $q$ égaux à 3 modulo 4, et $s$ est premier avec $N$. La sécurité de ce générateur repose sur la difficulté de factoriser $N$ en $pq$.

Il existe des générateurs pseudo-aléatoires plus pratiques, comme par exemple Yarrow (`http://www.schneier.com/yarrow.html`). Fonctionnement de base : $x_i(s) = \{i\}_K$, où $K = f(s)$. Fonctionne bien, sécurité par réduction à la sécurité du mécanisme de chiffrement utilisé.

# 9 Chiffrement asymétrique

En chiffrement asymétrique, on chiffre avec une clé $K$ (la clé *publique*), mais l'on utilise une autre clé, $K^{-1}$ (la clé *privée*), pour déchiffrer.

Ceci résout (en principe) le problème de distribution de clé, ou de génération de clé partagée : $B$ fabrique une paire $(K_B, K_B^{-1})$, publie $K_B$ et garde $K_B^{-1}$ pour lui ; pour chiffrer $M$ à destination de $B$, $A$ chiffre $M$ avec $K_B$ : seul $B$ pourra alors déchiffrer $\{M\}_{K_B}$, car seul $B$ possède $K_B^{-1}$.

**Définition 9.1 (Rivest-Shamir-Adleman)** *Le* chiffrement RSA *[6] fonctionne comme suit :*

1. *$B$ tire au hasard deux grands nombres premiers $p$, $q$, pose $N = pq$ (le* module*) ;*
2. *$B$ tire au hasard $d$ premier avec $\varphi(N) = (p-1)(q-1)$, entre 2 et $\varphi(N) - 1$ ;*

3. *$B$ calcule $e$ tel que $de = 1$ modulo $\varphi(N)$ (Bezout ; algorithme d'Euclide étendu) ;*
4. *La clé publique de $B$ est $K_B = (N, e)$, sa clé privée est $K_B^{-1} = d$ ;*
5. *Pour chiffrer à destination de $B$, $A$ calcule $\{M\}_{K_B} = M^e$ mod $N$ ;*
6. *Pour déchiffrer $M'$, $B$ calcule $M'^d$ mod $N$.*

▷ **Exercise 9.2**
Montrer que le déchiffrement de $\{M\}_{K_B}$ par $B$ redonne effectivement $M$.

Il existe d'autres algorithmes de chiffrement à clé publique. Par exemple, le suivant repose sur la sécurité du logarithme discret. Comparer avec Diffie-Hellman.

**Définition 9.3 (Chiffrement El Gamal)** *Le* chiffrement El Gamal *fonctionne comme suit :*

1. *$B$ tire au hasard un grand nombre premier $p$, et un générateur $g$ de $\mathbb{Z}/p\mathbb{Z}^*$ ;*
2. *$B$ tire au hasard $x$ entre $1$ et $p-2$ : $x$ est sa clé privée $K_B^{-1}$, et sa clé publique est $h = g^x$ modulo $p$ ;*
3. *Pour chiffrer à destination de $B$, $A$ tire $k$ au hasard entre $1$ et $p-2$, et calcule $\{M\}_{K_B} = (c_1, c_2)$ où $c_1 = g^k$, $c_2 = Mh^k$ modulo $p$.*

▷ **Exercise 9.4**
Comment $B$ déchiffre-t-il le message de $A$ en chiffrement El Gamal ? Donner une formule explicite en fonction de $c_1$, $c_2$, et la clé privée $K_B^{-1} = x$. Comparer avec l'exercice 7.4. En déduire que si $\log_g$ est faisable, alors le chiffrement El Gamal est attaquable.

D'autre part, on peut aussi remplacer le groupe $\mathbb{Z}/p\mathbb{Z}^*$ par un autre groupe cyclique. Si l'on prend par exemple le groupe engendré par un élément d'ordre grand dans le groupe d'une courbe elliptique sur $\mathbb{Z}/p\mathbb{Z}$, on obtient les variantes de RSA, El Gamal, etc., dites à courbes elliptiques.

# 10 Signatures, authentification

La *signature RSA* : cette fois-ci, $A$ génère une paire $(K_A, K_A^{-1})$, signe avec sa clé privée $K_A^{-1}$. $B$ vérifie la signature en vérifiant que le signé se déchiffre avec la clé publique $K_A$ de $A$.

Plus précisément, $A$ envoie $\{M\}_{K_A^{-1}}$ et $M$. $B$ vérifie que le signé $\{M\}_{K_A^{-1}}$ se déchiffre avec $K_A$ et fournit exactement $M$. Si oui, le message $M$ vient bien authentiquement de $A$.

La signature RSA est un algorithme de signature qui permet de récupérer le texte signé. D'autres algorithmes de signature permettent juste de vérifier la validité de la signature par $A$ sans révéler le clair $M$.

On notera $[M]_{K_A^{-1}}$ le signé de $M$ par la clé privée $K_A^{-1}$. Ce ne sera pas toujours le résultat d'un chiffrement, comme dans le cas de RSA. Voir ci-dessous.

Le procédé suivant ne doit pas être confondu avec le chiffrement El Gamal.

**Définition 10.1 (Signature El Gamal)** *La* signature El Gamal *fonctionne comme suit :*

1. *$A$ tire au hasard un grand nombre premier $p$ et un générateur $g$ de $\mathbb{Z}/p\mathbb{Z}^*$ ;*

2. *$A$ tire $x$ au hasard entre $2$ et $p-1$, pose $K_A^{-1} = x$ (clé privée), $K_A = g^x \bmod p$ (clé publique) ;*
3. *Pour signer $M$, $A$ tire $k$ au hasard entre $1$ et $p-2$ premier avec $p-1$, calcule $r = g^k \bmod p$, $s = (M - rx)/k \bmod (p-1)$ (et recommence jusqu'à obtenir un $s$ non nul) : le signé est $[M]_{K_A^{-1}} = (M, s)$.*

On dira qu'un algorithme de signature est *cassé* (vulnérable à l'existence de fausses signatures, ou *existential forgery* en anglais) si et seulement un attaquant peut produire un message qui est aussi une signature valide venant de $A$ (d'un message quelconque, pas forcément $M$).

▷ **Exercise 10.2**
Comment fait $B$ pour vérifier la validité de la signature El Gamal, étant donnés les paramètres publics $K_A = g^x$ modulo $p$, $s$, et $M$ ? Montrer que ce schéma est cassé si $\log_g$ est faisable.

▷ **Exercise 10.3**
La *signature DSA* fonctionne comme suit :

1. $A$ tire au hasard deux grands nombres premiers $p$ et $q$, avec $p-1$ multiple de $q$ ; on pose $z = (p-1)/q$ ;
2. $A$ tire $x$ au hasard entre $1$ et $q-1$, c'est sa clé privée $K_A^{-1} = x$ ;
3. $A$ tire $h$ au hasard entre $2$ et $p-1$, calcule $g = h^z$ modulo $p$, $y = g^x$ modulo $p$ : la clé publique $K_a$ est $(p, q, g, y)$ ;
4. Pour signer $M$, $A$ tire $s$ au hasard entre $2$ et $q-1$, calcule $s_1 = (g^s \bmod p) \bmod q$, $s_2 = (M + s_1 x)s^{-1} \bmod q$ ; la signature est $[M]_{K_A^{-1}} = (M, s_1, s_2)$.

Comment peut-on vérifier que le signé $(s_1, s_2)$ vient bien authentiquement de $A$ en signature El Gamal ? Exprimer le résultat en fonction des seuls paramètres publics $p$, $q$, $g$, $y$, $s_1$, $s_2$, $M$. Indication : calculer $w = s_2^{-1} \bmod q$, $u_1 = Mw \mod q$, $u_2 = s_1 w \bmod q$, $v = (g^{u_1} y^{u_2} \bmod p) \bmod q$.

Aucun de ces procédés de signature ne garantit le *secret* de $M$. Pour ceci, une astuce standard est de chiffrer le signé, c'est-à-dire de calculer $\{[M]_{K_A^{-1}}\}_{K_B}$.

Mais ceci ne suffit pas, voir cours de Stéphanie Delaune.

# Références


[1] Claude E. Shannon (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27 :379–423, 623–656.

[2] Claude E. Shannon (1949). Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4) :656–715.

[3] Roger Needham, David Wheeler (1997). xTEA. *Phrack* 63. Linenoise, paper 3.

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

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

[6] Ron Rivest, Adi Shamir, Len Adleman (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2) :120–126.